Handling Updates and Failures: Dynamic Graph Algorithms and Distributed Computing on Dynamic Networks

by

Antoine El-Hayek

July, 2026

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Committee in charge:

Georgios Katsaros, Chair
Monika Henzinger
Vladimir Kolmogorov
Claire Mathieu

The thesis of Antoine El-Hayek, titled *Handling Updates and Failures: Dynamic Graph Algorithms and Distributed Computing on Dynamic Networks*, is approved by:

Supervisor: Prof. Monika Henzinger, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Prof. Vladimir Kolmogorov, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Prof. Claire Mathieu, CNRS, Paris, France

Signature: _____

Defense Chair: Prof. Georgios Katsaros, ISTA, Klosterneuburg, Austria

Signature: _____

Signed page is on file

ISTA Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I accept full responsibility for the content and factual accuracy of this work, including the data and their analysis and presentation, and the text and citation of other work.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

I declare that no Gen AI tools or AI systems were used in the writing or content generation process in this thesis. All analyses, predictions, and written content were derived exclusively from traditional academic methods and authors individual skillset. The credibility of this original study is supported by the authors commitment to good scientific practices and academic standards. AI use was limited to proofreading.

Signature: _____

Antoine El-Hayek
July, 2026

Signed page is on file

Abstract

In this thesis, we took a look at networks, and more specifically, at networks that change over time, whether those are networks in the *distributed algorithms* sense of the word, or the *graph algorithm* sense.

In distributed algorithms, we looked at two main problems. First, the broadcast problem: given n agents, each agent is tasked to forward a (unique) message to every other agent. Agents collaborate and can copy and forward all messages they have received up until that point. Broadcast is achieved when one agent has successfully broadcast its message to everyone else. We studied the case where the communication network is controlled by an adversary, under the condition that the graph is rooted in every round of communication. We show that the adversary can delay broadcast for at most $\left\lceil (1+\sqrt{2})n \right\rceil$ rounds, improving on the $O(n \log\log n)$ previous upper bound [68], and asymptotically matching the $\sim 1.5n$ lower bound [129].

We then looked at the stochastic version of the problem: here, the adversary – parametrized by k where $k = 0$ signifies that the adversary has no control, and $k = n$ that the adversary has full control – can choose parts of the graph, and the graph is then completed stochastically. Here, we are able to look at a stronger version of broadcast: instead of having n messages trying to be broadcast in parallel, we can assume that only one message needs to be broadcasted. We show the bound $\Theta(k + \log n)$.

Then, we looked at undecided states dynamics in population protocols: given a population of n agents, where each initially holds an opinion among k different ones. In each round, two agents are chosen uniformly at random, and can interact. If they have different opinions, they forget their opinions and become undecided. If one of them is undecided while the other has an opinion, they undecided agent copies they opinion of the decided one. The question is then, how many interactions does it take for the whole population to share the same opinion? We show a $\Omega(kn \log \frac{\sqrt{n}}{k \log n})$ lower bound for any $k = o\left(\frac{\sqrt{n}}{\log n}\right)$. This is tight for any $k \leq n^{\frac{1}{2}-\epsilon}$, where $\epsilon > 0$ can be any small constant, matching the known $O(kn \log n)$ upper bound for $k = O\left(\frac{\sqrt{n}}{\log^2 n}\right)$ [9].

Finally, in dynamic algorithms, we study the minimum cut problem: we are given a graph, whose vertex set we want to partition into two subsets such that the number of edges crossing from one subset to the other is minimized. Then, the graph can be updated via edge insertions or deletions, and we must update the solution without recomputing everything from scratch. We present an exact fully-dynamic minimum cut algorithm that runs in $n^{o(1)}$ deterministic update time when the minimum cut size is at most $2^{\Theta(\log^{3/4-c} n)}$ for any $c > 0$, improving on the previous algorithm [89] whose minimum cut size limit is $(\log n)^{o(1)}$. Using sparsification and randomization techniques, we are able to extend this to all values of the minimum cut in weighted graphs, at the cost of a $(1 + o(1))$-approximation ratio.

Acknowledgements

First and foremost, I would like to thank Prof. Monika Henzinger, my supervisor, who worked with me, advised me, gave me invaluable advice, guided me, read my drafts, assisted to my practice talks, introduced me to the Graph-theoretic community, and invited me for lunches, dinners, ice creams for the last 5 years of my life, and the first 5 years of my career.

I would also like to particularly thank Prof. Stefan Schmid, who effectively acted as my co-supervisor, introducing me to the distributed computing community, inviting me over to his new group in TU Berlin, and overall giving me another point of view on the world of research.

I thank Prof. Gramoz Goranci, for acting as my fast-fleeting co-supervisor during my stay at the University of Vienna. His advice on how to tackle research proved particularly helpful.

Thank you to my past and current collaborators: Prof. Robert Elsässer, Prof. Jason Li, Prof Kathrin Hanauer, Dr. Gregor Bankhamer, Julien Dallot, and Tom-Lukas Breitkopf.

I thank Prof. Claire Mathieu, and Prof. Vladimir Kolmogorov, for reviewing my thesis, and attending the defense. Thank you for the advice on how to move forwards with my research career.

Thank you to Prof. Georgios Katsaros for chairing my defense.

Thanks to Lara, Sricharan, Eva, Sophia, and Ali for the board game night. To David, who was here during the hardest times. To the Martins and their involved stories. To Max to introducing us to the hackerspace. To Bardiya, Roodabeh, Anamay, Joel and David for the schedule hints. To Kathie and her skills in community building. To Sarah, Laurane, Lorenzo, Léo, and Val, for taking me along adventure I would have never taken on otherwise. Thanks to all members of the University of Vienna choir, who made me sing in the Musikverein. Thanks to François, Amélia, Anne, Armelle, Etienne, Jean-Yves, Alexis, Benoît, Diane, Cal, Raphaël, Liza, Charles, Chloé, Julie, Quentin, Yanis, Alexandre, Arno, Anne, Fabien, Arthur, Sophie, and Mathilde, for the wonderful campaign. To Amine, for hosting me in Paris on my long train trips. To Gabriel, for helping out on my defense, and everything else.

Thanks to my aunt Bénédicte, for managing situations always more hectic.

Thanks to my sister Tara, always hopeful, even in a country where it's harder every single day.

Thanks to my dad, who never shies away from saying how proud he is.

Thanks to my mom, who loved seeing me start my PhD, and would have loved to see me finish it.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (MoDynStruct, No. 101019564)

"The Design and Evaluation of Modern Fully Dynamic Data Structures"  , from the Austrian Science Fund (FWF) grant DOI 10.55776/I5982 "Static and Dynamic Hierarchical Graph Decompositions", and from the Austrian Science Fund (FWF) and netIDEE SCIENCE project P 33775-N, "Fast Algorithms for a Reactive Network Layer".

About the Author

Antoine El-Hayek completed a Licence (BSc equivalent) in Mathematics at Université Paris-Saclay in 2017 and a MSc in Mathematics and Foundations of Computer Science at the University of Oxford and École polytechnique in 2021. He then started his PhD at the University of Vienna under the supervision of Prof. Monika Henzinger, and joined ISTA in October 2024 as her group moved there. His main research interests include dynamic graph algorithms and dynamic networks, as well as distributed computing. During his PhD studies, Antoine also presented his research results at many high-impact conferences, namely SODA and PODC.

List of Collaborators and Publications

Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Brief Announcement: Broadcasting Time in Dynamic Rooted Trees is Linear". In: *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*. Ed. by Alessia Milani and Philipp Woelfel. ACM, 2022, pp. 54–56. DOI: 10.1145/3519270.3538460. URL: https://doi.org/10.1145/3519270.3538460 re-used in part in Chapter 3.

Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Asymptotically Tight Bounds on the Time Complexity of Broadcast and Its Variants in Dynamic Networks". In: *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*. ed. by Yael Tauman Kalai. Vol. 251. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 47:1–47:21. DOI: 10.4230/LIPICS.ITCS.2023.47. URL: https://doi.org/10.4230/LIPIcs.ITCS.2023.47 re-used in full in Chapter 3.

Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Broadcast and Consensus in Stochastic Dynamic Networks with Byzantine Nodes and Adversarial Edges". In: *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*. Ed. by Dan Alistarh. Vol. 319. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 21:1–21:15. DOI: 10.4230/LIPICS.DISC.2024.21. URL: https://doi.org/10.4230/LIPIcs.DISC.2024.21 re-used in full in Chapter 4.

Antoine El-Hayek, Monika Henzinger, and Jason Li. "Fully Dynamic Approximate Minimum Cut in Subpolynomial Time per Operation". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*. Ed. by Yossi Azar and Debmalya Panigrahi. SIAM, 2025, pp. 750–784. DOI: 10.1137/1.9781611978322.22. URL: https://doi.org/10.1137/1.9781611978322.22 re-used in part in Chapter 6.

Antoine El-Hayek, Robert Elsässer, and Stefan Schmid. "An Almost Tight Lower Bound for Plurality Consensus with Undecided State Dynamics in the Population Protocol Model". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2025, Hotel Las Brisas Huatulco, Huatulco, Mexico, June 16-20, 2025*. Ed. by Alkida Balliu and Fabian Kuhn. ACM, 2025, pp. 532–540. DOI: 10.1145/3732772.3733505. URL: https://doi.org/10.1145/3732772.3733505 re-used in full in Chapter 5.

Antoine El-Hayek, Monika Henzinger, and Jason Li. "Deterministic and Exact Fully-dynamic Minimum Cut of Superpolylogarithmic Size in Subpolynomial Time". In: *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*. Ed. by Kasper Green Larsen and Barna Saha. SIAM, 2026,

pp. 613–663. DOI: 10.1137/1.9781611978971.25. URL: https://doi.org/10.1137/1.9781611978971.25 re-used in full in Chapter 6.

Table of Contents

List of Figures

List of Definitions

CHAPTER 1

Introduction

It is two in the afternoon. You are sitting in your office. Your smartphone is in your pocket, or on your desk. It is wirelessly connected to the nearest cell phone tower via your favourite cellular network technology, say 5G. Other people around the world also have cell phones, each connected to a tower. The towers are then connected to data centers which form the backbone of the cellular network technology. You can represent this with a graph: every phone, antenna, or data center is represented by a node, and every direct connection between two of them is represented by an edge. You'd need a lot of ink to draw that out.

It is now nine in the evening. You are back home, and are thinking again about that graph. However, your home is far away from your workplace, so the antenna your phone connects to is probably different now than it was earlier in the day. Your graph has changed over time. It is *dynamic*.

It is probably time to relax, however, and forget about graphs. You go on your favourite social media, say Mastodon. There are people you follow and others who follow you. It looks like you cannot really avoid it: there is a graph again. Nodes are the different accounts, and there is an edge between two accounts if one follows the other. This graph is *directed*. It is also dynamic, and you have direct power over it: you just followed someone new. There is now a new edge in the graph.

Ten. You go make yourself a cup of tea. You plug in the kettle. That's also a graph, isn't it? The graph of all electrical appliances and the electricity grid. You turn the kettle on. Well, you try to. It doesn't work. It looks like somewhere along the cable to the power outlet you're using, there is a *failure*. Thankfully, your electricity grid is resilient, and you can just use another power outlet.

Eleven. You go to sleep. You reflect on all the graphs you saw today, and how they are all changing over time. They seem to be everywhere. Maybe studying them would make for a nice PhD topic.

Understanding specific structural properties of dynamic networks is challenging, but they are efficient ways of communication.

1.1 Broadcast in the heard-of model

Here is one way to model a network that changes over time: consider that time is discrete, call every time step a *round*, and show, at each round, the snapshot of the graph at that particular point in time.

In this model, we look at a fundamental problem: Broadcast, where a node of the network has a message it wants to forward to everyone, possibly by forwarding it through intermediate nodes. This problem has been widely studied in different models [81].

A natural question arises: how many rounds does Broadcast need for it to happen, in the worst case? Well, it depends on the restrictions we impose on our graph: if the graph can be empty all the time, then Broadcast can be delayed indefinitely. If the graph is complete, then it takes one round. The interesting question is then to understand the tradeoff between restrictions and speed of Broadcast.

We model the restrictions by classes of graphs: Assume you have a set $\mathcal{A}$ of available graphs, and in each round, an adversary, whose goal is to delay broadcast as much as possible, can choose a graph independently from all other rounds. How long can such an adversary delay broadcast? Different classes of graphs lead to different Broadcast times [46, 47, 34, 129, 68].

In Chapter 3, we look at the class of graphs that is the least restrictive but still leads to broadcast: Trees. Indeed, if the graph is allowed to be disconnected, there is no hope of broadcast. In fact, we go one step further, by slightly modifying the definition of broadcast, we only look at directed rooted trees, with edges directed away from the root. Here, starting the message transmission from only one node does not guarantee broadcast, as this node could be the leaf at every round. However, if a different message starts at every node, and we declare Broadcast once *one of those messages* has reached everyone, then one can guarantee broadcast again.

While this definition of Broadcast might seem overly complicated, it turns out to be the exact definition one needs to better analyze Consensus, another very fundamental problem. Indeed, Broadcast time in this form is proven to be a lower bound for Consensus [68].

In Chapter 4, we look at the variant where the adversary is stochastic, that is, the graph is chosen partially by an adversary, partially at random. We do a smooth analysis: using a parameter k, we can model a wide range of regimes where the adversary has no control over the graph ($k = 0$), to a regime where the adversary controls most of the graph ($k \sim n$).

1.2 Relative majority in population protocols

Another way to look at a dynamic network, in the setting where the network is a series of interactions between individual nodes, is to simply list the interactions one by one, one after the other. This is exactly what population protocols [13] looks at: given n different agents, at each time step, two agents are able to interact, update their *state* (or internal memory),

then go their separate ways. The point of population protocols is to think that every agent is a mobile agent with very limited memory, like ants[1] moving around, possibly erratically, and interacting when they meet. Their goal is to solve a problem together, when each of them runs a simple algorithm locally, the same algorithm running for every agent: we call this algorithm the *protocol*.

There are two main questions when it comes to population protocols: The first is, for a given problem, what is the optimal space complexity protocol to solve the problem? In that case, we assume that every edge appears infinitely many times. The second question is to find to the time it takes to solve a problem. Here, simply assuming that all edges appear infinitely often is not enough, as one could have the same edge over and over again initially, which would lead to very limited progress, before stating to have all the other edges more frequently. Instead, we consider the model where at each time step, the edge is chosen uniformly at random, independently form other rounds. Once these questions are understood, one usually tries to find different trade-offs between time and space.

Two main problems studied in the population protocol model are the *leader election* [6, 52, 4] problem and the *majority* [4, 20, 5, 51] problem. In the leader election problem, as stated in the name, the goal of the agents is to choose a leader among themselves. In the majority problem, each agent is initially given as an input an *opinion*, either 0 or a 1, and the agents must work together to find out which opinion was initially the majority one.

While these two problems are now well understood, we shifted our attention to a natural generalization of the majority problem: the relative majority problem [72, 16, 15, 9]. In this case, the initial opinion of agents is not restricted to 0 or 1 anymore, but can take any integer value between 1 and k for a given k.

There is a very natural Monte-Carlo protocol to solve this problem: the Undecided States Dynamics [9]. In this protocol, when two agents with different opinions meet, they forget their opinion, and become undecided. When an undecided agent meets an agent with an opinion, it mimics that agent's opinion. While this protocol is not guaranteed to output the correct result after convergence, surprisingly, it does converge with high probability as soon as the additive difference between the most popular opinion and the second most popular one is $\Omega(\sqrt{n \log n})$. Beyond the hope of solving the relative majority problem, this protocol is also interesting as it can be seen as a way to model opinions in a society.

In Chapter 5, we take a closer look at this protocol, in particular, we show a lower bound on its convergence time.

1.3 Fully-Dynamic Minimum Cut

The final model for dynamic networks we look at is from the fully-dynamic algorithms perspective. Here, there is a graph that changes over time, probably a graph that represents data of some sort, like the relationships between accounts on a social media platform. The graph is then modeled as an initial graph, and a series of insertion and deletions to the edges as the graph evolves. The goal is then to develop an algorithm that runs on the initial graph to solve

[1] Ants also interact by leaving pheromones on their paths, which makes them move less erratically, but we will ignore this ability here.

a specific problem, but which is also able to handle the updates as they arrive to maintain the solution to the problem, without rerunning the algorithm from scratch after each update.

The three main goals for dynamic algorithms is to reduce the *update time*, the *query time*, and the *recourse* of the algorithms. The update time captures how much time should the algorithm be given after it has received an update to update its internal data structure. If the running time of a *static*[2] algorithm is $O(T)$, the goal is to find an algorithm whose update time is $o(T)$, as otherwise one would simply run the static algorithm after every update. The query time of an algorithm is the time it needs, when a query is made, to output a solution – or a series of changes to be made to the previously output solution. Again, for a problem where a static algorithm of $O(T)$ running time exists, one should aim for $o(T)$ query time. The recourse is the number of changes per update the algorithm has to make to the solution. Recourse is obviously[3] upper-bounded by the update time or the query time, but one could have algorithms where the upper bound on recourse is significantly smaller than either of them. In fact, it turns out to be a very crucial feature when one wants to use one or many dynamic algorithms as a subroutine to another one, as the recourse of the first algorithm is often the number of updates to the second one, and thus one wants to avoid the exponential blow-up.

Many, many problems have been studied in this model. Some examples include Connectivity [84, 90, 124, 88, 105], Minimum Spanning Tree [128], Matching [17, 79, 120, 23], Flow problems [71, 27], Expander Decomposition [77, 118], and many more. We focused on the Minimum Cut problem, where the goal is to partition the graph into two subgraphs, minimizing the number of edges that cross from one one set of the partition to the other. This problem has been widely studied in the static setting and is known to be solvable in near-linear time [93, 91, 95, 82], and also in the dynamic setting [83, 123, 76, 89, 126], but none of the state of the are algorithms were able to maintain a $(1 + o(1))$-approximation of the minimum cut in sublinear time with no restrictions on the graph nor the updates, which is what we achieved in our work.

1.4 Organization

In the next chapter, I give the full definition and an overview of the techniques in each of the projects covered in this thesis. In Chapter 3, we discuss Broadcast in rooted trees with a deterministic adversary, and look at the case of the stochastic adversary in Chapter 4. We then discuss the Undecided States Dynamics in Population Protocols in Chapter 5, and finally we give a fully-dynamic algorithm to solve the Minimum Cut problem in Chapter 6.

1.5 A note on the carbon footprint of this thesis

During my PhD over the last few years, I got to attend conferences and present my work at Salerno, Cambridge MA, Prague, Berkeley, Paris, Lyon, Geneva, Warsaw, Madrid, New Orleans, Zurich, Liverpool, Huatulco and Vancouver. While I did my best to use the train

[2] A static algorithm is a "normal" algorithm, as opposed to *dynamic*: It cannot handle updates.
[3] most of the time

when possible, the flights alone I took averaged to roughly 2.7 tons of CO_2 per year[4]. For comparison, by the Paris agreements, we should aim for 1.5 tons of CO_2 per person per year *for all human activities* if we want to keep the planet below $1.5°C$ of global warming, or 2 tons of CO_2 per person to stay under the $2°C$ mark.

[4]To get this estimate, I got the CO2 emission estimations from atmosfair.de for round-trip flights from Vienna to Salerno, Cambridge MA, Berkeley, New Orleans, Huatulco and Vancouver, then divided by the number of years since the beginning of my PhD

CHAPTER 2

Overview of Chapters

In this chapter, I give an overview of each of the upcoming chapters, highlighting my contribution to every project.

2.1 Broadcast in Dynamic Trees and Forests with a Deterministic adversary

2.1.1 Problem Definition and Results

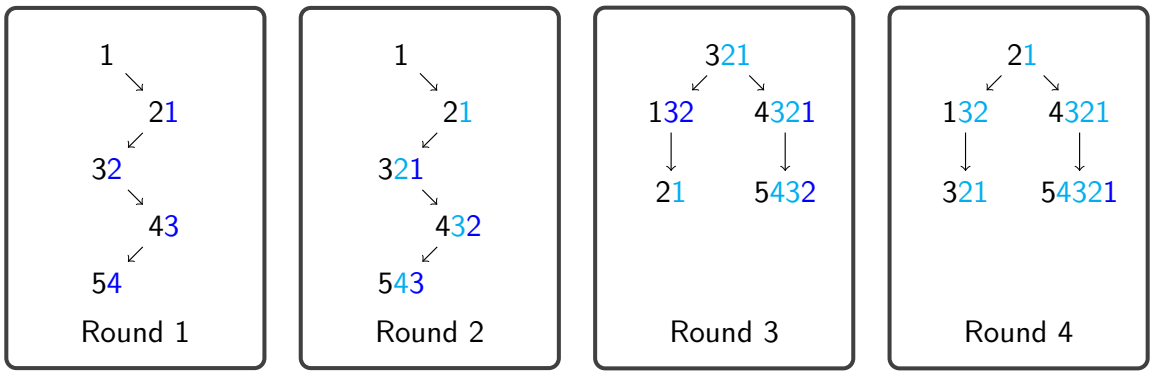


Figure 2.1: The broadcast problem. Black numbers represent the processes' IDs, dark blue numbers represent the messages received in the current round, light blue numbers represent messages received in previous rounds. Broadcast is achieved in round 4 as message "2" (as well as messages "1" and "3") was received by all processes.

We are given n processes, each with a different ID in $\{1, \dots n\}$. Each process initially holds a message, identified by its ID. The goal is for one process to *broadcast* its message, that is, to forward it to everyone else. Communication happens in rounds, where in each round the communication is represented by a directed graph, where each process can send as many

messages along its outgoing edges before receiving the messages sent along its ingoing edges. Processes can make copies and store as many messages as needed, and forward messages from processes it has received in previous rounds.

The communication graph in each round is chosen by an adversary, whose goal is to delay broadcast as much as possible, under the restriction that the graph must be rooted in each round.

A representation of the model can be found in Figure 2.1.

The question is then: for how many rounds can the adversary delay broadcast? An immediate result is $\Omega(n)$: the adversary can choose the chain $1 \to 2 \to \cdots \to n$ for every round. The message from the root of the chain needs to travel down to the leaf, and that takes $n-1$ rounds. Notice that the adversary cannot delay broadcast further by swapping the chain for the chain $n \to n-1 \to \cdots \to 1$ at round $n-1$, as broadcast of the message 2 would be achieved.

We showed a $O(n)$ upper bound that match this lower bound, improving on the $O(n \log\log n)$ upper bound by Függer, Nowak, and Winkler [68].

2.1.2 The Three Representations

One main tool to better understand the problem is to represent the problem in multiple ways. In this case, one of the representations is the dual of the other, and thus, any result that is obvious in one representation gives a dual result that is not that obvious in the other one. Let us then discuss the three representations :

- The *heard-of representation* [34] is the representation used in Figure 2.1. There, to each process we attach the set of processes whose message it has received. In this representation, broadcast is achieved iif there exist an ID that is present in every set.
- In the *matrix representation*, the entry (i, j) of the binary matrix represents whether or not process j has received a message from process i. Note that the columns of the matrix are exactly the indicator vectors of the sets in the heard-of representation. Here, broadcast is achieved iif there exist a row without any 0.
- In the *sent-to representation*, we attach to each process the set of processes to whom it has sent a message already. Note that the rows of the matrix (defined in the previous bullet point) are exactly the indicator vectors of the sets in this representation. Here, broadcast is achieved iif there exist a set with n distinct elements.

It is easy to see that the heard-of and the sent-to representations are dual of each other. Moreover, one can note that, for a single round t of communication, the matrix representation is exactly the sum of A_t, the adjacency matrix of the graph, and I_n, the identity matrix. In fact, it is easy to show that to obtain the matrix of multiple communication rounds, one can simply multiply together the matrices of each round using the boolean rules $0+0=0, 0+1=1+1=1, 0\times 0=0\times 1=0, 1\times 1=1$.

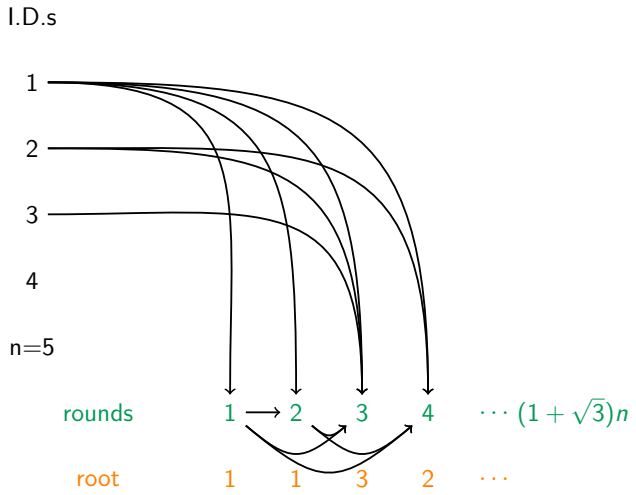


Figure 2.2: The auxiliary graph of the example in Figure 2.1. Each round i with root r_i has the following in-edges: one edge for each process the root had heard of at this round, and one edge for each round $j < i$ with root r_j such that r_i had heard of r_j. Note that round i has in-degree at least i.

2.1.3 A first solution: The *forwards-backwards* analysis

This is an easy observation:

Observation 2.1.1 (The forwards analysis)**.** *Assume that after a certain number of rounds k, and before broadcast is achieved, an process a has heard of the message m. Assume moreover that in the next round $k+1$, this process is the root of the communication graph. Then we are guaranteed that, after round $k+1$, the "sent-to" set of process m must be strictly larger than after round k.*

Indeed, as broadcast was not yet achieved, after round k, there must be at least one process b that hasn't heard of m yet. In the graph of round $k+1$, there is a path from the root a to b, and that path must contain an edge from an process c that had heard of m before the round, to one d which hadn't. During that round, c forwards the message m to d.

This yields another immediate observation:

Observation 2.1.2. *If a message m was received by the root of $n-1$ different rounds, broadcast is achieved.*

Indeed, as at each of those rounds m is received by at least one new process by the previous observation, and that the number of processes is at most n, m needs at most $n-1$ of those rounds to broadcast its message.

To better capture the process, we then introduce the *auxiliary* graph. In this graph, there are two types of nodes: the *process nodes* of which there are n, one for each process, and the round nodes, of which there is one for each round considered so far. We add an edge from every process m node to every round k node when the root of the round k has heard of the message m. By the observation above, we automatically get:

Observation 2.1.3. *If a node has out-degree $n-1$ or more in the auxiliary graph, then broadcast is achieved.*

This, however, is not enough to conclude, as there is no clear way to argue that such a node with high degree must exist. In fact, we add more edges to this graph: we add an edge from round k_1 to round $k_2 > k_1$ if the root of round k_2 has heard of the root of round k_1. Clearly, this does not disturb the above observation.

A representation of the auxiliary graph can be found in Figure 2.2

To see how this is useful, we need a backwards analysis, which is the exact dual of the forwards analysis observation, although less trivial. Here, instead of starting at round one and increasing the rounds one by one, we start at a round far in the future, then add previous rounds one by one. In the matrix representation, this is about starting at a matrix of round k and multiplying on the left the matrices of previous rounds one by one, instead of starting at the first round and multiplying on the right rounds as they arrive.

Observation 2.1.4 (The backwards analysis)**.** *Let r be the root of round k_2, and look only at rounds from $k_1+1 \leq k_2$ to k_2. Assume that in those rounds, r has not heard of the root of k_1. Then considering rounds from k_1 to k_2, r must have a "heard-of" set strictly larger than in rounds from $k_1+1 \leq k_2$ to k_2.*

This observation might be hard to understand, but one should keep in mind that it is the dual of the forward analysis. This translates well into the auxiliary graph, where it guarantees that the in-degree of round k node is at least k for every k.

The final argument is then a pigeonhole principle:

Observation 2.1.5. *After $(1+\sqrt{3})n$ rounds, there must exist a node in the auxiliary graph that has out-degree at least n.*

Indeed, since the in-degree of round i is at least i, we know that after k rounds, the number of edges is at least $\sum_{i=1}^{k} i = \frac{k(k+1)}{2} \geq \frac{k^2}{2}$. After k rounds, the number of nodes in the auxiliary graph is $n+k$, with n process nodes and k round nodes. And thus, the pigeonhole principle asserts that there must exist a node with out degree at least $d = \frac{k^2}{2}\frac{1}{n+k}$. For $k = (1+\sqrt{3})n$, we get $d = \frac{(1+\sqrt{3})^2 n^2}{2(1+\sqrt{3})n} = n$.

A more careful analysis, detailed in Chapter 3 of the auxiliary graph gives a $\left\lceil (1+\sqrt{2})n - 1 \right\rceil$ upper bound.

2.1.4 A second solution: The *shrinking covers* analysis

Here is another solution to this problem, yielding a slightly worse result but generalizable to different problems. This time, we are only going to work with the backwards analysis, and work with the notion of *covers*.

Definition 2.1.6 (Cover, simplified)**.** *A set of processes S at time t is said to be a cover of time t' if all processes in $[1,n]$ have, at time t', are reached by at least one process in S between times t and t'.*

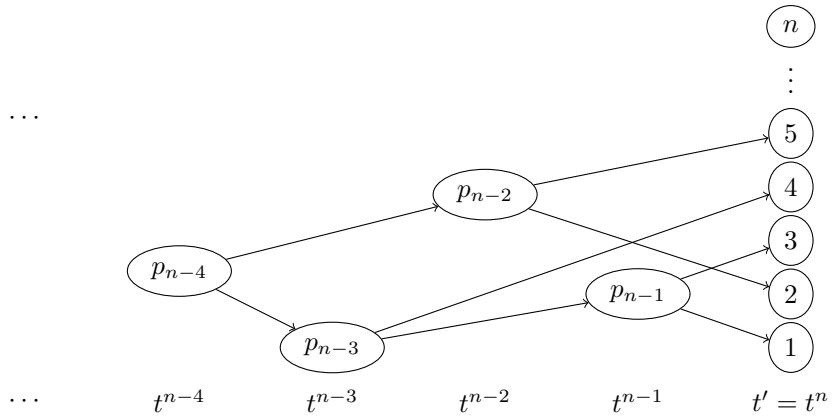


Figure 2.3: Main idea of the of the shrinking covers analysis. An edge from process x in column t_1 to process y in column t_2 means that x can reach y between times t_1 and t_2. We start with a cover of size n at t^n then go back in time, finding a smaller cover at every step. After the first step, we don't need to cover 1 and 3 anymore, because if we reach p_{n-1} before round $t^{(n-1)}$, we also reach 1 and 3 before round t', by going through p_{n-1}.

The point is then to find covers of smaller and smaller cardinality as we go back in time, until we find a cover of cardinality 1. To do so, we start with a cover of size n at time t^n. We then go to time $t^n - 1$ and try to find an process that can reach two processes between times $t^n - 1$ and t^n. We continue going back in time until t^{n-1} where such an process p_{n-1} exists. We then remove the two reached processes and replace them with the reaching process p_{n-1}. We now have a cover S at time t^{n-1} of time t^n of cardinality $n - 1$. We now do the same procedure again to find a cover of cardinality $n - 2$ at time t^{n-2}, by finding an process that reaches two elements of S. The crucial point now is we need to reach p_{n-1} at time no later than t^{n-1}, as otherwise one cannot guarantee that all processes in $[n]$ are reached by time t^n. On the other hand all other processes can be reached by time t^n.

We then repeat this procedure until we have a cover of size 1. A representation of this idea can be found in Figure 2.3.

To analyze this procedure, we work with the heard-of sets and the backwards analysis. Indeed, as we look at the heard-of sets of each process in our cover, the goal is to find an process p in at least two sets, so we can remove the two corresponding processes and add p to the cover. This means that at any particular round, the root of the tree can be in at most one heard-of set. And thus, by the backwards analysis, at least all but one heard-of sets must strictly increase their cardinality by one.

In particular, if our cover is of cardinality $|S|$, then we know that we need to include at most $\frac{n+1}{|S|-1}$ previous rounds to find a smaller cover. Indeed, by then, the sum of cardinalities of the heard-of sets it at least $n+1$ by then, which implies there must be an process in two heard-of sets.

Define Δ_u to be the number of rounds the cover we consider is of size u. Then the total number of rounds we need is $\sum_{u=2}^{n} \Delta_u$. The above analysis gives $\Delta_u \leq \frac{n+1}{u-1}$, which would only give a $O(n \log n)$ upper bound on the total number of rounds.

We thus refine our analysis: let us look again at the case where our cover is of cardinality

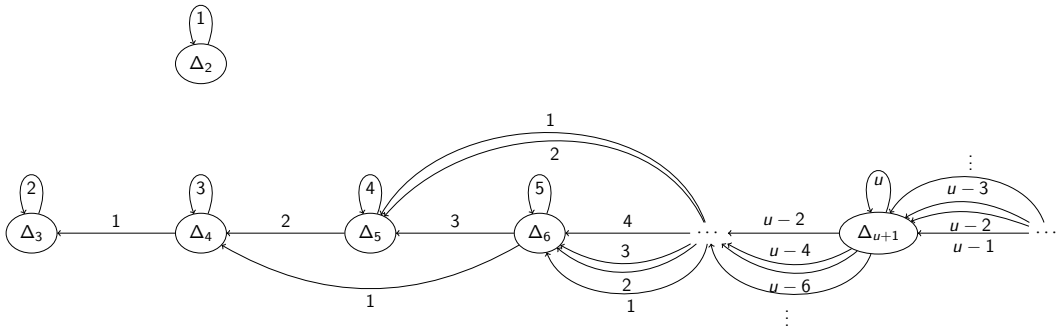


Figure 2.4: The strict rounds graph. Vertex u's weight represents $\Delta_u = t^{(u)} - t^{(u-1)}$. The weight of an edge (u, s) represents how much a round $t \in [t^{(u-1)} + 1, t^{(u)}]$ contributes at least to the total cardinality of the heard-of sets of the cover of cardinality s. We have, for every vertex v, that $\sum_{(u,v)\in E} w(u)w(u, v) \leq n + 1$.

$|S|$, and assume that we did indeed require all $\frac{n+1}{|S|-1}$ rounds to find a smaller cover. Then our cover S' of cardinality $|S| - 1$ has a head start: indeed, this cover is constructed by removing two processes from S and adding a fresh one. This means that the heard-of sets of S' are exactly the ones taken from S, apart from one fresh one. Hence, during the $\frac{n+1}{|S|-1}$ rounds needed to increase the total cardinality of the heard-of sets of S by $|S| - 1$ in each round, it also increased the cardinality of the heard-of sets of S' by $|S| - 3$.

We thus get a new interdependency between the Δ_u for different values of u. In fact, this dependency is generalizable, and can be represented in the *strict rounds graph* depicted in Figure 2.4.

In that graph, the node u's weight $w(u)$ represents the numbers of rounds the cover stays of cardinality u, and the edge (u, v)'s weight $w(u, v)$ represents by how much a round, when the cover is of cardinality u, increases the total cardinality of the heard-of sets. Thus, for each node v, we have that $\sum_{(u,v)\in E} w(u)w(u, v) \leq n + 1$. We then show that in this graph under such a constraint, one must have that $\sum_{u=2}^{n} w(u) = \sum_{u=2}^{n} \Delta_u \leq \frac{\pi^2+1}{6} \cdot n$.

2.2 Broadcast in Dynamic Trees with a Stochastic adversary

2.2.1 Problem Definition and Results

We are given n processes (also called nodes), each with a different ID in $\{1, \ldots n\}$. Process 1 has a message it wants to broadcast. Communication happens in rounds, where in each round the communication is represented by a directed graph, where each process can send the message along its outgoing edges before receiving the messages sent along its ingoing edges. Processes can make a copy of and store the message once it is received, and forward the message if it has received it in a previous round.

The communication graph in each round is chosen as follows: an adversary, who knows which processes have received the message until then, chooses k edges to be inserted in the

communication graph, insuring that no cycles is created, and that no node has two incoming edges. Then the communication graph is chosen uniformly at random among all rooted[1] trees that contain these edges.

The question is then: for how many rounds can the adversary delay broadcast? An immediate result is $\Omega(k)$: the adversary can choose the chain $1 \to 2 \to \cdots \to k$ for every round. The message from the root of the chain needs to travel down to the leaf, and that takes $k-1$ rounds.

We showed a $O(k + \log n)$ upper bound, which holds with high probability. We also showed that in the case where $k = 0$, broadcast needs $\Omega(\log n)$ rounds with high probability, which make our bounds tight.

2.2.2 Combinatorics of Rooted Trees

How does one choose a tree among all rooted trees, uniformly at random?

Cayley [31] showed that there is a bijection correspondence between sequences of $n-2$ integers between 1 and n, and undirected trees on n vertices. He also gave an algorithm that computes this bijection. This gives a straightforward algorithm to compute an undirected tree uniformly at random: choose a random sequence of $n-2$ integers as above, and use Cayley's algorithm to transform it into a tree.

Since there is also a trivial bijection between all rooted trees and pairs (T, r) of undirected trees T and a choice of a root r, this gives an algorithm to compute a rooted tree uniformly at random among all rooted trees on n vertices. However, Pitman [116] showed that there is another way: start with an empty graph, and choose a directed edge uniformly at random. We now have a rooted forest (a forest where each weakly connected component is rooted). For the second edge, choose any edge uniformly at random among all edges that do not "clash" with the previous edge, that is, maintain the fact that our graph is a rooted forest. Doing so iteratively for $n-1$ total steps yields a directed rooted tree.

Pitman's algorithm in fact generalizes to the case where we need a rooted tree uniformly at random among all trees that contain a rooted forest F: given such a forest, simply run Pitman's algorithm on top, for a total of $n-1-|E|$ iterations, where E is the set of edges of F. He further shows that the total number of such trees is $n^{n-1-|E|}$, a very clean formula.

In contrast, there is no straightforward algorithm for the undirected case. Even the corresponding formula for the number of trees that contain an undirected forest is more complex:

Theorem 2.2.1 (Lemma 6 of [104]). *Let us be given an undirected forest F on n vertices, with connected components $C_1, C_2, \ldots, C_m$ of $f_1, \ldots, f_m$ vertices with integer $m \geq 1$. Let $|E|$ be the number of edges in F. Then, the number of undirected trees T on n vertices, such that $F \subseteq T$, is $\left(\prod_{i\in[m]} f_i\right) n^{n-2-|E|}$*

However, this theorem still proves to be useful for us. Indeed, it turns out we require one more tool to be able to compute all the probabilities we need: What is the probability that, given a rooted forest F, and a node v, that v is the root of a tree chosen uniformly at random among all rooted trees that contain the forest?

[1] Not necessarily at 1

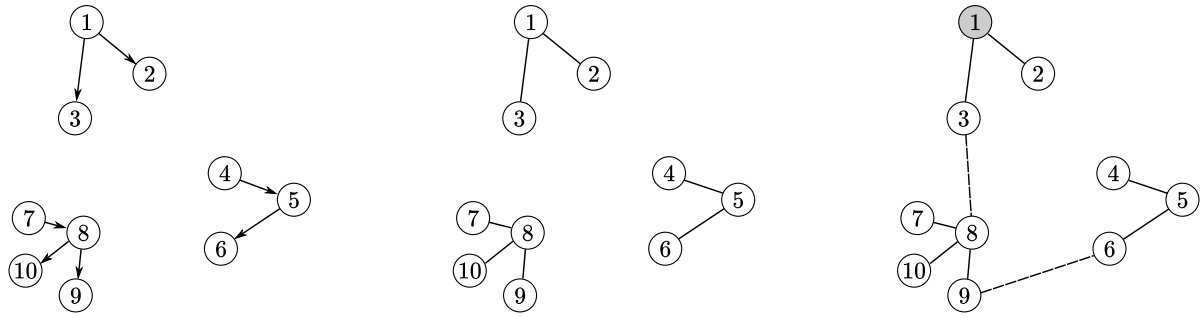


Figure 2.5: Left: A directed rooted forest F. Middle: its undirected version. Right: A directed rooted tree T, as seen as an undirected tree with a choice of a root. T contain the *undirected* F, but not the *directed* F.

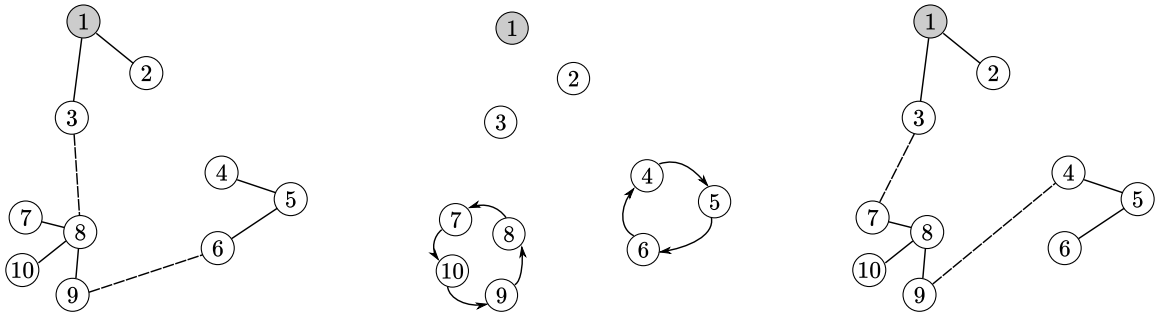


Figure 2.6: Left: A tree T that does not contain the directed forest F from Figure 2.5. Middle: An element σ of the group that acts on the set of trees. Right: The result of the action of σ on T. Note that the directed forest F is now included in the new tree.

To compute that probability, we look at the following procedure: ignore the roots of the connected components of F (that is, look at F as an undirected forest), then choose a tree uniformly at random among all undirected trees that contain that forest. Choose a root uniformly at random among all the nodes. The tree obtained then does not necessarily contain F in its directed form. To ensure that, in each connected component C of F, we change the incoming edge (u, w) with $u \notin C$, $w \in C$, to (u, w') where $w' \in C$ is the root of C in F. If the root of T is in the same connected component of F as v, then we set v to be the root. This procedure is illustrated in Figure 2.5 and Figure 2.6.

Trivially, we then have that the probability we are looking for is $C(v)/n$, where $C(v)$ is the connected component of v in F. It remains unclear, however, that this procedure satisfies the requirements, that is the tree obtained is chosen uniformly at random among all rooted trees that contain F.

To show that this is indeed what we need, we borrow a trick from group algebra: Note first that the first part of the procedure: completing the undirected forest then choosing a random root, is a uniform distribution over all rooted trees that contain the undirected forest F.

To show that the second part of the procedure: moving the roots of different connected components around, yields then a uniform distribution over all rooted trees that contain the *directed* forest F, it suffices to show that the mapping of that second part is a fixed ℓ ratio, for some $\ell \in \mathbb{N}$, that is, for any given rooted tree that contains the *directed* forest F, there are ℓ rooted trees that contain the *undirected* forest F that map to it.

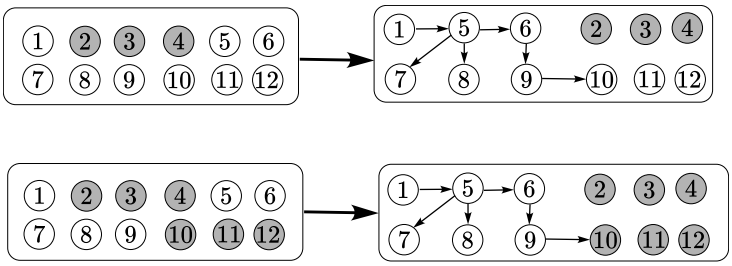


Figure 2.7: The best strategy for the adversary A, with $k = 6$. Shaded nodes are informed nodes. In the top example, nodes $5, 6, 7, 8, 9$ and 10 are protected from being informed, whereas node 1 can still be informed. In the bottom example, nodes $5, 6, 7, 8,$ and 9 are protected, whereas node 1 can still be informed. However, node 1 is protected from being informed by node 10.

To do so, we show that the second part of the procedure can be seen as the action of an element of a group on the set S of rooted trees containing the *unidirected* F. In such a case, the group action partitions S into *orbits* of equal cardinality. We further show that in each orbit, there is exactly one tree which also contains the *directed* F, and that all elements of the orbit map to that tree under the second part of the procedure, which completes the proof of the following theorem:

Theorem 2.2.2. *Let F be a directed rooted forest on n vertices, with connected components $C_1, \ldots, C_m$ of $f_1, \ldots, f_m$ vertices with integer $m \geq 1$. Let $|E|$ be the number of edges in F. Let $v \in C_i$ for some $i \in [m]$. Then the number of rooted trees T on n vertices, such that $F \subseteq T$ and v is the root of T is $f_i n^{n-2-|E|}$*

2.2.3 Finding the best strategy for the adversary

To solve the problem of how long the adversary can delay broadcast, one must think like the adversary, and understand what its strategy might be to achieve that goal. However, as the problem is probabilistic in nature, the problem itself is ill-defined: what does it mean to delay broadcast as much as possible? Is it in expectation? With high probability? Or some other metric?

It turns out that one can find a strategy that works for all possible (and reasonable) definitions of "delay broadcast as much as possible". Indeed, we show that there exists a strategy that *stochastically dominates* all other strategies. Formally, we say that a random variable $X \in \mathcal{R}$ stochastically dominates another random variable $Y \in \mathcal{R}$ if and only if $\mathbb{P}(X \geq x) \geq \mathbb{P}(Y \geq x)$ for all $x \in \mathcal{R}$.

Here is a classic example of stochastic dominance: let X be a gain associated to a die roll: if a 1, 2 or 3 is rolled, the gain is 3, while the gain is 6 for any other roll. Let Y be the straightforward gain of 1 if the die rolls a 1, 2 if a 2 is rolled, and so on. X stochastically dominates Y as one always earns more with X rather than Y. Let Z be the gain symmetric to X: if a 1, 2 or 3 is rolled, the gain is 6, otherwise the gain is 1. If the die is fair, Z has the same probability distribution as X, and we also say that Z dominates Y.

Let us now think what it means for the adversary to insert an edge (u, v) between two nodes u and v. If they were both *uninformed*, that is, none of them have received the message

yet, then v would not be able to receive the message in the current round either. Indeed, v can have at most one in-neighbour, and thus, v would be protected from broadcast for at least one round. If, on the other hand, u was informed, then v would be guaranteed to be informed in the next round, which is counter-productive for the adversary. If both u and v were informed, then having an edge between them or not would make no difference on the broadcast process. If u was uninformed while v was informed, then u would be guaranteed to not have v as an in-neighbour (as there are no cycles in a tree), and thus would guaranteed to not be informed in the next round *via node* v. However, u could still receive the message from another informed node, and is thus only *partially* protected.

Therefore, there are two interesting strategies to have with an edge for the adversary: completely protect an uninformed node v by adding an incoming edge from another uninformed node u, or partially protect an uninformed node u by adding as many informed nodes as possible among its descendants. For a given tree on $k_1 \leq k$ nodes, the best strategy for the adversary is thus to populate the nodes near the root with uninformed nodes, and the ones near the leave with informed ones, ensuring that no edge goes from an informed node to an uninformed node.

Three questions remain: first, does the shape of the tree on $k_1 \leq k$ nodes matter? Is it better to have a star or a line graph? It turns out, due to the symmetry of the rooted trees, the shape does not matter.

Second, is it smarter to fill the trees with as many uninformed nodes as possible, or is there a sweet spot of a number of informed nodes to add to the tree? As inserting uninformed nodes fully protects them while inserting an informed node only partially protects the root, intuitively one should aim for as many uninformed nodes as possible. This intuition can be proved formally.

Finally, is it better to divide the k edges to form different trees, or should one aim for a single tree? Interestingly, if all nodes involved in all trees are uninformed, it ultimately does not matter. However, creating only one connected component requires only $k+1$ many uninformed nodes to use up all edges, while creating ℓ many connected components requires only $k+\ell$ many uninformed nodes. Hence, using only one connected component allows to fully protect k nodes as long as there are at least $k+1$ uninformed nodes.

This concludes the best strategy for the adversary: Create one connected component with k edges, populate it with as many uninformed nodes as possible, complete the rest with informed nodes, ensure that no edge goes from an informed node to an uninformed one. An illustration of this strategy can be found in Figure 2.7.

2.2.4 Analyzing the information dissemination

We are now ready to analyze the information dissemination. We divide the analysis into two phases. The first, where the adversary is able to fully populate a tree with $k+1$ many uninformed nodes, and the second, where all nodes outside of the tree chosen by the adversary are informed.

In the first phase, there are k nodes that are protected from being informed in that round. If N is the total number of nodes that were informed before that round, there are thus $n-N-k$ many nodes that are susceptible to be informed during that round.

There, there are two distinct types of nodes to consider: the nodes that are isolated in the graph given by the adversary, and the root of the tree with $k+1$ nodes.

Consider first the root of the tree given by the adversary. By Theorem 2.2.2, the probability that this node is also the root of the randomly selected tree is $\frac{k+1}{n}$. If it is not the root, then it must have a parent among the $n-(k+1)$ isolated nodes. since these nodes are symmetric for the purposes of randomly choosing a spanning tree, the probability of any of them being chosen is $\frac{1-(k+1)/n}{n-(k+1)} = 1/n$. Hence, the probability that its parent is an informed node is N/n.

Consider now an uninformed node that is isolated in the graph given by the adversary. We know, by Theorem 2.2.2, that the probability that this node is the root of the graph is $\frac{1}{n}$. We further know that this node has $n-1$ possible incoming edges (one from each other node, as this edge creates no cycles and no node with two incoming edges). Select one of them arbitrarily. Pitman's result shows that there are $n^{n-1-(k+2)}$ many rooted trees that contain the graph given by the adversary and that extra edge. This proves that all the $n-1$ incoming edges must have the same probability of appearing, and thus each of them appears with probability $\frac{1-1/n}{n-1} = 1/n$. Hence, the probability that its parent is an informed node is N/n.

Hence, if we denote by X_i the 0-1 random variable for a uninformed node i of its parent being informed or not, the number of newly informed nodes is $X = \sum_i X_i$, where each X_i is a Bernoulli random variable of parameter N/n. We further exploit the symmetries of the rooted trees combinatorics to show that these random variables are in fact independent. We thus show that X follows a binomial distribution of parameters $(n-N-k, \frac{N}{n})$.

In the second phase, remember that all of the nodes outside of the tree given by the adversary are informed, and the root of the tree is uninformed. Here, all uninformed nodes in the tree, apart from the root, are protected from being informed. This means that the only question is whether the root itself gets informed.

Again, by Theorem 2.2.2, the probability that the root of the tree given by the adversary is also the root of the randomly chosen spanning tree is $\frac{k+1}{n}$. If it is not the case, then it must have an incoming edge from one of the isolated vertices, which are all informed, and which happens thus with probability $1-\frac{k+1}{n}$. Thus, the number of informed nodes increases by one with probability $\frac{n-k-1}{n}$, and stays the same otherwise.

We then show that as long as $k < \frac{2}{3}n$, phase 1 completes in $O(\ln n)$ time, while phase two completes in $O(\ln n + k)$ time, with high probability.

2.3 A Lower Bound for Solving the Relative Majority Problem with Undecided States Dynamics

For this paper, the initial goal was to find a better algorithm for solving the problem of relative majority in population protocols. One of the state of the art algorithms was the Undecided States Dynamics, which solves the problem in roughly $O(k \log n)$ parallel time. Our hope was to get rid of the k factor, as $\Omega(\log n)$ was an obvious lower bound inherited from the broadcast problem.

This quest led us to better understand the Undecided States Dynamics algorithm itself. And after trying long to show that the upper bound given by previous authors is not tight, we

realised we were wrong, and came up with a matching lower bound instead.

This overview is a light rewrite of the overview originally published in [57].

2.3.1 Problem Definition and Results

We are given n agents, also called nodes. Each agent i is initially given an opinion in $s_i \in \{1, \ldots, k\}$. The goal of the agents is to decide which opinion was the (relative) majority at the start of the computation. The computation then happens in rounds, where in each round two agents are selected, uniformly at random (without replacement), independently of the other time steps. The two agents are able to see each other's state, and then update their own state according to a transition function.

The transition function we consider is given by the Undecided State Dynamics: there are $k+1$ possible states. The opinion states $\{1, \ldots, k\}$, and the undecided state $\perp$. When two agents with different opinions meet, they both become undecided, but when an opinionated agent meets an undecided one, the undecided agent takes on the opinion of the opinionated one.

We ask how long it takes for the system to stabilize, in the particular case where the majority opinion starts with an initial additional bias of $\Omega(\sqrt{n \log n})$. Such a bias is needed to guarantee that w.h.p.[2] the opinion with the largest (relative) initial support wins (cf. [9, 15]).

Our main contribution is an almost tight lower bound on the stabilization time of Undecided State Dynamics for plurality consensus in population protocols. Specifically, we show that the time needed to stabilize is $\Omega\left(kn \log \frac{\sqrt{n}}{k \log n}\right)$ interactions, or in $\Omega\left(k \log \frac{\sqrt{n}}{k \log n}\right)$ parallel time[3], in the case where $k = o(\frac{\sqrt{n}}{\log n})$. In our proofs we assume that our initial configuration has bias at most $O(\sqrt{n}/(k \log n)^{1/4}\sqrt{n \log n})$. Interestingly, this includes even an initial bias of $\omega(\sqrt{n \log n})$. This bound is tight for any $k \leq n^{\frac{1}{2}-\epsilon}$, where $\epsilon > 0$ can be any small constant, as Amir, Aspnes, Berenbrink, Biermeier, Hahn, Kaaser, Lazarsfeld [9] gave a $O(k \log n)$ parallel time upper bound. For larger values of k, as any initial configuration that is valid for k_0 is also valid for $k \geq k_0$, we can simply plug in $k_0 = \frac{\sqrt{n}}{\log n \log\log n}$ to get a $\Omega\left(\frac{\sqrt{n} \log\log\log n}{\log n \log\log n}\right)$ parallel time lower bound.

Our analysis is based on a precise characterization of the number of undecided nodes over time, using drift analysis and random walks.

2.3.2 Notation

$\mathbf{x}$ represents a configuration the system can be in: $\mathbf{x} = (x_1, \ldots, x_k, u)$, where x_i is the number of agents with opinion i for all i, and u is the number of undecided agents. We denote by $\mathbf{x}(t)$ the configuration of the system after the t-th interaction, and by $\mathbf{x}(0)$ the initial configuration. $x_i(t)$ is the number of agents with opinion i after interaction t. Similarly, $u(t)$ is the number of undecided nodes after t interactions. Accordingly, $x_i(0)$ is the initial number of nodes with opinion i (clearly, $u(0) = 0$). We assume that the opinions are initially ordered from most common opinion to least common, i.e, $x_1(0) \geq x_2(0) \geq \cdots \geq x_k(0)$.

[2] with high probability

[3] The parallel time is equal to the number of interactions divided by n.

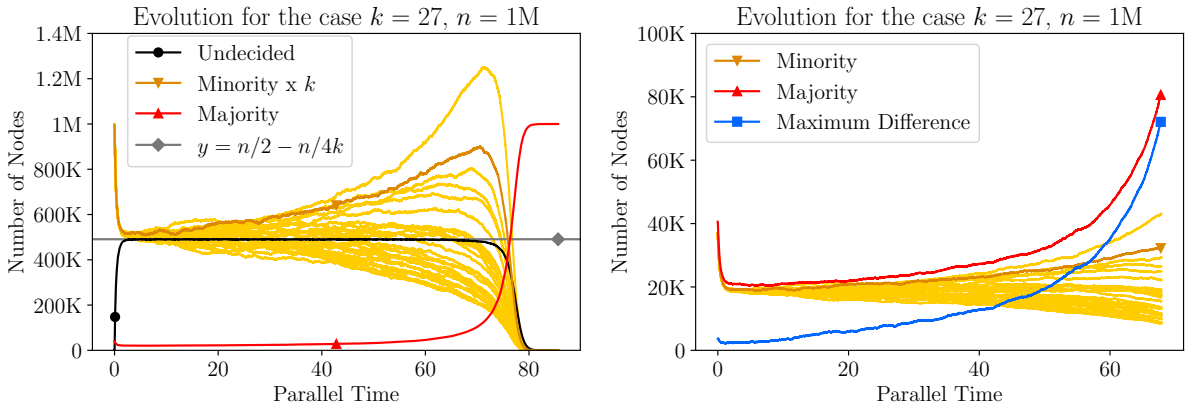


Figure 2.8: Intuition for stabilization of plurality consensus in undecided state dynamics. On the left figure, we scale up the minority opinions by a k factor for visibility. In this experiment, $n = 1,000,000$ agents and $k = \frac{\sqrt{n}}{\log n \log\log n}$. In the initial configuration $k-1$ opinions had the same support, while opinion 1 had an additive advantage of $\sqrt{n \log n}$. Out of all the minority opinions, we plot one with a darker color for visibility.

2.3.3 Technical Overview

Before we delve into the technical details of our lower bound proof, let us provide some intuition about how agent opinions propagate across the distributed system, and how majority and minority opinions evolve accordingly. Based on this intuition and the observed challenges, we will then give an overview of our analytical approach and its key ideas.

To illustrate the evolution of opinions and the stabilization behavior, Figure 2.8 (left) plots a typical simulation run over parallel time. In this example, all minority opinions are initially equally frequent, while the majority one has an initial additional bias of $\sqrt{n \log n}$. The majority opinion is plotted in red. The minority opinions are plotted in yellow, except for a random one which we select as an example and which we highlight in orange; for better visibility we multiplied the number of each minority opinion by k.

We make several observations which highlight parts of the complexity of the problem. First, note that different minority opinions evolve differently. In particular, not all minority opinions are strictly decreasing over time, but many are actually increasing over a long time period. In the example in Figure 2.8, one opinion even surpasses its initial count. On the other hand, the majority opinion in this example (which is typical for many runs) remains low for a long time during stabilization, but then increases quickly toward the end. We can also see that the number of undecided opinions stays close to $\frac{n}{2} - \frac{n}{4k}$ throughout the execution.

In Figure 2.8 (right), we zoom in on the time period for which it takes x_1 to double from its initial number of nodes. We can see that it takes most of the stabilization time to reach this configuration, as the system needs around 70 parallel time to do so, after which only 20 more rounds are required to fully stabilize.

These simulations also provide some intuitive motivation for our choices for analyzing the problem theoretically. We will prove that the number of undecided nodes does not substantially exceed $\frac{n}{2} - \frac{n}{4k}$ with high probability, and we only consider the interactions before x_1 reaches $2x_1(0)$, which we will approximate as $2\frac{n}{k}$. One particular ingredient we consider in our analysis

is the *maximum* difference between the majority opinion and minority ones (also plotted in Figure 2.8 (right)): $\max_{j\geq 2}\{x_1 - x_j\}$. The idea is that as long as this difference stays small, the system is very slow to change.

More concretely, for our analysis, we proceed as follows.

We first observe that a precise characterization of the number of undecided nodes $u(t)$ is key to understand the state dynamics as, intuitively, for any opinion i, the larger the number of undecided nodes, the more new nodes can the i-opinionated nodes "convert" to the opinion i, hoping to compensate for the i-opinionated nodes that meet nodes with other opinions and thus become themselves undecided. In fact, in their analysis of the stabilization time, Amir, Aspnes, Berenbrink, Biermeier, Hahn, Kaaser and Lazarsfeld [9] derived an upper bound and a lower bound on the number of undecided nodes throughout the interactions: $\frac{n}{2} - \frac{x_1}{2} \leq u(t) \leq \frac{n}{2}$ for any t after the first $n \log n$ interactions. More precisely, the evolution of the number of nodes in opinion i is driven by the number of undecided nodes. For each opinion, there exists a value u_i of the number of undecided nodes that acts as a threshold. That is, if the number of undecided nodes u is above u_i, then the corresponding opinion x_i increases, whereas if u is below the threshold, then x_i decreases. The threshold is a decreasing function in the number of nodes in opinion i: the larger x_i is, the smaller u_i is. One can see this in action in Figure 2.8 on the left. When $u(t)$ is still very small in the beginning, all of the opinions decrease quickly. Then, once $u(t)$ settles around $\frac{n}{2} - \frac{n}{4k}$, some opinions start to steadily increase (even minority ones) while others decrease. One can understand this the following way: in the initial steps, where the number of undecided nodes increase quickly, some opinions, due to randomness, end up having an advantage on the others. Towards the end, the number of undecided nodes starts dropping, it thus goes below all thresholds but the one of the majority opinion, which means all opinions drop quickly apart from the majority one.

In our case, for the study of the lower bound, we fully control the initial configuration of the opinions, and we are able to give a better upper bound on $u(t)$ over time, which happens to be very close to the threshold of any opinion i. To do so, we use drift analysis (for a nice introduction, we refer to Lengler [100]). More specifically, we use a result by Oliveto and Witt [115]. Here is a quick intuition on drift analysis: assuming we know the current configuration of the system, we compute the expectation of the change in the number of agents in each state after the next interaction. The idea is, if a number changes in expectation by a value α at each interaction, and our goal is to show that the actual number does not wander off by more than β from its original value, this takes at least $\Omega(\beta/\alpha)$ many interactions. A few more hypotheses are needed to ensure that this result holds with high probability, and to ensure that the probabilities are well-behaved, but this is the main idea.

It turns out that $u(t)$ settles around $\frac{n}{2} - \frac{n}{4k}$, and to show that $u(t)$ never substantially exceeds this value, we prove that if it slightly exceeds this value, then at each interaction, in expectation, $u(t)$ decreases by at least $\sqrt{\log n/n}$. Drift analysis tools then allow us to ensure that $u(t)$ will drift no more than (roughly) $\theta(\sqrt{n \log n})$ away from that position.

The second step we make after giving an upper bound on $u(t)$, is to show that in $\theta(kn)$ interactions, no opinion can go from $\frac{3n}{2k}$ nodes to $\frac{2n}{k}$ agents. Here, the naive approach of drift analysis fails to help, and that for one main reason: it essentially looks at how much the expectation increases or decreases over time, but sometimes, the expectation gets beaten by the variance of the process, and the naive drift analysis approach fails to capture the lack

of concentration of the process. To understand that, consider a random walk starting at 0 and which at each step either increases or decreases by 1, each with probability $1/2$. The expectation of its increase is 0 so in expectation, after m steps, it is still at 0. However, we know that with high probability it will have reached $\theta(\sqrt{m})$ at some point during those m steps, as one can see the position after m steps as the result of a binomial distribution with parameters $(1/2, m)$, whose standard deviation is of the order of $\sqrt{m}$.

In our case, we avoid this problem by looking more carefully at the probabilities involved. By considering the evolution of x_i, while its expectation increases slowly over time, it is not overtaken by the variance. To see that, think again of our example of our ± 1 random walk: Imagine that now, the random walk increases with probability $p/2$, decreases with probability $p/2$, and stays put with probability $1-p$. Look now at what happens after m steps. The walk actually moved for pm out of those steps, and thus the standard deviation is now around $\sqrt{pm}$. If $p = o(1)$, this has a significant impact.

This is exactly what happens in our case, where for x_i to change, in an interaction, a node with opinion i must have been chosen. If at most $2\frac{n}{k}$ nodes with this opinion exist, then the probability of this event happening is of the order of $1/k = o(1)$.

While we could have chosen to stop here for our analysis, and thus give a $\Omega(k)$ lower bound, we go one step further: there is a weakness in this analysis, i.e., we overestimate the number of nodes in opinion i, by stating that the initial count is at most $\frac{3n}{2k}$, while in practice, most opinions stay close to $\frac{n}{2k}$ for most of the process. This (high) estimate of $2\frac{n}{k}$ then gives a too low threshold on u for x_i to increase, and thus the upper bound on the rate at which x_i increases is not tight enough. To improve the result, we would need to give a better estimate on the initial x_i, and show that it is close to $\frac{n}{2k}$. This is not straightforward, as the initial count is close to $\frac{n}{k}$, and we would thus need to analyze the initial phase where u increases sharply while all the x_i decrease.

We find a workaround for this difficulty: while it might take effort to accurately approximate x_i after the first quick-changing phase, a value that is easier to estimate is $\Delta_{ij} = x_i - x_j$. Here, instead of requiring a good estimate on x_i and x_j to analyze the evolution of Δ_{ij}, it turns out that we only need their order of magnitude, as well as a good estimate of Δ_{ij}, which we have (it starts at or close to 0 and only decreases in the initial, quick-changing phase). Therefore, knowing that $x_i \leq 2\frac{n}{k}$ for $\theta(kn)$ iterations allows us to show that the maximum Δ_{ij} needs at least $\theta(kn)$ iterations to double. However, if all Δ_{ij} are small enough (if they are all $o(\frac{n}{k})$), then it means that no opinion drifts far from the others, and thus $x_i \leq \frac{3n}{2k}$ after those $\theta(kn)$ interactions.

Hence, our high level argument works as follows: First give a good estimate on u, then on the number of nodes in each opinion, and on the maximum difference between opinions: As long as $x_i \leq \frac{3n}{2k}$ at a time t and $\Delta_{ij} = o(\frac{n}{k})$ for all i, j, then the order of x_i does not change in the next $\theta(kn)$ interactions, which in turn is used to show that Δ_{ij} does not double during those same interactions. This in turn shows that $x_i \leq \frac{3n}{2k}$, which allows us to restart another iteration of the induction. The induction holds for $\log\left(\frac{\sqrt{n}}{k \log n}\right)$ iterations, which then gives the lower bound.

2.4 Fully-Dynamic Minimum Cut in Subpolynomial Time

This overview contains sections of the overviews originally published in [59] and [58].

2.4.1 Problem Definition and Results

We are given an unweighted graph $G = (V, E)$ on n nodes and m edges, where parallel edges are allowed (and an edge's multiplicity counts towards m). The graph is *fully-dynamic*, that is, is subject to edge insertions or deletions. The problem is to maintain a *minimum cut* of the graph after each update, that is, a partition of V into two sets such that the number of edges crossing from one set to the other is minimized.

For values λ of the minimum cut smaller than $2^{\Theta(\log^{3/4} n)}$,we develop an algorithm that maintains such a partition in $n^{o(1)}$ amortized update time and can return the number of cut-edges in $O(1)$ time. To explicitly return the partition, we require $|V_1|$ time, where V_1 is the set of the partition of smallest cardinality. Our algorithm is exact and deterministic.

For larger values of λ, and/or if the graph is weighted, our algorithm is a Monte-Carlo randomized algorithm which succeeds with high probability. This algorithm works against an oblivious adversary.

To get to this result, we first get a partial result: a deterministic algorithm that, given a fixed lower bound $\lambda_{\min}$ and an upper bound $\lambda_{\max}$ on λ satisfying $\lambda_{\min} \leq \lambda_{\max} \leq \mathbb{1}.2\lambda_{\min} = n^{o(1)}$, implicitly maintains a minimum cut in $n^{o(1)}$ amortized update time. The rest of this overview focuses on this partial result.

2.4.2 Technical Overview

We are thus given a fully-dynamic graph, and two (fixed) parameters $\lambda_{\min}$ and $\lambda_{\max}$ satisfying $\lambda_{\min} \leq \lambda \leq \lambda_{\max} \leq 1.2\lambda_{\min} = n^{o(1)}$, and the goal is to maintain a minimum cut on this graph.

Our algorithm relies on three key techniques: the dynamic expander decomposition, as presented by Goranci, Räcke, Saranurak and Tan [77], the concept of $(1-\delta)$-boundary sparseness, as presented by Henzinger, Li, Rao, and Wang [82], and a "local" version of Karger's minimum cut algorithm, which we call *LocalKCut*, introduced by Nalam and Saranurak [112], which we derandomize.

Expander decomposition An expander is a graph that is "well connected", as illustrated in Figure 2.9. To formalize this idea, we use the notion of the *conductance* of a cut, which relies on the notion of *volume*:

Definition 2.4.1 (Volume)**.** *In a graph $G = (V, E)$, the volume of a set $U \subseteq V$, is defined by $\mathrm{Vol}_G(U) := \sum_{u \in U} d(u)$, where $d(u)$ is the degree of node u.*

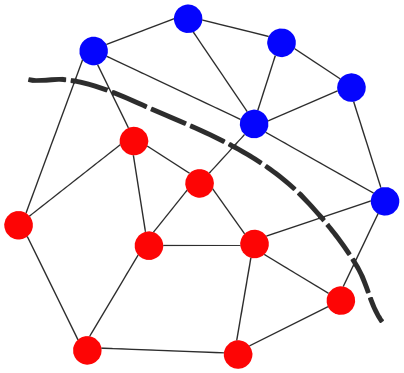


Figure 2.9: A 5/22-expander. The cut represented is a cut of conductance $\frac{5}{22}$. Since this is the cut of lowest conductance, this graph is an expander of conductance $\frac{5}{22}$.

Definition 2.4.2 (Conductance)**.** *The conductance of a cut $U \subsetneq V$ in the graph $G = (V, E)$ is*

$$\phi_G(U) = \frac{w(U, V \setminus U)}{\min\{\mathrm{Vol}_G(U), \mathrm{Vol}_G(V \setminus U)\}}.$$

A cut with high conductance has many edges crossing it. An expander (with parameter ϕ) is a graph where all cuts have conductance at least ϕ.

Definition 2.4.3 (Expander)**.** *A graph G is a ϕ-expander if every cut in G has conductance at least ϕ.*

A *ϕ-expander decomposition* of a graph $G = (V, E)$ is a partition of V into subsets $V_1, \ldots, V_k$ for some $k \in \mathbb{N}$ such that the induced subgraph $G[V_i]$ is a ϕ-expander for every $i \in [1, k]$. To find such a decomposition, one can simply look for a cut of conductance strictly lower than ϕ. If no such cut is found, then the graph is an expander, and there is nothing to do. If a low-conductance cut is found, one can partition the graph along it, then recurse on the induced graph on each subset newly created.

What is interesting in this case, is to understand how many edges are *interexpander* edges after all recursive calls, that is, how many edges cross from one expander to the other. To evaluate this quantity, we make two observations: first, note that at any time we decide to split a set into two, we do so along a sparse cut. That is, the number of edges we cut is a ϕ factor smaller than the volume of either side of the cut. If cutting an edge "costs" 1, we can thus "charge" the cost of all cut-edges to the edges of the subset of smaller volume, each of those edges paying just ϕ. The second observation is that every edge can be on the side of smaller volume at most $O(\log m)$ times, as the volume is m initially, and the volume of the set the edge is in gets halved at least every time it is on the side of smaller volume. Hence, each edge is charged at most $O(\phi \log m)$ time, and the total cost of the procedure is $\tilde{O}(m\phi)$. There are thus at most $\tilde{O}(m\phi)$ cut edges in such an expander decomposition.

Expander decomposition is the topic of many research projects in the recent years, the one that we use is the one by Goranci, Räcke, Saranurak, and Tan [77]. They are able to achieve a

fully-dynamic expander decomposition for $\phi = 2^{-\Theta(\log^{3/4} n)}$, with recourse[4] $\rho = 2^{\Theta(\log^{1/2} n)}$, and $n^{o(1)}$ amortized update time, while keeping the guarantee that the number of interexpander edges is at most $\tilde{O}(m\phi)$.

We apply this algorithm as a first step of our algorithm: indeed, as it partitions the graph into well-connected subgraphs, this looks like a promising first step towards finding the minimum cut. Ideally, we would like to contract every expander we find that way, then call the algorithm recursively until only two nodes remain. However, this does not really work: the idea of "well-connectedness" of expanders, which relies on cuts of high conductance, is different from the "well-connectedness" of minimum cuts, which rely on edge-connectivity. In practice, this means that a minimum cut S can *cross* an expander C, that is, we might have $S \cap C \neq \varnothing$ and $\bar{S} \cap C \neq \varnothing$. Contracting all expanders might thus strictly increase the minimum cut without any good enough guarantees on it even being approximable.

There is, however, a property of expanders that we can leverage: if a minimum cut S were to cross an expander C, then we know that either $S \cap C$, or $\bar{S} \cap C$, has small volume in $G[C]$, or more specifically, has volume at most $\frac{\lambda}{\phi}$. Indeed, looking at $S \cap C$ in $G[C]$, which is an expander, we know that the number of edges from $S \cap C$ to $\bar{S} \cap C$ is at most λ. Therefore, by definition of an expander, we have that $\min\{\mathrm{Vol}_{G[C]}(S \cap C), \mathrm{Vol}_{G[C]}(\bar{S} \cap C)\} \leq \frac{\lambda}{\phi}$, which immediately gives us our claim.

We can thus refine our algorithm: after running the expander decomposition algorithm, we can decompose our expanders further along cuts where a minimum cut could cross the expander: in an expander C, cut along all cuts of volume at most $\frac{\lambda_{\max}}{\phi}$ and of cut-size at most λ. This guarantees that, in the contracted graph the minimum cut is preserved.

This raises two questions, first, how do we find such cuts? And second, does that not decompose the graph too much, to the point where each node might end up in its own component, sending us in an infinite loop? The first question is addressed by the LocalKCut algorithm, and the second, by considering $(1-\delta)$-boundary sparseness. We will treat those two concepts in reverse order.

$(1-\delta)$-Boundary Sparseness This concept is the answer to the question: does cutting along all cuts of small size and small volume not decompose the graph too much, to the point where each node might end up in its own component, sending us in an infinite loop? And the answer is yes, it does. Hence, among all cuts found in the LocalKCut step, one should only select a handful to make progress, while discarding the rest. The selection criteria is exactly the $(1-\delta)$-boundary sparseness, illustrated in Figure 2.10.

Definition 2.4.4 (Boundary-sparse)**.** *(Def. 2.3 of [82]) For a set $C \subsetneq V$ and parameter $\delta \leq 1$, a cut $U \subseteq C$ is $(1-\delta)$-boundary sparse in C iff*

$$w(U, C \setminus U) < (1-\delta)\min\{w(U, V \setminus C), w(C \setminus U, V \setminus C)\}$$

[4]The recourse of an algorithm is the number of changes to the solution. In the case of the expander decomposition, the algorithm maintains which edges are intercluster edges and which ones are intracluster edges. The recourse is thus the number of edges per update that switch from being intercluster ones to intracluster ones and vice-versa.

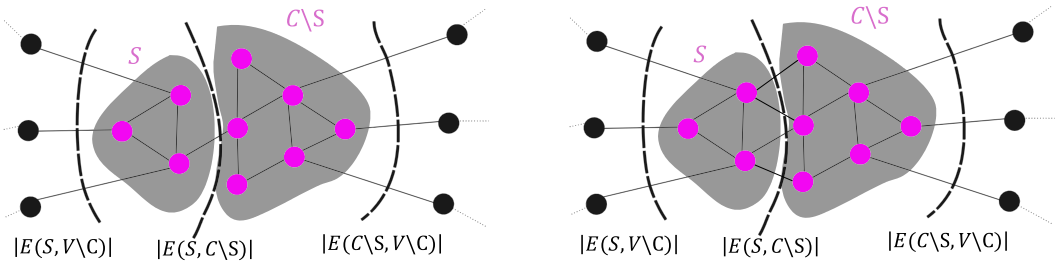


Figure 2.10: Left: A cut that is $(1-\epsilon)$-boundary sparse. Right: A cut that is *not* $(1-\epsilon)$-boundary sparse. On the right figure, a minimum cut will not separate S from $C \setminus S$, as going around S (through the edges of $E(S, V \setminus C)$ instead of the edges of $E(S, C \setminus S)$) cuts though (approximately) fewer edges.

Intuitively, a set $U \subsetneq C$ is $(1-\delta)$-boundary sparse in C if it (or its complement in C) is less connected to C than it is to its complement in C. To understand why this is useful, one should consider cuts that are *not* boundary sparse.

Indeed, intuitively, such cuts are equally or more connected to the outside of the cluster, than they are to the inside of the cluster. Thus, this cut can be approximated by simply going around the cluster, a procedure we call *uncrossing*:

Definition 2.4.5 (Crossing and uncrossing)**.** *We say that a cut S* crosses *cluster C if both $S \cap C$ and $C \setminus S$ are nonempty. A cut S'* uncrosses a cut S in a cluster C *if S crosses C and S' either equals $S \cup C$ or $S \setminus C$.*

For $\delta \lesssim \frac{1}{\lambda}$, the approximation ratio is small enough to ensure that, in fact, the approximate cut has the exact same size as the original cut.

Considering only these $(1-\delta)$-boundary sparse cuts[5], we are able to show that the number of *intercluster* edges – the edges that are incident to two different clusters – is only a $\tilde{O}(\frac{1}{\delta^3})$ factor larger than the initial number of interexpander edges, which ensures that contracting each cluster sufficiently reduces the number of edges in the recursive call, and thus allows us to efficiently bound the number of recursive calls. The algorithm is represented in Figure 2.11

How to dynamize the algorithm: While the expander decomposition is itself fully-dynamic, we have only discussed how to find and cut along $(1-\delta)$-boundary sparse cuts staticly. We describe next the challenges we faced to make it dynamic:

(1) Finding all boundary sparse cuts of size at most $\lambda_{\max}$ while edges are inserted and deleted.

(2) As our hierarchy structure is recursive, we have to make sure that we do not get a multiplicative blowup in the number of update operations per hierarchy level, as well as the depth of our recursion. More specifically, one update on the top level of the hierarchical decomposition can lead to up to $h > 1$ changes to the next level and, thus, h^r many on the bottom level. We show that for our decomposition, the depth r of the recursion is $\log^{1/4} n$, and the number of changes is $h = 2^{O(\log^{1/2} n)}$, so that $h^r = 2^{O(\log^{3/4} n)} = n^{o(1)}$.

[5]This has to be done in a correct order, which is discussed in the full chapter

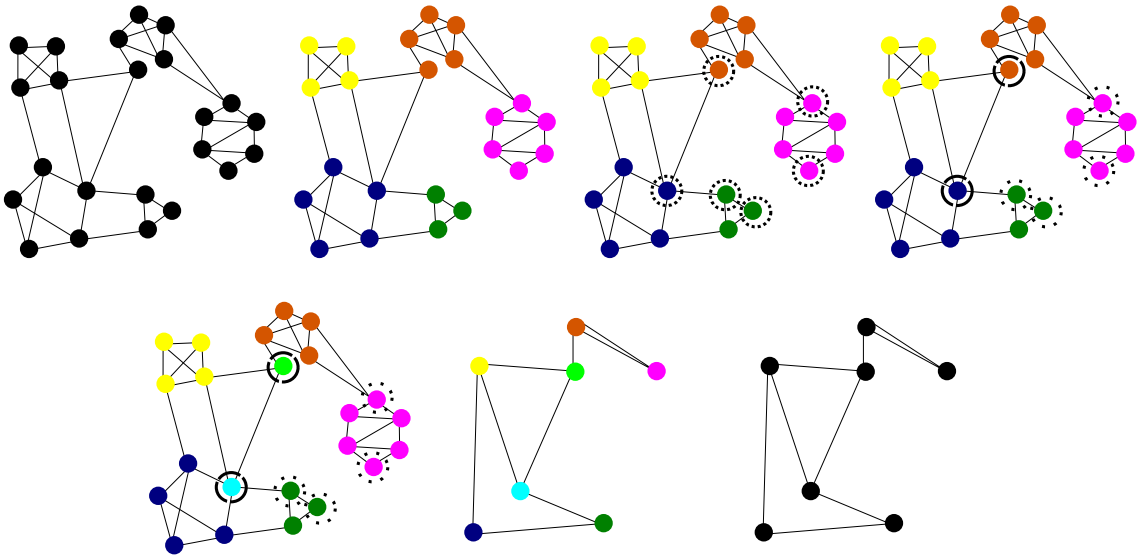


Figure 2.11: Main steps of the minimum cut algorithm. From left to right, top to bottom: Initial graph. Expander Decomposition. Run LocalKCut from every vertex. Select only $(1-\epsilon)$-boundary sparse cuts. Separate cuts from expanders to create clusters. Contract Clusters. Graph on which to recurse.

(3) We need to watch out for the case where the minimum cut uncrosses in a way that one side of the cut becomes the empty graph.

(4) We need to maintain a data structure that enables us to reach and update the values and objects (nodes, edges, clusters) whenever needed.

We now explain how we address each of the above challenges:

(1) Boundary-sparse cuts were used in the static setting to compute a minimum cut in [82]. The algorithm in [82] does not use an expander decomposition as it runs in deterministic time $\tilde{O}(m)$ and no deterministic algorithm for computing an expander decomposition is known in that running time. Thus, it had to find boundary-sparse cuts with arbitrarily large volume on either side of the cut, while we use the crucial observation that in a subset of an expander it is sufficient to find *unbalanced* boundary-sparse cuts, i.e. cuts where the volume on one of the sides is small, as discussed above. This allows us to deal with edge insertions and deletions as follows: We maintain the invariant that no cluster contains a $(1-\epsilon)$-boundary sparse cut of size at most $\lambda_{\max}$. Then, after each insertion or deletion, we only have to check *locally* (i.e. at the endpoints of the updated edge) using the LocalKCut algorithm if the invariant still holds, and, if not, enforce it by cutting any $(1-\epsilon)$-boundary sparse cut that LocalKCut may find. The expander property here again is key, as the volume we need to explore is $\frac{\lambda_{\max}}{\phi} = \lambda_{\max} \cdot n^{o(1)}$. Note that this is the first use of LocalKCut in the dynamic setting.

(2) By cutting in this way, however, we need to make sure that we do not cut too many edges, as this might either make the depth of our hierarchy too large, or the number of updates per level too high. To deal with this, we use the fact that the dynamic expander decomposition of [77] is recomputed from scratch every $\Theta(\frac{m}{n^{o(1)}})$ updates, called a *restart* of the cluster decomposition. The number of updates is carefully chosen so as to, on the one side, not end

up with too many inter-cluster edges and, on the other side, amortize the running time accrued since the last restart over these updates. Using the restarts and a potential function argument, we can bound the number of inter-cluster edges per level at any point in time to be only a factor $\tilde{O}(\frac{1}{\delta^3})$ larger than the sum of the number of *inter-expander* edges plus the recourse of the expander decomposition at the same point in time. This implies that the number of inter-cluster edges decreases suitably when going from one level to the next, and, thus, limits the depth r of our hierarchy to $\log^{1/4} n$. We can also bound the amortized number of updates per level by $2^{O(\log^{1/2} n)}$.

(3) To make sure that we catch the minimum cuts that might uncross to the empty cut, we build, for each cluster C, a copy of the graph where all the nodes outside of the cluster are contracted into one node. We call it the *mirror cluster* of cluster C. We can show that if a minimum cut uncrosses to the empty cut for a cluster C, then the mirror cluster of C must have a cut of similar size and volume $n^{o(1)}$ and, thus, can be found efficiently using LocalKCut. Note that we also have to maintain the mirror clusters dynamically. These graphs were not used in prior minimum cut algorithms.

(4) Our data structure essentially keeps the cluster decomposition, and along with each cluster, its corresponding mirror cluster, its set of outgoing edges and the number of outgoing edges. To maintain this data structure efficiently after edge updates, we guarantee that between two consecutive restarts, we only refine our decomposition, i.e., we *do not merge clusters, even if the expander decomposition would do so*. Indeed, the expander decomposition might need to merge clusters back together to maintain the expansion property, but since we only need to maintain that each cluster is a subset of an expander, merging back is unnecessary. Since the expander decomposition creates $\tilde{O}(m\phi)$ inter-cluster edges on each rebuild, and has amortized recourse $\rho = 2^{\Theta(\sqrt{\log n})}$ per edge update over a period of $O(\frac{m\phi}{\rho})$ updates until the next rebuild, the number of inter-expander edges that exist at any point between two consecutive rebuilds is $\tilde{O}(m\phi)$. Hence the number of intercluster edges in the refinement of the expander decomposition that our algorithm maintains is $\tilde{O}(m\phi)$. Our data structure allows us to split a cluster C in time linear in $\mathrm{Vol}(S)$, where S is the side of the cut of smaller volume. As each edge can be on the smaller side of the cut only $O(\log m)$ times, and the total time for maintaining our structure is $\tilde{O}(m)$, we can amortize that time over $\Theta(\frac{m\phi}{\rho})$ updates and achieve an $n^{o(1)}$ amortized time per update.

Master Algorithm. Finally, we describe how to use the algorithm for a limited choice of λ to cover the case where λ can be as large as polynomial in n, and in the case where the edges are weighted:

Our first step is to sparsify the graph as to ensure that the minimum cut value is reduced to at most $2^{\Theta(\log^{3/4} n)}$ using the edge-sampling approach of Karger [93]. It samples each edge with a probability p aptly chosen as described in the following lemma:

Theorem 2.4.6 (Karger sparsification, Theorem 2.1 and Corollary 2.1 of [93])**.** *Let G be any graph with minimum cut λ and let $p \geq \frac{54 \ln n}{\epsilon^2 \lambda}$. Let $G(p)$ be the graph obtained from G by sampling each edge independently with probability p. Then with high probability, the minimum cut S in $G(p)$ will correspond to a $(1+\epsilon)$-minimum cut of G. Moreover, also with high probability, the value of* any *cut S in $G(p)$ has value no less than $(1-\epsilon)$ and no more than $(1+\epsilon)$ times its expected value.*

Ensuring that the minimum cut is small is crucial for our analysis as our running time depends on the size of the minimum cut. However, one challenge with sparsification is that it is hard to dynamize whenever the value of the minimum cut changes, and, thus, p has to change as well. In that case, one update to the graph can lead to many updates to the sparsified graph, which we cannot afford. To deal with this problem we run $O(\log n)$ many algorithms, each with a different value of p, in parallel, and we run a *master algorithm* to decide which answer to return based on the answers of these algorithms. In the following we use λ to denote the minimum cut in the full graph[6]. Define $b_i = 1.1^i$ for all integers $i \in [1, \log_{1.1} n^2]$ and run one algorithm for each index i. The i-th algorithm, i.e., the algorithm for index i, assumes that the value λ of the minimum cut belongs to the range $[b_i, b_{i+1})$ (called the *range* of the ith-algorithm), and sparsifies with $p_i = \frac{54 \ln n}{\epsilon^2 b_i}$ to create graph G_i. It follows that, if $\lambda \in [b_i, b_{i+1})$, then the minimum cut λ_i in the sampled graph G_i is a $(1+\epsilon)$-approximate minimum cut in the full graph and its size in G_i belongs to the range $[(54(1-\epsilon)\ln n)/\epsilon^2, (54 \cdot 1.1(1+\epsilon)\ln n)/\epsilon^2]$. Thus, for each $i \in [1, \log_{1.1} n^2]$ the i-th algorithm calls a subroutine – which is the central algorithm of our paper, we call it the *bounded mincut algorithm* – with two parameters $\lambda_{\min}$ and $\lambda_{\max}$, on G_i. The bounded mincut algorithm's requirements are:

1. If $\lambda_i < \lambda_{\min}$: any output is acceptable.
2. If $\lambda_{\min} \leq \lambda_i \leq \lambda_{\max}$, output an approximation of the minimum cut.
3. If $\lambda_i > \lambda_{\max}$, output "$\lambda_i > \lambda_{\max}$".

As per our above discussion, we choose $\lambda_{\min} = (54(1-\epsilon)\ln n)/\epsilon^2$ and $\lambda_{\max} = (54 \cdot 1.1(1+\epsilon)\ln n)/\epsilon^2$.

Hence, out of all the i-th algorithms, at least one algorithm outputs a correct approximation. There is, however, only one i such that $\lambda \in [b_i, b_{i+1})$. It thus remains to decide which answer out of the $O(\log n)$ many answers returned by the $O(\log n)$ algorithms should be returned by the master algorithm.

For that, we design the algorithms so that for every i if $b_i \leq \lambda$ (which corresponds w.h.p. to $\lambda_{\min} \leq \lambda_i$), the i-th algorithm outputs a correct answer, more specifically it either outputs a $(1+\epsilon)$-approximation of λ or it outputs that the minimum cut in G_i is larger than $\lambda_{\max}$ (called the *correctness requirement* of the algorithm). However, if $b_i > \lambda$ (which corresponds w.h.p. to $\lambda_{\min} > \lambda_i$), the algorithm can output an arbitrary value. The reason for this is as follows: A b_i value that is much larger than λ can lead to undersampling of the graph, and, thus, the minimum cut in G_i can deviate too much from its expected value. On the other side, a b_i value that is smaller than λ results – in the worst case – in oversampling, which is not a problem for correctness. The only issue that might arise is that the minimum cut λ_i in the sampled graph G_i is larger than $\lambda_{\max}$, but our algorithm is required to detect this. Since, by Theorem 6.2.5 and our choice of $\lambda_{\min}$, $\lambda_{\max}$, and p_i, it holds w.h.p. for the unique i with $b_i \leq \lambda < b_{i+1}$ that (a) $\lambda_{\min} \leq \tilde{\lambda} \leq \lambda_{\max}$ and that (b) we did not undersample, the master algorithm would output an $(1+\epsilon)$-approximation of λ if it output the value returned

[6]We will denote λ the size of the minimum cut in the original graph, and $\tilde{\lambda}$ the size of the minimum cut in the sparsified graph. We further specify λ_i to denote the size of the minimum cut in the sparsifed graph G_i when necessary. b_i is a fixed parameter of algorithm i.

by the i-th algorithm. However, many algorithms might return a value (instead of returning just the fact $\lambda > \lambda_{\max}$ (case 3 above)) and, thus, it is not possible for the master algorithm to determine i. This problem is resolved as follows: As all algorithms j with $b_j \leq \lambda$ give a correct answer and, for $\epsilon \leq 0.1$, w.h.p. for only two such algorithms it can be the case that $\lambda_j \leq \lambda_{\max}$ (by the choice of $\lambda_{\max}$), namely for $j = i$ and $j = i - 1$. Thus, at most two algorithms with $b_j \leq \lambda$ return a value that is at most $\lambda_{\max}$, all other such algorithms will either return a value that is larger than $\lambda_{\max}$ or will state that $\lambda_j > \lambda_{\max}$. Thus the master algorithm takes the output of the algorithm with the smallest j that did not output that $\lambda_j > \lambda_{\max}$ or return a value larger than $\lambda_{\max}$. This guarantees that either the output of algorithm i or algorithm $i - 1$ is returned.

This then results in a $(1+\epsilon)^2$-approximation of λ. The overlap in the ranges of the algorithms is needed to ensure that we catch all the corner cases.

CHAPTER 3

Asymptotically Tight Bounds on the Time Complexity of Broadcast and its Variants in Dynamic Networks

This chapter appeared in parts or fully in

Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Brief Announcement: Broadcasting Time in Dynamic Rooted Trees is Linear". In: *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*. Ed. by Alessia Milani and Philipp Woelfel. ACM, 2022, pp. 54–56. DOI: 10.1145/3519270.3538460. URL: https://doi.org/10.1145/3519270.3538460

and

Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Asymptotically Tight Bounds on the Time Complexity of Broadcast and Its Variants in Dynamic Networks". In: *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*. ed. by Yael Tauman Kalai. Vol. 251. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 47:1–47:21. DOI: 10.4230/LIPICS.ITCS.2023.47. URL: https://doi.org/10.4230/LIPIcs.ITCS.2023.47

3.1 Abstract

Data dissemination is a fundamental task in distributed computing. This paper studies *broadcast problems* in various innovative models where the communication network connecting n processes is dynamic (e.g., due to mobility or failures) and controlled by an adversary.

In the first model, the processes transitively communicate their ids in synchronous rounds along a rooted tree given in each round by the adversary whose goal is to maximize the number of rounds until *at least one id is known by all processes*. Previous research has shown a $\lceil \frac{3n-1}{2} \rceil - 2$ lower bound and an $O(n \log \log n)$ upper bound. We show the first linear upper bound for this problem, namely $\lceil (1+\sqrt{2})n - 1 \rceil \approx 2.4n$.

We extend these results to the setting where the adversary gives in each round k-disjoint forests and their goal is to maximize the number of rounds until there is a set of k ids such that *each process knows of at least one of them*. We give a $\left\lceil \frac{3(n-k)}{2} \right\rceil - 1$ lower bound and a $\frac{\pi^2+6}{6}n + 1 \approx 2.6n$ upper bound for this problem.

Finally, we study the setting where the adversary gives in each round a directed graph with k roots and their goal is to maximize the number of rounds until *there exist k ids that are known by all processes*. We give a $\left\lceil \frac{3(n-3k)}{2} \right\rceil + 2$ lower bound and a $\lceil (1+\sqrt{2})n \rceil + k - 1 \approx 2.4n + k$ upper bound for this problem.

For the two latter problems no upper or lower bounds were previously known.

3.2 Introduction

Data dissemination is one of the most fundamental tasks in distributed systems. This paper studies data dissemination in an innovative model where the communication network connecting n processes is *dynamic*. In particular, we consider a worst-case perspective and assume that the information flow between the processes is controlled by an *oblivious message adversary* which may drop an arbitrary set of messages sent by some processes in each round. This results in a sequence of directed communication graphs, whose edges tell which process can successfully send a message to which other process in a given round. The oblivious message adversary model is appealing because it is conceptually simple and still provides a highly dynamic network model: The set of allowed graphs can be arbitrary, and the nodes that can communicate with one another can vary greatly from one round to the next. It is, thus, well-suited for settings where significant transient message loss occurs, such as in wireless networks subject to interference, jamming, or mobility.

We look into three data dissemination problems in dynamic networks: *broadcast, cover* and *k-broadcast*. These problems come in many flavors and feature intriguing connections to other classic problems such as leader(s) election, regular and k-set consensus (also known as k-set agreement), for which our problems' time complexity is typically a lower bound.

In particular, we assume that each process has a unique id and in every message each process communicates all the ids it knows of so far. We first study a fundamental model where communication happens along arbitrary rooted trees, chosen by an adversary who aims to maximize the *broadcast time*, which is the number of rounds it takes until there exists a process that everyone knows of. We then extend our investigations to sparser networks, considering k-forests (a union of k rooted trees). Here the adversary will maximize the *cover time*, which is the number of rounds it takes until there exists k process such that everyone knows of at least one of them. Moreover, in more highly connected networks, we study k-rooted dynamic networks (directed graphs with k roots), where the adversary aims at maximizing the *k-broadcast time*, which is the number of rounds it takes until there exist k processes that everyone knows of. Before presenting our results, we introduce our model more formally.

Model Let n be the number of processes and let each process have a unique identifier from $[n]$. Let $\mathcal{G}$ be a fixed set of directed networks with n nodes such that each node has a unique identifier from $[n]$. There will be a sequence of rounds $t = 1, 2, \ldots$, such that, in each round

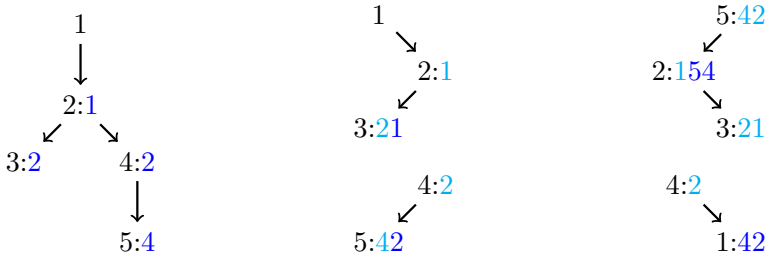


Figure 3.1: Example of information propagation. Self-loops are omitted, but are present on every node in every round. $x : yz$ means that y was an in-neighbor of x before the current round (and still is), whereas z is a new in-neighbor of x. We omit mentioning x in its own in-neighbors as it is always the case.

t, an adversary chooses a network $\tilde{G}_t$ from $\mathcal{G}$, which determines the communication links in round t as follows: Initially every process *knows* (or *has heard*) of its own identifier. During round t process i sends all identifiers it knows of to its out-neighbors in $\tilde{G}_t$. The rounds stop whenever the objective – broadcast, k-broadcast or cover of size k – is attained. The goal of the adversary is to maximize the number of rounds.

To model the information propagation we use graph products:

Definition 3.2.1 (Graph product)**.** *If $A = ([n], E_1)$ and $B = ([n], E_2)$ are two directed networks on n nodes, then the* product graph *$A \circ B$ is the network on n nodes, with edge set E, where $(x, y) \in E$ if and only if there exist a node z such that $(x, z) \in E_1$ and $(z, y) \in E_2$.*

Consider round t and let $G(t)$ be the *product graph* $G(t) = G_1 \circ \ldots \circ G_t$, where G_i is created from $\tilde{G}_i$ by adding a self-loop to every node. Note that in $G(t)$ the in-neighbors of a process x are exactly the processes that x knows of after t rounds, and its out-neighbors are the processes it has sent information to. We added a self-loop to every node in G_i to capture the fact that no process "forgets" any piece of information in any round. An example is given in Figure 3.1.

We will consider three different models, each of which has a different objective, detailed below. Figure 3.2 summarizes the 3 models, gives examples and states the results.

Broadcasting on Trees

Definition 3.2.2 (Set of rooted trees with self-loops)**.** *Let $\mathcal{T}_n$ be the set of all rooted trees with a self-loop added at every node.*

In this variant, the networks given by the adversary are restricted to trees in $\mathcal{T}_n$ and we analyze the *broadcasting time* t^*, which is the smallest round t such that there exists a node in $G(t)$ with an out-edge to every other node. Note that this corresponds to a process such that every other process has heard of its identifier. We will say that *the node has broadcast (its identifier to everyone)* or simply *broadcast has happened*. Trees being rooted ensure that broadcast happens in a finite number of rounds[1].

[1] If the adversary can choose a non-rooted graph, it could repeat this graph indefinitely, preventing broadcast.

Definition 3.2.3 (Broadcast time of a sequence of graphs)**.** *The broadcast time $t^*(G_1, G_2, \ldots)$ of a sequence of graphs $G_1, G_2, \ldots$, is*

$$t^*(G_1, G_2, \ldots) = \min\{t \in \mathbb{N} : \exists x \in [n], \forall y \in [n], (x, y) \in G_1 \circ \ldots \circ G_t\}$$

Definition 3.2.4 (Broadcast time of an adversary)**.** *The broadcast time $t^*(\mathcal{G})$ of an adversary $\mathcal{G}$, is defined as follows:*

$$t^*(\mathcal{G}) = \max\{t^*(G_1, G_2, \ldots) : \forall i \in \mathbb{N}, G_i \in \mathcal{G}\}$$

We will give tight asymptotic bounds on $t^*(\mathcal{T}_n)$. Note that even in the simple case where the adversary gives the same directed tree in each round, the broadcast time can be as large as $n - 1$, namely if the tree is simply a path. Conversely, in each round, it is easy to see that at least one new edge appears in the product graph[2], and thus the broadcast time is at most n^2. This raises the question of how large the broadcast time can be made if in each round a different directed tree can be used.

This has been an open question for several years. Results from Charron-Bost and Schiper in 2009 [34] and Charron-Bost, Függer, and Nowak in 2015 [33] imply an $n \log n$ upper bound. In 2019, Zeiner, Schwarz, and Schmid [129] gave a linear upper bound when the adversary is restricted to trees with either a constant number of leaves or a constant number of inner nodes. They also gave a $\left\lceil \frac{3n-1}{2} \right\rceil - 2$ lower bound. In 2020, Függer, Nowak, and Winkler [68] improved the general upper bound to $2n \log \log n + O(n)$. So far, it has been an open conjecture [129] whether the broadcast time is linear for arbitrary sequences of rooted trees.

Covering on k-forests

Definition 3.2.5 (k-forests with self-loops)**.** *We define $\mathcal{F}_n^k$ to be the set of all forests over n processes which are the union of k rooted trees and a self-loop is added at every node.*

In this variant, the networks given by the adversary are restricted to networks from $\mathcal{F}_n^k$, and we analyze the *cover time* t_k^c, which is the smallest round t such that there exists a set $I \subset [n]$, $|I| \leq k$, such that every node x of $G(t)$ has at least one in-neighbor from I. Said differently, there exists a set of k nodes such that every node knows of *at least one* of them.

Definition 3.2.6 (Cover time of a sequence of graphs)**.** *The cover time $t_k^c(G_1, G_2, \ldots)$ of a sequence of graphs $G_1, G_2, \ldots$, is*

$$t_k^c(G_1, G_2, \ldots) = \min\{t \in \mathbb{N} : \exists x_1, \ldots, x_k \in [n], \forall y \in [n], \exists i \in [k], (x_i, y) \in G_1 \circ \ldots \circ G_t\}$$

Definition 3.2.7 (Cover time of an adversary)**.** *The cover time $t_k^c(\mathcal{G})$ of an adversary $\mathcal{G}$, is defined as follows:*

$$t_k^c(\mathcal{G}) = \max\{t_k^c(G_1, G_2, \ldots) : \forall i \in \mathbb{N}, G_i \in \mathcal{G}\}$$

We will give tight asymptotic bounds on $t_k^c(\mathcal{F}_n^k)$. To the best of our knowledge, there is no prior work that gives upper or lower bounds on $t_k^c(\mathcal{F}_n^k)$.

[2] In each round, the identifier of the root reaches someone new.

k-broadcasting on k-rooted networks

Definition 3.2.8 (k-rooted Networks)**.** *Let $\mathcal{R}_n^k$ be the set of all (directed) networks over n nodes that (1) have k roots, that is k different processes $r_1, \ldots, r_k$ such that there exists a directed path from any r_i to any process in $[n]$, and (2) have a self-loop at every node.*

In this variant, the networks given by the adversary are restricted to networks from $\mathcal{R}_n^k$, and we analyze the *k-broadcasting time* t_k^*, which is the smallest round t such that there exist k nodes in $G(t)$ with an out-edge to every other node. Said differently, there exists a set of k nodes such that every node knows of *all* of them.

Definition 3.2.9 (k-broadcast time of a sequence of graphs)**.** *The k-broadcast time $t_k^*(G_1, G_2, \ldots)$ of a sequence of graphs $G_1, G_2, \ldots$, is*

$$t_k^*(G_1, G_2, \ldots) = \min\{t \in \mathbb{N} : \exists x_1, \ldots, x_k \in [n], \forall i \neq j, x_i \neq x_j \\ \wedge \forall y \in [n], \forall i \in [k], (x_i, y) \in G_1 \circ \ldots \circ G_t\}$$

Definition 3.2.10 (k-broadcast time of an adversary)**.** *The k-broadcast time $t_k^*(\mathcal{G})$ of an adversary $\mathcal{G}$, is defined as follows:*

$$t_k^*(\mathcal{G}) = \max\{t_k^*(G_1, G_2, \ldots) : \forall i \in \mathbb{N}, G_i \in \mathcal{G}\}$$

We will give tight asymptotic bounds on $t_k^*(\mathcal{R}_n^k)$. To the best of our knowledge, there is no prior work that gives upper or lower bounds on $t_k^*(\mathcal{R}_n^k)$.

Contribution We give asymptotically tight bounds for all three settings, see also Figure 3.2.

(1) First we settle the open problem about time complexity of broadcast in dynamic trees, by showing that it is *linear*. Hence, Zeiner et al.'s [129] conjecture is true. In particular, we present an upper bound of $\left\lceil (1+\sqrt{2})n \right\rceil \approx 2.4n$, which complements their $\left\lceil \frac{3n-1}{2} \right\rceil - 2$ lower bound.

(2) We further show that covering on k-forests also takes linear time, by giving a $\frac{\pi^2+6}{6}n + 1 \approx 2.6n$ upper bound and a $\left\lceil \frac{3(n-k)}{2} \right\rceil - 1$ lower bound.

(3) Finally, we show that k-broadcasting on k-rooted networks is linear as well, by giving an upper bound of $\left\lceil (1+\sqrt{2})n \right\rceil + k - 1 \approx 2.4n + k$, and a lower bound of $\left\lceil \frac{3(n-3k)}{2} \right\rceil + 2$.

Organization The remainder of this paper is organized as follows. Section 3.3 introduces some basic tools that will be useful throughout the paper, and presents first insights. In Section 3.4, we give the linear upper bound for broadcast time in our model. Section 3.5 and Section 3.6 respectively showcase our results for the cover on k-forests and the k-broadcast on k-rooted networks. After reviewing related work in Section 3.7, we conclude and provide some future research directions in Section 3.8.

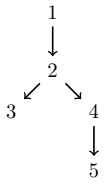
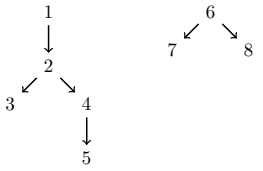
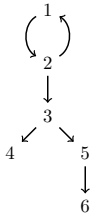
Model:	Broadcasting on Trees	Covering on k-Forests	k-Broadcasting on k-rooted Networks
Adversary:	 Rooted trees over n nodes. Example for $n = 5$	 Forests over n processes, composed of k rooted trees. Example for $n = 8$ and $k = 2$	 Networks over n processes and k roots. Example for $n = 6$ and $k = 2$
Objective:	$1 : 1\underline{2}$ $2 : \underline{2}3$ $3 : \underline{2}3$ $4 : \underline{2}34$ Broadcast Example for $n = 4$	$1 : \underline{1}2$ $2 : 2\underline{4}$ $3 : \underline{1}23$ $4 : 3\underline{4}$ Cover of size k Example for $n = 4$ and $k = 2$	$1 : 1\underline{24}$ $2 : \underline{24}$ $3 : \underline{2}3\underline{4}$ $4 : 1\underline{24}$ k-broadcast of k processes Example for $n = 4$ and $k = 2$
Lower Bound:	[129]: $\left\lceil \frac{3n-1}{2} \right\rceil - 2$	$\left\lceil \frac{3(n-k)}{2} \right\rceil - 1$	$\left\lceil \frac{3(n-3k)}{2} \right\rceil + 2$
Upper Bound:	[68]: $O(n \log \log n)$ $\left\lceil (1+\sqrt{2})n \right\rceil$	$\frac{\pi^2+6}{6}n + 1$	$\left\lceil (1+\sqrt{2})n \right\rceil + k - 1$

Figure 3.2: Summary of models and results. The adversary chooses a sequence of graphs as stated in order to delay the objective as much as possible. In the objective examples, $x : y$ means that y is an in-neighbour of x at the round the objective is attained. Underlines are for emphasis purposes only.

3.3 Basic Tools

In this section, we define generalizations of the in- and out-neighborhoods in the product graph, and some basic properties these tools follow.

Definition 3.3.1 (In-neighbourhood)**.** *The in-neighborhood of process x between rounds t and t', $t, t' \geq 0$, denoted by $\mathcal{I}_t^{t'}(x)$, is defined as follows: If $t \leq t', \mathcal{I}_t^{t'}(x)$ is the in-neighborhood of process x in the graph $G_t \circ \ldots \circ G_{t'}$. If $t = t' + 1$, set $\mathcal{I}_t^{t'}(x) = \{x\}$. Otherwise, $\mathcal{I}_t^{t'}(x) = \varnothing$.*

Definition 3.3.2 (Out-neighbourhood)**.** *The out-neighborhood of process x between rounds t and t', $t, t' \geq 0$, denoted by $\mathcal{O}_t^{t'}(x)$, is defined as follows: If $t \leq t', \mathcal{O}_t^{t'}(x)$ is the out-neighborhood of process x in the graph $G_t \circ \ldots \circ G_{t'}$. If $t = t' + 1$, set $\mathcal{O}_t^{t'}(x) = \{x\}$. Otherwise, $\mathcal{O}_t^{t'}(x) = \varnothing$.*

We prove that these generalized neighborhoods behave as expected:

Lemma 3.3.3. *Let $t, t' \geq 0, x, y \in [n]$. Then $x \in \mathcal{O}_t^{t'}(y) \Leftrightarrow y \in \mathcal{I}_t^{t'}(x)$.*

Proof. If $t > t' + 1$, the equivalence is empty. If $t = t' + 1$, then $x \in \mathcal{O}_t^{t'}(y) \Leftrightarrow x = y \Leftrightarrow y \in \mathcal{I}_t^{t'}(x)$. If $t \leq t'$, $x \in \mathcal{O}_t^{t'}(y) \Leftrightarrow (y, x) \in G_t \circ \ldots \circ G_{t'} \Leftrightarrow y \in \mathcal{I}_t^{t'}(x)$. □

Lemma 3.3.4 (Transitivity). *Let $t, t', t'' \geq 0$, and $x, y, z \in [n]$. We have the following properties:*

i. *If $y \in \mathcal{O}_t^{t'}(x)$ and $z \in \mathcal{O}_{t'+1}^{t''}(y)$, then $z \in \mathcal{O}_t^{t''}(x)$.*

ii. *If $x \in \mathcal{I}_t^{t'}(y)$ and $z \in \mathcal{O}_{t'+1}^{t''}(y)$,then $z \in \mathcal{O}_t^{t''}(x)$.*

iii. *If $x \in \mathcal{I}_t^{t'}(y)$ and $y \in \mathcal{I}_{t'+1}^{t''}(z)$, then $z \in \mathcal{O}_t^{t''}(x)$.*

Proof. *ii.* and *iii.* will follow from *i.* and Lemma 3.3.3, so we only prove *i.*.

Since $y \in \mathcal{O}_t^{t'}(x)$, this means $\mathcal{O}_t^{t'}(x) \neq \varnothing$. Therefore, $t \leq t' + 1$. If $t = t' + 1$, we have that $\mathcal{O}_t^{t'}(x) = \{x\} \Rightarrow y = x$, and thus $z \in \mathcal{O}_t^{t''}(x)$. We will then only need to consider the case $t \leq t'$.

Similarly, since $z \in \mathcal{O}_{t'+1}^{t''}(y)$, we have that $\mathcal{O}_{t'+1}^{t''}(y) \neq \varnothing$ and thus $t' + 1 \leq t'' + 1$. If $t' = t''$, we have that $\mathcal{O}_{t'+1}^{t''}(y) = \{y\}$ thus $z = y$ and therefore $z \in \mathcal{O}_t^{t''}(x)$. Hence we only need to consider the case $t \leq t' \leq t'' - 1$.

In that last case, we know that (x, y) is an edge in $G_t \circ \ldots \circ G_{t'}$ and (y, z) is an edge in $G_{t'+1} \circ \ldots \circ G_{t''}$. By definition of a product graph, this means that (x, z) is an edge in $G_t \circ \ldots \circ G_{t''}$, and thus that $z \in \mathcal{O}_t^{t''}(x)$. □

And finally, we show that these neighborhoods only grow over time:

Lemma 3.3.5 (Monotonicity). *If in each round, all nodes have a self-loop, then for any $t_1 \leq t_2$ and $t_3 \leq t_4$, for any process x we have:*

i. $\mathcal{I}_{t_2}^{t_3}(x) \subseteq \mathcal{I}_{t_1}^{t_4}(x)$.

ii. $\mathcal{O}_{t_2}^{t_3}(x) \subseteq \mathcal{O}_{t_1}^{t_4}(x)$.

Proof. Let us first consider the case $t_2 > t_3 + 1$. Then $\mathcal{I}_{t_2}^{t_3}(x) = \mathcal{O}_{t_2}^{t_3}(x) = \varnothing$ and the result is trivial. Similarly, if $t_2 = t_3 + 1$, then $\mathcal{I}_{t_2}^{t_3}(x) = \mathcal{O}_{t_2}^{t_3}(x) = \{x\}$, and then either $t_1 = t_4 + 1$ which means $\mathcal{I}_{t_1}^{t_4}(x) = \mathcal{O}_{t_1}^{t_4}(x) = \{x\}$, or $t_1 \leq t_4$, and then x has a self-loop in G_t for every $t_1 \leq t \leq t_4$, which implies $x \in \mathcal{O}_{t_1}^{t_4}(x)$ and $x \in \mathcal{I}_{t_1}^{t_4}(x)$.

Let's now consider the case $t_2 \leq t_3$. Let $y \in \mathcal{I}_{t_2}^{t_3}(x)$ (respectively $z \in \mathcal{O}_{t_2}^{t_3}(x)$). Then edge (y, x) ((x, z)) is in graph $G_{t_2} \circ \ldots \circ G_{t_3}$. Since all rounds have self-loops, we clearly have that edge (y, y) ((x, x)) is in the graph $G_{t_1} \circ \ldots \circ G_{t_2}$. Similarly, edge (x, x) ((z, z)) is in graph $G_{t_3} \circ \ldots \circ G_{t_4}$. This shows that (y, x) ((x, z)) is in graph $G_{t_1} \circ \ldots \circ G_{t_4}$ and thus $y \in \mathcal{I}_{t_1}^{t_4}(x)$ ($z \in \mathcal{O}_{t_1}^{t_4}(x)$). □

3.4 Broadcasting on Trees

In this section, we focus on the fundamental problem of broadcasting on dynamic trees. We give an upper bound for the problem, before recalling a lower bound.

3.4.1 The Upper Bound

We will show that the key to understand how information propagate is to consider what the root knows – or the in-neighbors of the root – before the beginning of every round. We will show that the root must either have a lot of in-neighbors that were roots in previous rounds, or many in-neighbors in general. We will then show that any in-neighbor of the root before a round has at least one more out-neighbor after the round than before it. We will finally show that any adversary that tries to balance these two facts will fail to prevent broadcast for a time longer than linear.

Definition 3.4.1 (Root of a round)**.** *Let t be a round. We denote by r_t the root of G_t, and call it the root of the round t.*

Harnessing the fact that there always exists a path from a root to any other process in a network, we give the two following lemmas:

Lemma 3.4.2. *Let t, t' be rounds such that $t \leq t'$. We have that:*

i If x is a process such that $r_t \notin \mathcal{I}_{t+1}^{t'}(x)$, then $\left|\mathcal{I}_t^{t'}(x)\right| > \left|\mathcal{I}_{t+1}^{t'}(x)\right|$.

ii If x is a process such that $r_{t'} \in \mathcal{O}_t^{t'-1}(x)$, then $\left|\mathcal{O}_t^{t'}(x)\right| > \left|\mathcal{O}_t^{t'-1}(x)\right|$, unless $\left|\mathcal{O}_t^{t'-1}(x)\right| = n$.

Proof. i. We will show that there exists a process $y \in \mathcal{I}_{t+1}^{t'}(x)$ that has an in-neighbor z in G_t such that $z \notin \mathcal{I}_{t+1}^{t'}(x)$. Then, by Transitivity (Lemma 3.3.4), $z \in \mathcal{I}_t^{t'}(x)$. By Monotonicity (Lemma 3.3.5), we have $\mathcal{I}_{t+1}^{t'}(x) \subset \mathcal{I}_t^{t'}(x)$, this will show that $\left|\mathcal{I}_t^{t'}(x)\right| > \left|\mathcal{I}_{t+1}^{t'}(x)\right|$.

Let us now find such a z. Consider the path from r_t to x in G_t. Since $r_t \notin \mathcal{I}_{t+1}^{t'}(x)$, and trivially $x \in \mathcal{I}_{t+1}^{t'}(x)$, this path must include an edge (z, y) such that $z \notin \mathcal{I}_{t+1}^{t'}(x)$, $y \in \mathcal{I}_{t+1}^{t'}(x)$.

ii. Let us look at the case $\left|\mathcal{O}_t^{t'-1}(x)\right| < n$. We will show that there exists a process $y \in \mathcal{O}_t^{t'-1}(x)$ that has an out-neighbor z in $G_{t'}$ such that $z \notin \mathcal{O}_t^{t'-1}(x)$. Then, by Transitivity (Lemma 3.3.4), $z \in \mathcal{O}_t^{t'}(x)$. By Monotonicity (Lemma 3.3.5), we have $\mathcal{O}_t^{t'-1}(x) \subset \mathcal{O}_t^{t'}(x)$, this will show that $\left|\mathcal{O}_t^{t'}(x)\right| > \left|\mathcal{O}_t^{t'-1}(x)\right|$.

Let us now find such a z. Since $\left|\mathcal{O}_t^{t'-1}(x)\right| < n$, there exists a process a such that $a \notin \mathcal{O}_t^{t'-1}(x)$. Consider the path from $r_{t'}$ to a in $G_{t'}$. Since $r_{t'} \in \mathcal{O}_t^{t'-1}(x)$, and $a \notin \mathcal{O}_t^{t'-1}(x)$, this path must include an edge (y, z) such that $z \notin \mathcal{O}_t^{t'-1}(x)$, $y \in \mathcal{O}_t^{t'-1}(x)$. □

The following lemma will link the number of in-neighbors a node has to the number of in-neighbors it has among the roots of the preceding rounds:

Lemma 3.4.3. *Let x be a process, and t_1, t_2 be rounds such that $t_1 \leq t_2$. Then:*

$$\left|\{t : t_1 \leq t \leq t_2, r_t \notin \mathcal{I}_{t_1}^{t_2}(x)\}\right| + 1 \leq \left|\mathcal{I}_{t_1}^{t_2}(x)\right|$$

This will be proven using Lemma 3.4.2.i, since every time $r_t \notin \mathcal{I}_{t_1}^{t_2}(x)$, $\mathcal{I}_{t_1}^{t_2}(x)$ gets larger:

Proof. Let $A = \{t : t_1 \leq t \leq t_2, r_t \notin \mathcal{I}_{t_1}^{t_2}(x)\}$. Then, in particular, for any $t \in A$, we have $r_t \notin \mathcal{I}_{t+1}^{t_2}(x)$. Then, for all $t \in a$, applying Lemma 3.4.2.i, we have $\left|\mathcal{I}_t^{t_2}(x)\right| > \left|\mathcal{I}_{t+1}^{t_2}(x)\right|$. Let $A = \{t^1, \ldots, t^k\}$, with $t^i < t^{i+1}$ for any $1 \leq i < k$. Then:

$$\left|\mathcal{I}_{t_1}^{t_2}(x)\right| \geq \left|\mathcal{I}_{t^1}^{t_2}(x)\right| > \left|\mathcal{I}_{t^1+1}^{t_2}(x)\right| \geq \left|\mathcal{I}_{t^2}^{t_2}(x)\right| > \left|\mathcal{I}_{t^2+1}^{t_2}(x)\right| \geq \ldots > \left|\mathcal{I}_{t^k+1}^{t_2}(x)\right| \geq 1$$

Where the non-strict inequalities derive by Monotonicity(Lemma 3.3.5), and the last one from the fact that $t^k + 1 \leq t_2 + 1 \Rightarrow x \in \mathcal{I}_{t^k+1}^{t_2}(x)$. There are $k = |A|$ strict inequalities over integers, which concludes the proof. □

We now define the *rounds graph*, which will keep track of the information – the in-neighbors – the root of each round has:

Definition 3.4.4 (The rounds graph)**.** *We define the* rounds graph *as follows:*

The graph has $2n + \left\lceil \sqrt{2}n \right\rceil$ *nodes: one node representing each process, and one node for each of the first* $\left\lceil (1+\sqrt{2})n \right\rceil$ *rounds.*

And it has the directed edges: one edge from a process p *to a round* t *if* $p \in \mathcal{I}_1^{t-1}(r_t)$*, and one edge from a round* $t < \left\lceil \sqrt{2}n \right\rceil$ *to a round* $t' > t$ *if* $r_t \in \mathcal{I}_1^{t'-1}(r_{t'})$*.*

We will now show that there is at least a node of out-degree n in that graph, which will translate into a process that has broadcast its piece of information to everyone:

Lemma 3.4.5. *In the rounds graph, there is at least a node of out-degree* n*.*

Proof. Let us look at round t. If $t \leq \left\lceil \sqrt{2}n \right\rceil$, then t has in-degree at least t. Indeed, it has in-degree:

$$\begin{aligned}
&\left|\{t' : 1 \leq t' \leq t-1, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| + \left|\mathcal{I}_1^{t-1}(x)\right| \\
&\quad \geq 1 + \left|\{t' : 1 \leq t' \leq t-1, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| + \left|\{t' : 1 \leq t' \leq t-1, r_{t'} \notin \mathcal{I}_1^{t-1}(r_t)\}\right| = t
\end{aligned}$$

where we used Lemma 3.4.3 for the inequality. Similarly, if $t > \left\lceil \sqrt{2}n \right\rceil$, then t has in-degree at least $\left\lceil \sqrt{2}n \right\rceil$. Indeed, it has in-degree:

$$\begin{aligned}
&\left|\{t' : 1 \leq t' < \left\lceil \sqrt{2}n \right\rceil, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| + \left|\mathcal{I}_1^{t-1}(r_t)\right| \\
&\quad \geq 1 + \left|\{t' : 1 \leq t' < \left\lceil \sqrt{2}n \right\rceil, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| + \left|\{t' : 1 \leq t' \leq t-1, r_{t'} \notin \mathcal{I}_1^{t-1}(r_t)\}\right| \\
&\quad \geq 1 + \left|\{t' : 1 \leq t' < \left\lceil \sqrt{2}n \right\rceil, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| + \left|\{t' : 1 \leq t' < \left\lceil \sqrt{2}n \right\rceil, r_{t'} \notin \mathcal{I}_1^{t-1}(r_t)\}\right| \\
&\quad = \left\lceil \sqrt{2}n \right\rceil
\end{aligned}$$

where we used Lemma 3.4.3 for the first inequality.

Summing the in-degrees over all the rounds, we get that the number of edges $|E|$ is at least:

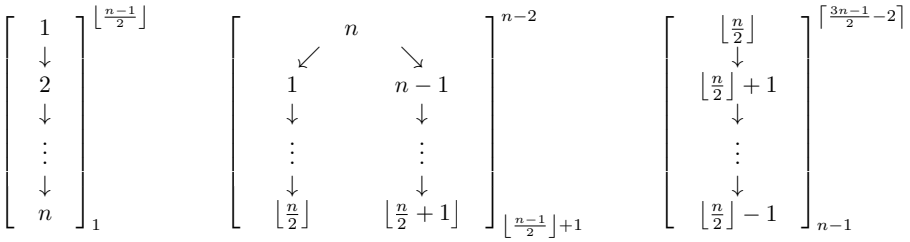


Figure 3.3: The lower bound example for the broadcasting on trees problem [129]. $[G]_a^b$ means that network G is the communication graph for all rounds between a and b (inclusive).

$$|E| \geq \sum_{t=1}^{\lceil \sqrt{2}n \rceil} t + n \times \left\lceil \sqrt{2}n \right\rceil = \frac{\left\lceil \sqrt{2}n \right\rceil \left(\left\lceil \sqrt{2}n \right\rceil + 1 \right)}{2} + n \left\lceil \sqrt{2}n \right\rceil > \left\lceil (1+\sqrt{2})n \right\rceil n$$

But only the n nodes representing the processes and the nodes representing the first $\left\lceil \sqrt{2}n \right\rceil - 1$ rounds have out edges. The pigeonhole principle asserts then that one of those nodes has an out-degree of at least n. □

Theorem 3.4.6. $t^*(\mathcal{T}_n) \leq \left\lceil (1+\sqrt{2})n \right\rceil$

Proof. Assume it is not the case, that is, for every process $x \in [n]$, for every round $t \leq \left\lceil (1+\sqrt{2})n \right\rceil$, $|\mathcal{O}_1^t(x)| < n$. We know that in the rounds graph, there is a node y of degree at least n. Define z as follows: if y represents a process, let z be that process. If y represents a round, let z be the root of that round. We will show that z must have broadcast before $\left\lceil (1+\sqrt{2})n \right\rceil$ rounds.

Let $t_1 < \ldots < t_n$ be the rounds y has out-edges to. By definition, this means that $z \in \mathcal{I}_1^{t_i - 1}(r_{t_i}) \Leftrightarrow r_{t_i} \in \mathcal{O}_1^{t_i - 1}(z)$ for every $i \in [n]$. By Lemma 3.4.2ii, we thus have, for every $i \in [n]$, that $\left|\mathcal{O}_1^{t_i}(z)\right| > \left|\mathcal{O}_1^{t_i - 1}(z)\right|$. Then, using Monotonicity (Lemma 3.3.5) for non-strict inequalities:

$$\left|\mathcal{O}_1^{t_n}(z)\right| > \left|\mathcal{O}_1^{t_n - 1}(z)\right| \geq \left|\mathcal{O}_1^{t_{n-1}}(z)\right| > \left|\mathcal{O}_1^{t_{n-1} - 1}(z)\right| \geq \ldots \geq \left|\mathcal{O}_1^{t_1}(z)\right| > \left|\mathcal{O}_1^{t_1 - 1}(z)\right| \geq 0$$

We have n strict inequalities over non-negative integers, the largest one must be at least n, which is a contradiction. □

3.4.2 The Lower Bound

A lower bound for this problem has been given by Zeiner, Schwarz, and Schmid [129]:

Theorem 3.4.7. $t^*(\mathcal{T}_n) \geq \left\lceil \frac{3n-1}{2} - 2 \right\rceil$

A figure of that lower bound can be found in Figure 3.3.

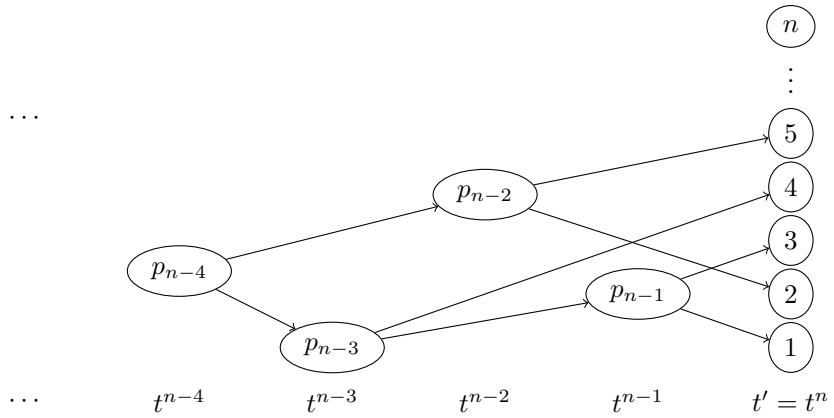


Figure 3.4: Main idea of the proof for the upper bound on covering on k-forests. An edge from process x in column t_1 to process y in column t_2 means that $x \in \mathcal{I}_{t_1}^{t_2-1}(y)$ (unless $t_2 = t'$ where we don't require the offset). We start with a cover of size n at t' then go back in time, finding a smaller cover at every step. After the first step, we don't need to cover 1 and 3 anymore, because if we reach p_{n-1} before round $t^{(n-1)}$, we also reach 1 and 3 before round t', by going through p_{n-1}.

3.5 Covering on k-Forests

In this section, we study an adversary that has to choose a communication network in each round that is a union of k trees. In this setting, we cannot ensure broadcast, so we look at the time when there exists a cover of size k: k processes such that any other process has heard of at least one of them.

3.5.1 The Upper Bound

Even though the problem is very similar to broadcasting on trees, the proofs of Section 3.4 do not translate in a straightforward way into an upper bound for covering on k-forests. We thus have a completely different proof for this problem.

The intuition of our approach, depicted in Figure 3.4, is as follows: We will start with a cover of size n at some time $t' = t^{(n)}$ that is large enough, and then go back in time until we find a process that can reach two other processes, say, x and y, of that cover. Calling this process p_{n-1} and the corresponding time $t^{(n-1)}$, we thus have $p_{n-1} \in \mathcal{I}_{t^{(n-1)}}^{t^{(n)}}(x)$ and $p_{n-1} \in \mathcal{I}_{t^{(n-1)}}^{t^{(n)}}(y)$. When repeating the process, we then remove x and y from our set of processes to cover, add p_{n-1}, and start over, until the cover has size k. We need to be careful to guarantee that rounds do not overlap.

Indeed, to remove p_{n-1} from our set of processes to cover, we have to reach it before round $t^{(n-1)}$, otherwise we will not be guaranteed to reach x or y at time $t^{(n)}$. Thus, we will store with each process x of the cover the corresponding round t such that x has to be reached by round t by the process that replaces x in the cover. More specifically, we model the cover by a series of sets $(A_u)_{k \leq u \leq n}$, where each A_u is a collection of u pairs (p, t), where p is a process, and t is a round. To compute A_{u-1} from A_u, we have to find a process that can reach two processes p_1, p_2 by rounds t_1, t_2, such that $(p_1, t_1), (p_2, t_2) \in A_u$ and then we replace these two pairs by a new pair, creating the cover A_{u-1}.

In this section we first state the definitions and results of this section, before giving the full proof to each of our claims. We first define what it means for a set A of pairs (p,t) to be a cover.

Definition 3.5.1 (Cover between rounds)**.** *A set $A = \{(a_1,t_1),\dots,(a_s,t_s)\}$ of s (process, round) pairs is a* cover of a set B of processes for round $t \geq \max_s\{t_s\}$ *if for every $b \in B$, there exists an $i \in [s]$ such that $b \in \mathcal{O}^t_{t_i+1}(a_i)$.*

We next couple the cover property of set A with *strictness*, which indicates that we did not (yet) go back enough in time to find a process that reaches two different processes in A.

Definition 3.5.2 (Strict cover)**.** *A set $A = \{(a_1,t_1),\dots,(a_s,t_s)\}$ of s (process, round) pairs is* strict at round t *if there exists no process $p \in [n]$ and $i,j \in [s], i \neq j$ such that $p \in \mathcal{I}^{t_i}_t(a_i)$ and $p \in \mathcal{I}^{t_j}_t(a_j)$.*

As we consider earlier and earlier rounds, the sets $\mathcal{I}^{t_i}_t(a_i)$ will get larger and larger, and A will lose its strictness. Thus, we then define the following sequence of covers of $[n]$, and analyze their strictness over time. We carefully choose our set A_s so that it has cardinality s. This means that A_k has cardinality k, which is our goal.

Definition 3.5.3 (Sequence of strict covers)**.** *Let t' be a large enough round. For every $k \leq s \leq n$, we define a sequence of strict sets A_s and rounds $t^{(s)}$ as follows:*

Define $A_n = \{(i,t') : i \in [n]\}$, $t^{(n)} = t'$.

Define $t^{(s)} = \max_{j\in\mathbb{N}}\{A_{s+1}$ is not strict at round $j\}$. As A_{s+1} is not strict at round $t^{(s)}$, there exist $i,j \in [s+1]$ and a process p_s such that $p_s \in \mathcal{I}^{t_i}_{t^{(s)}}(a_i) \cap \mathcal{I}^{t_j}_{t^{(s)}}(a_j)$. If $(p_s, t^{(s)}-1) \in A_{s+1}$, we define $A_s = A_{s+1}\backslash\{(a_i,t_i)\}$, else we define $A_s = (A_{s+1}\backslash\{(a_i,t_i),(a_j,t_j)\}) \cup \{(p_s,t^{(s)}-1)\}$.

Define $\Delta_s = t^{(s)} - t^{(s-1)}$ for $k+1 \leq s \leq n$.

Recall that our goal is to upper bound t', which can be done if we upper bound $\sum_{s=k+1}^{n} \Delta_s$. The strictness of a set is a key notion as a strict set has the following very useful property:

Lemma 3.5.4 (Strict increments)**.** *Let $s \geq k+1$ and let $A = \{(a_1,t_1),\dots,(a_s,t_s)\}$ be a strict set of size s at round t. Let $I \subseteq \{i \in [s] : t \leq t_i+1\}$. Then there exists a set of indices $J \subseteq I$, $|J| \geq |I| - k$, such that for every $i \in J$, $\left|\mathcal{I}^{t_i}_{t-1}(a_i)\right| > \left|\mathcal{I}^{t_i}_t(a_i)\right|$.*

Proof. Consider a root r of round $t-1$. As A is strict at round t is follows that $r \in \mathcal{I}^{t_i}_t(a_i)$ for at most one a_i with $i \in I$. As there are at most k roots in round $t-1$, it follows that there are at least $|I|-k$ values $i \in I$ such that none of the roots of round $t-1$ is in $\mathcal{I}^{t_i}_t(a_i)$. Let J be the set of all such values of i. Let us denote for any $p \in [n]$ the root of the tree that contains p in round t by the value $r_t(p)$. It follows that in particular $r_{t-1}(a_i) \notin \mathcal{I}^{t_i}_t(a_i)$. Since $t \leq t_i+1$, we have that $a_i \in \mathcal{I}^{t_i}_t(a_i)$. Now for each such i the path from $r_{t-1}(a_i)$ to a_i in G_{t-1} must contain an edge (x,y) such that $x \notin \mathcal{I}^{t_i}_t(a_i)$, and $y \in \mathcal{I}^{t_i}_t(a_i)$. By Transitivity (Lemma 3.3.4), it holds that $(\mathcal{I}^{t_i}_t(a_i) \cup \{x\}) \subseteq \mathcal{I}^{t_i}_{t-1}(a_i)$, which implies that $\left|\mathcal{I}^{t_i}_{t-1}(a_i)\right| > \left|\mathcal{I}^{t_i}_t(a_i)\right|$. □

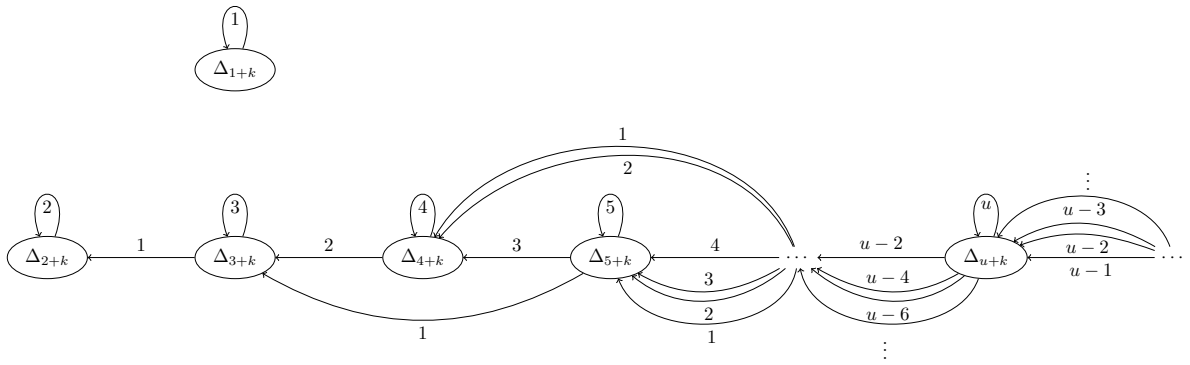


Figure 3.5: The strict rounds graph. Vertex u's weight represents $\Delta_u = t^{(u)} - t^{(u-1)}$. The weight of an edge (u, s) represents how much a round $t \in [t^{(u-1)} + 1, t^{(u)}]$ contributes at least to S_s. Because of the strictness of A_s, the sum of the weights of the in-neighbors (multiplied by the edges' weight) of a node s, which is smaller than S_s, should not exceed n.

It is not hard to show that *all* a_i of A_s fulfill $t^{(s)} \leq t_i + 1$. The following lemma helps us find more sets that satisfy the conditions of the Strict Increments Lemma. This essentially follows from the fact that, by construction, at least $2s - u$ elements from A_u are shared with A_s.

Lemma 3.5.5. *Let $s, u \in \{k, \ldots, n\}$ such that $s \leq u$. Let $A_s = \{(a_1, t_1), \ldots, (a_s, t_s)\}$. Let $I = \{i \in [s] : t^{(u)} \leq t_i + 1\}$. Then it holds that $|I| \geq 2s - u$.*

We now define the strict rounds graph, which we use to analyze the values of Δ_s as s varies from n to k. A depiction of that graph can be seen in Figure 3.5.

Definition 3.5.6 (Strict rounds graph)**.** *The* strict rounds graph (V, E) *consists of $n - k$ vertices, labeled from $k + 1$ to n, where each vertex s has weight Δ_s. There exists a directed edge from vertex u to vertex s if $s \leq u \leq \min\{2s - k - 1, n\}$, and its weight is $w(u, s) = 2s - k - u$.*

The next lemma is the crucial lemma: To bound t' we first bound the following "weighted volumes". If we define α_s to be the weighted out-degree of a node s, and the *volume* of s to be α_s multiplied by its own weight, then in the strict rounds graph, the cumulative sum over the volumes of the first j vertices is at most $j \cdot n$.

Lemma 3.5.7. *Let $u \in \{k + 1, \ldots, n\}$. Then $\sum_{s \leq u} \Delta_s \alpha_s \leq (u - k) \cdot n$.*

We briefly sketch the proof of this lemma. We first prove that for every $k+1 \leq w \leq u$ the sum $\sigma_w := \sum_{v=w}^{\min\{2w-k-1,n\}} (2w - k - v)\, \Delta_v$ is at most n as follows. Since for every $k+1 \leq w \leq u$ the set A_w is strict at all rounds $t \geq t^{(w-1)} + 1$, it follows that $S_w := \sum_{(a_i,t_i) \in A_w} \left| \mathcal{I}^{t_i}_{t^{(w-1)}+1}(a_i) \right|$ is at most n. We then lower bound S_w by $\sum_{v=w}^{\min\{2w-k-1,n\}} (2w - k - v)\, \Delta_v$ in two steps: First, Lemma 3.5.5 allows us to find a set I of cardinality larger than $2w - v$ for each round in the interval $[t^{(v-1)} + 1, t^{(v)}]$ that fits the Strict Increments Lemma's conditions. We then use the Strict Increments Lemma to show that S_w increases by $2w - v - k$ in each of those rounds, which results in a lower bound of σ_w for S_w, and thus the upper bound $\sigma_w \leq n$. Summing this inequality over all $u - k$ nodes w with $w \leq u$ gives $\sum_{w \leq u} \sigma_w \leq (u - k) \cdot n$.

Next note that σ_w is the weighted in-degree of node w in the strict rounds graph where each edge is weighted by the product of its edge weight and the weight of its tail. By definition, the tail of every (directed) edge has a higher label than its head and, thus, every outgoing edge of a node $s \leq u$ is also an incoming edge of a node $w \leq u$. This allows us to argue that $\sum_{w \leq u} \sigma_w$ is at least $\sum_{s \leq u} \Delta_s \alpha_s$, which leads to the final result.

We use Lemma 3.5.7 to bound t' as follows. We first show that any vertex weight distribution on the strict rounds graph following the volume bound must fulfill the following property.

Lemma 3.5.8. *Let $\{\delta_s\}_{k+1 \leq s \leq n}$ with $\delta_s \in \mathbb{R}$ be a vertex weight distribution over the strict rounds graph such that for every $u \in \{k+1, \dots n\}$, $\sum_{s \leq u} \delta_s \alpha_s \leq (u-k) \cdot n$. Then $\sum_{s=k+1}^{n} \delta_s \leq \frac{\pi^2+6}{6} n$.*

Lemma 3.5.7 and Lemma 3.5.8 give the desired bound on $\sum_{s=k+1}^{n} \Delta_s$, which in turn bounds t'.

Corollary 3.5.9. $\sum_{s=k+1}^{n} \Delta_s \leq \frac{\pi^2+6}{6} n$

Theorem 3.5.10. $t_k^c(\mathcal{F}_n^k) \leq \frac{\pi^2+6}{6} n + 1$

The rest of this section is dedicated to proving in detail all of those claims, as well as introducing all concepts, stating and proving any intermediate lemmata that would be necessary. First we analyze in detail the sets defined in Definition 3.5.3, making sure they have the desired cardinality and proving that they form a cover of $[n]$ from round 1 to t'. Note that it directly follows from Definition 3.5.3 that the size of A_s is s and we will in the following always use (a_i, t_i) for $1 \leq i \leq s$ to denote the elements of the set A_s.

Lemma 3.5.11. *For every $s \in \{k, \dots, n\}$, $|A_s| = s$.*

Proof. It is true by reverse induction. Indeed, it is trivially true for $s = n$. Suppose it is true for some $s \in \{k+1, \dots, n\}$ and consider two cases: If $(p_{s-1}, t^{(s-1)} - 1) \in A_s$, then $|A_s| - |A_{s-1}| = 1$, as only (a_i, t_i) is removed from A_s. If $(p_{s-1}, t^{(s-1)} - 1) \notin A_s$, then both (a_i, t_i) and (a_j, t_j) are removed from A_s and $(p_{s-1}, t^{(s-1)} - 1)$ is added, which implies that $|A_s| - |A_{s-1}| = 1$. Thus, in both cases $|A_s| - |A_{s-1}| = 1$. As by induction $|A_s| = s$, it follows that $|A_{s-1}| = s - 1$. □

In Definition 3.5.3, we offset $t^{(s)}$ by one when introducing p_s in A_s, which guarantees that A_s is a cover of $[n]$ for round t', as shown below.

Lemma 3.5.12. *For every $s \in \{k, \dots, n\}$, we have that A_s is a cover of $[n]$ for round t'.*

Proof. We show the claim by reverse induction. It holds trivially for $s = n$. Assume now that A_{s+1} is a cover of $[n]$ for round t'. We will show that this implies that A_s is also a cover of $[n]$ for round t'. By the fact that A_{s+1} is a cover, it follows for each $x \in [n]$ that there exists a $h \in [s+1]$ such that $x \in \mathcal{O}^{t'}_{t_h+1}(a_h)$. If $(a_h, t_h) \in A_s$, then there is nothing to do. If however $(a_h, t_h) \notin A_s$, then $p_s \in \mathcal{I}^{t_h}_{t^{(s)}}(a_h)$, and by Transitivity (Lemma 3.3.4) it follows that $x \in \mathcal{O}^{t'}_{t^{(s)}}(p_s)$. □

Now that we are assured that the cover property will hold throughout, we give the intuition of what follows. As defined above, we have $\Delta_u = t^{(u)} - t^{(u-1)}$ for $k < u \leq n$. We will then look at A_s for some $s \in \{k, \dots, n\}$. Since A_s is strict at round $t^{(s-1)} + 1$, it follows that $\left(\mathcal{I}^{t_i}_{t^{(s-1)}+1}(a_i)\right)_{i\in[s]}$ are pairwise disjoint subsets of $[n]$, which implies that $S_s = \sum_{(a_i,t_i)\in A_s} \left|\mathcal{I}^{t_i}_{t^{(s-1)}+1}(a_i)\right|$ is at most n. We will find a lower bound for that value that depends on the values of Δ_u.

To do so, we will use the Strict Increments Lemma (Lemma 3.5.4). Indeed we will first prove that for every $i \in [s]$, it holds that $t^{(s)} \leq t_i + 1$. This will allow us to use the Strict Increments Lemma (Lemma 3.5.4) between rounds $t^{(s-1)} + 2$ and $t^{(s)}$ and so each of those rounds contributes an additive $s - k$ to the lower bound of S_s. Of course, such a contribution only happens if $t^{(s-1)} + 2 \leq t^{(s)}$.

We will further prove that there exist $s-1$ elements $i \in [s]$ such that $t^{(s+1)} \leq t_i + 1$. This will follow from the fact that $|A_s \cap A_{s+1}| \geq s - 1$, and that for every $(a, t) \in A_{s+1}$, we have that $t^{(s+1)} \leq t + 1$. This allows us to use the Strict Increments Lemma (Lemma 3.5.4) between rounds $t^{(s)}+1$ and $t^{(s+1)}$, so that each of those rounds contributes at least an additive $s-1-k$ to the lower bounds of S_s *in addition to the contribution of rounds* $[t^{(s-1)} + 2, t^{(s)}]$.

In fact, we will generalize this analysis to all values of $u \geq s$ with $A_u \cap A_s \neq \varnothing$, and for each u every round in $[t^{(u-1)} + 1, t^{(u)}]$ contributes at least $2s - k - u$ to the lower bound of S_s, leading to a contribution at at least $(2s - k - u)\Delta_u$ for u.

Lemma 3.5.13. *A_s is strict at round $t^{(s)} + 1$, for any $s \in \{k, \dots, n\}$.*

Proof. Let $s \in \{k, \dots, n\}$. A_{s+1} is strict at round $t^{(s)} + 1$. Assume by contradiction that A_s is not strict at round $t^{(s)} + 1$, that is, there exists a $x \in [n]$ and $i, j \in [s]$ such that $x \in \mathcal{I}^{t_i}_{t^{(s)}+1}(a_i)$ and $x \in \mathcal{I}^{t_j}_{t^{(s)}+1}(a_j)$. Since $\mathcal{I}^{t^{(s)}-1}_{t^{(s)}+1}(p_s) = \varnothing$ because $t^{(s)} + 1 > (t^{(s)} - 1) + 1$, we have that $x \notin \mathcal{I}^{t^{(s)}-1}_{t^{(s)}+1}(p_s)$. This implies that $(a_i, t_i), (a_j, t_j) \in A_{s+1}$, which contradicts the strictness of A_{s+1}. This concludes the proof. □

Corollary 3.5.14. $t^{(s)} \leq t^{(s+1)} \quad \forall s \in \{k, \dots, n-1\}$

Corollary 3.5.15. *For every $s \in \{k, \dots, n\}$, we have that for every $i \in [s]$, $t^{(s)} \leq t_i + 1$.*

Proof. By reverse induction, it is true for $s = n$, as for every $i \in [n], t_i = t^{(n)}$. Assume it is true for some $s + 1$ for $s \in \{k, \dots n - 1\}$, and let us look at A_s. Then by Corollary 3.5.14 and the induction hypothesis, for every $i \in [s]$ we have that either $(a_i, t_i) \in A_{s+1}$ and thus $t^{(s-1)} \leq t^{(s)} \leq t_i + 1$, or that $(a_i, t_i) = (p_s, t^{(s)} - 1)$ and trivially the inequality holds. □

Lemma 3.5.16. *Let $s, u \in \{k, \dots, n\}$ such that $s \leq u$. Then $A_s \cap A_u \geq 2s - u$.*

Proof. By reverse induction, it is trivially true for $s = u$. Assume we have $|A_{s+1} \cap A_u| \geq 2(s + 1) - u$ for some s such that $k \leq s \leq u - 1$.

We have that, by distribution of the intersection over the union operator:

$$A_{s+1} \cap A_u = ((A_{s+1} \cap A_s) \cup (A_{s+1} \setminus A_s)) \cap A_u = (A_{s+1} \cap A_s \cap A_u) \cup ((A_{s+1} \setminus A_s) \cap A_u)$$

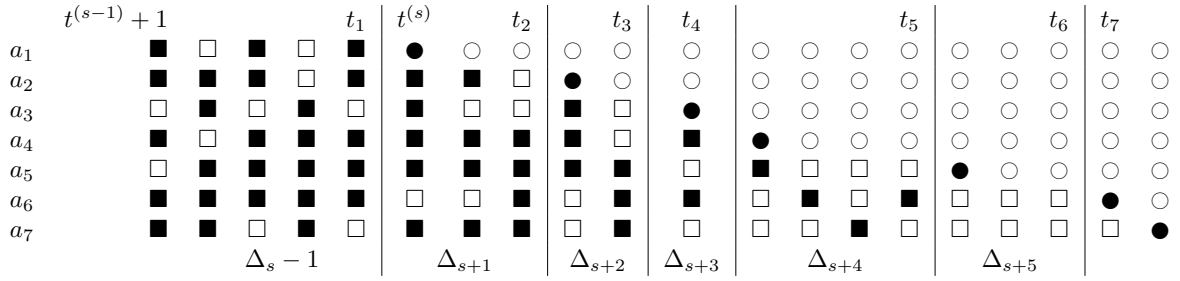


Figure 3.6: Illustration of proof of Lemma 3.5.17, with $s = 7$ and $k = 2$. Each column represents a round, each row represents an element (a_i, t_i) of A_s, ordered according to t_i. Remember that every t_i directly precedes a $t^{(u)}$. We add a square if $t \leq t_i$, and a circle otherwise. The entry is black if $\left|\mathcal{I}_t^{t_i}(a_i)\right| > \left|\mathcal{I}_{t+1}^{t_i}(a_i)\right|$ and white otherwise. We have one black circle per row. Indeed, we have that $\left|\mathcal{I}_{t_i+1}^{t_i}(a_i)\right| = |\{a_i\}| = 1 > 0 = \left|\mathcal{I}_{t_i+2}^{t_i}(a_i)\right|$. Lemma 3.5.5 gives a lower bound on the number of squares for each column. The Strict Increments Lemma asserts that there are at most k white squares per column. A lower bound for S_s is thus the number of black entries.

Hence:

$$|A_s \cap A_u| \geq |A_{s+1} \cap A_s \cap A_u| \geq |A_{s+1} \cap A_u| - |(A_{s+1} \setminus A_s) \cap A_u| \geq 2(s+1) - u - 2 = 2s - u$$

Where we used that $|(A_{s+1} \setminus A_s) \cap A_u| \leq |A_{s+1} \setminus A_s| \leq 2$, because we removed at most two elements from A_{s+1} to get A_s. □

Lemma 3.5.5. *Let $s, u \in \{k, \ldots, n\}$ such that $s \leq u$. Let $A_s = \{(a_1, t_1), \ldots, (a_s, t_s)\}$. Let $I = \{i \in [s] : t^{(u)} \leq t_i + 1\}$. Then it holds that $|I| \geq 2s - u$.*

Proof. It is a direct implication of Corollary 3.5.15 and Lemma 3.5.16.

Indeed, $|I| = \left|\{(a,t) \in A_s : t^{(u)} \leq t + 1\}\right|$, and we have $\{(a,t) \in A_s : t^{(u)} \leq t+1\} \supseteq \{(a,t) \in A_s \cap A_u : t^{(u)} \leq t+1\} = \{(a,t) \in A_s \cap A_u\} = A_s \cap A_u$, where the penultimate equality holds by Corollary 3.5.15 applied to u. Lemma 3.5.16 states that $|A_s \cap A_u| \geq 2s - u$. □

Lemma 3.5.17. *Let $\Delta_s = t^{(s)} - t^{(s-1)}$. Then, for every $s \in \{k+1, \ldots, n\}$:*

$$\sum_{u=s}^{\min\{2s-k-1,n\}} (2s - k - u)\,\Delta_u \leq n$$

An illustration of the proof can be found in Figure 3.6.

Proof. Let $s \in \{k+1, \ldots, n\}$. If for every u such that $s \leq u \leq \min\{2s-k-1, n\}$, we have $\Delta_u = 0$, then the result is immediate. If not, let $x = \min\{u : s \leq u \leq \min\{2s-k-1, n\} \wedge \Delta_u > 0\}$. In the rest of the proof, let u be restricted to fulfill $x \leq u \leq \min\{2s-k-1, n\}$ and let $I_s^u = \{i \in [s] : t^{(u)} \leq t_i + 1\}$. By Lemma 3.5.5, we have that $|I_s^u| \geq 2s - u$. To prove the claim we will give upper and lower bounds for $S_s = \sum_{i \in [s]} \left|\mathcal{I}_{t^{(s-1)}+1}^{t_i}(a_i)\right|$. The upper

bound directly follows from the fact that $t^{(s-1)}+1$ is the smallest round at which A_s is strict, which implies that $S_s \leq n$.

We next fix a round t with $t^{(u-1)}+1 \leq t \leq t^{(u)}$. We have that $t^{(s-1)}$ is the largest round A_s is not strict at (by Definition 3.5.3), so it is strict at $t \geq t^{(u-1)}+1 > t^{(s-1)}$. By the Strict Increments Lemma (Lemma 3.5.4), there exists a set $J_s^u(t) \subseteq I_s^u$ with $|J_s^u(t)| \geq |I_s^u| - k \geq 2s-u-k$ such that for every $j \in J_s^u(t)$, $\left|\mathcal{I}_{t-1}^{t_j}(a_j)\right| > \left|\mathcal{I}_t^{t_j}(a_j)\right|$. As every $j \in J_s^u(t)$ belongs to I_s^u it follows that $t \leq t^{(u)} \leq t_j+1$. Thus, for every $(j,t) \in [s] \times \mathbb{N}$ with $t > t_j+1$ it holds that it does not belong to $J_s^u(t)$, or, equivalently, $\mathbb{1}(j \in J_s^u(t)) = 0$.

We can then write:

$$S_s := \sum_{i\in[s]} \left|\mathcal{I}_{t^{(s-1)}+1}^{t_i}(a_i)\right| = \sum_{i\in[s]} \left(\left|\mathcal{I}_{t_i+1}^{t_i}(a_i)\right| + \sum_{t=t^{(s-1)}+1}^{t_i} \left(\left|\mathcal{I}_t^{t_i}(a_i)\right| - \left|\mathcal{I}_{t+1}^{t_i}(a_i)\right| \right) \right)$$

We first separate the second sum according to which interval $[t^{(u-1)}+1, t^{(u)}]$ round t belongs to, and furthermore add a third sum that is equal to 0, since $\mathbb{1}(j \in J_s^u(t)) = 0$ for every $(j,t) \in [s] \times \mathbb{N}$ such that $t > t_j+1$. For the consecutive inequality note that by the definition of $J_s^u(t)$, we have that in the second sum $\left|\mathcal{I}_t^{t_i}(a_i)\right| - \left|\mathcal{I}_{t+1}^{t_i}(a_i)\right| \geq \mathbb{1}(i \in J_s^u(t+1))$.

$$\begin{aligned}
S_s &\geq \sum_{i\in[s]} \left(\left|\mathcal{I}_{t_i+1}^{t_i}(a_i)\right| + \sum_{t=t^{(s-1)}+1}^{t_i} \sum_{u=s}^{\min\{2s-k-1,n\}} \mathbb{1}(t^{(u-1)}+1 \leq t+1 \leq t^{(u)}) \left(\left|\mathcal{I}_t^{t_i}(a_i)\right| - \left|\mathcal{I}_{t+1}^{t_i}(a_i)\right| \right) \right. \\
&\qquad \left. + \sum_{t=t_i+2}^{t^{(\min\{2s-k-1,n\})}} \sum_{u=s}^{\min\{2s-k-1,n\}} \mathbb{1}(t^{(u-1)}+1 \leq t \leq t^{(u)}) \mathbb{1}(i \in J_s^u(t)) \right) \\
&\geq \sum_{i\in[s]} \left(\left|\mathcal{I}_{t_i+1}^{t_i}(a_i)\right| + \sum_{t=t^{(s-1)}+1}^{t_i} \sum_{u=s}^{\min\{2s-k-1,n\}} \mathbb{1}(t^{(u-1)}+1 \leq t+1 \leq t^{(u)}) \mathbb{1}(i \in J_s^u(t+1)) \right. \\
&\qquad \left. + \sum_{t=t_i+2}^{t^{(\min\{2s-k-1,n\})}} \sum_{u=s}^{\min\{2s-k-1,n\}} \mathbb{1}(t^{(u-1)}+1 \leq t \leq t^{(u)}) \mathbb{1}(i \in J_s^u(t)) \right) \\
&= \sum_{i\in[s]} \left(\left|\mathcal{I}_{t_i+1}^{t_i}(a_i)\right| + \sum_{t=t^{(s-1)}+2}^{t_i+1} \sum_{u=s}^{\min\{2s-k-1,n\}} \mathbb{1}(t^{(u-1)}+1 \leq t \leq t^{(u)}) \mathbb{1}(i \in J_s^u(t)) \right. \\
&\qquad \left. + \sum_{t=t_i+2}^{t^{(\min\{2s-k-1,n\})}} \sum_{u=s}^{\min\{2s-k-1,n\}} \mathbb{1}(t^{(u-1)}+1 \leq t \leq t^{(u)}) \mathbb{1}(i \in J_s^u(t)) \right)
\end{aligned}$$

$$\begin{aligned}
S_s \geq \sum_{i\in[s]} \left(\left|\mathcal{I}_{t_i+1}^{t_i}(a_i)\right| \right. \\
\left. + \sum_{t=t^{(s-1)}+2}^{\max\{t_i+1, t^{(\min\{2s-k-1,n\})}\}} \sum_{u=s}^{\min\{2s-k-1,n\}} \mathbb{1}(t^{(u-1)}+1 \leq t \leq t^{(u)}) \mathbb{1}(i \in J_s^u(t)) \right) =: X_1
\end{aligned}$$

We next invert the two sum symbols, and delete terms where $\mathbb{1}(t^{(u-1)}+1 \leq t \leq t^{(u)}) = 0$:

$$X_1 \geq \sum_{i\in[s]} \left(1 + \sum_{t=t^{(x-1)}+2}^{t^{(x)}} \mathbb{1}(i \in J_s^x(t)) + \sum_{u=x+1}^{\min\{2s-k-1,n\}} \sum_{t=t^{(u-1)}+1}^{t^{(u)}} \mathbb{1}(i \in J_s^u(t))\right)$$
$$\geq s + \sum_{t=t^{(x-1)}+2}^{t^{(x)}} \sum_{i\in[s]} \mathbb{1}(i \in J_s^x(t)) + \sum_{u=x+1}^{\min\{2s-k-1,n\}} \sum_{t=t^{(u-1)}+1}^{t^{(u)}} \sum_{i\in[s]} \mathbb{1}(i \in J_s^u(t)) =: X_2$$

Using that $\sum_{i\in[s]} \mathbb{1}(i \in J_s^u(t)) = |J_s^u(t)| \geq 2s-k-u$ and $\Delta_u = t^{(u)} - t^{(u-1)}$:

$$X_2 \geq s + (\Delta_x - 1)(2s-k-x) + \sum_{u=x+1}^{\min\{2s-k-1,n\}} (2s-k-u)\Delta_u \geq \sum_{u=s}^{\min\{2s-k-1,n\}} (2s-k-u)\Delta_u,$$

where the last inequality follows since $x \geq s$ which implies that $2s-k-x \leq 2s-k-s \leq s$ and that $\Delta_u = 0$ for $u < x$.

As $\sum_{i\in[s]} \left|\mathcal{I}_{t^{(s-1)}+1}^{t_i}(a_i)\right| = S_s \leq n$, the claim follows. □

We now recall the definition of the strict rounds graph, which is built to harness the previous result.

Definition 3.5.6 (Strict rounds graph)**.** *The* strict rounds graph (V, E) *consists of* $n-k$ *vertices, labeled from* $k+1$ *to* n*, where each vertex* s *has weight* Δ_s*. There exists a directed edge from vertex* u *to vertex* s *if* $s \leq u \leq \min\{2s-k-1, n\}$*, and its weight is* $w(u,s) = 2s-k-u$*.*

The graph is represented in Figure 3.5. Each node s with $k+1 \leq s \leq (n+k)/2$ has $s-k+2$ incoming edges, namely (s,s), $(s+1,s)$, $\ldots$, $(2s-k-1,s)$, with weights $s-k, s-k-1, \ldots, 1$, respectively. Each node s with $(n+k)/2 < s \leq n$ has $n+1-s$ incoming edges, namely (s,s), $(s+1,s)$, $\ldots$, (n,s), with weights $s-k, s-k-1, \ldots, 2s-k-n$, respectively.

Each node s has $s+1-\left\lfloor \frac{s+k}{2} \right\rfloor$ outgoing edges, namely $(s,s), (s,s-1), \ldots, (s, \left\lfloor \frac{s+k}{2} \right\rfloor + 1)$ of weight $s-k, s-k-2, \ldots,$ respectively. If $s-k$ is even, all outgoing edges of s have even weight and the edge $(s, \left\lfloor \frac{s+k}{2} \right\rfloor + 1)$ has weight 2. If $s-k$ is odd, all outgoing edges of s have odd weight and the edge $(s, \left\lfloor \frac{s+k}{2} \right\rfloor + 1)$ has weight 1.

Lemma 3.5.18. *Let* $u \in [n]$*, and define* $E_u^{out} := \{(s,v) \in E : s \leq u\}$*. In the strict rounds graph, we have that*

$$\sum_{(s,v)\in E_u^{out}} \Delta_s w(s,v) \leq (u-k)\cdot n$$

Proof. We are summing over all the out-edges of all the vertices with a label at most u. Let $E_u^{in} = \{(s,v) \in E : v \leq u\}$. Every outgoing edge of a node $s \leq u$ must end at a node v with $v \leq s \leq u$. Thus, it is contained in the set of incoming edge of all nodes v with $v \leq u$. Said differently, $E_u^{out} \subseteq E_u^{in}$.

Since all the terms are positive, we have:

$$\sum_{(s,v)\in E_u^{out}} \Delta_s w(s,v) \leq \sum_{(s,v)\in E_u^{in}} \Delta_s w(s,v) = \sum_{v\leq u} \sum_{s:(s,v)\in E} \Delta_s w(s,v)$$
$$\leq \sum_{v\leq u} \sum_{s:(s,v)\in E} \Delta_s(2v-k-s) \leq \sum_{v\leq u} \sum_{s=v}^{\min\{n,2v-s-1\}} \Delta_s(2v-k-s)$$

Applying Lemma 3.5.17, the claim follows. □

Lemma 3.5.19. *Let $s \geq k+1$. If $s-k$ is odd, we have $\sum_{v:(s,v)\in E} w(s,v) = (\frac{s-k+1}{2})^2$. If it is even, we have $\sum_{v:(s,v)\in E} w(s,v) = (\frac{s-k}{2})^2 + \frac{s-k}{2}$.*

Proof. If $s-k$ is odd, let ℓ be such that $s-k = 2\ell+1$:

$$\sum_{v:(s,v)\in E} w(s,v) = \sum_{v:(s,v)\in E} 2v-k-s = \sum_{v:v\leq s\leq 2v-k} 2v-k-s = \sum_{(s+k)/2\leq v\leq s} 2v-k-s$$

but then $s-k \equiv s-k+2k \mod 2$, so $s+k$ is also odd, and $\left\lceil \frac{s+k}{2} \right\rceil = \ell+k+1$, therefore:

$$\sum_{v:(s,v)\in E} w(s,v) = \sum_{\ell+k+1\leq v\leq 2\ell+k+1} 2v-k-2\ell-k-1 = \sum_{0\leq v\leq \ell} 2v+1$$
$$= 2\frac{\ell(\ell+1)}{2} + \ell + 1 = \frac{s-k-1}{2}\frac{s-k+1}{2} + \frac{s-k+1}{2}$$

If $s-k$ is even, let ℓ be such that $s-k = 2\ell$:

$$\sum_{v:(s,v)\in E} w(s,v) = \sum_{v:(s,v)\in E} 2v-k-s = \sum_{v:v\leq s\leq 2v-k} 2v-k-s = \sum_{(s+k)/2\leq v\leq s} 2v-k-s$$

Therefore:

$$\sum_{v:(s,v)\in E} w(s,v) = \sum_{\ell+k\leq v\leq 2\ell+k} 2v-k-2\ell-k = \sum_{0\leq v\leq \ell} 2v = 2\frac{\ell(\ell+1)}{2} = \frac{s-k}{2}\frac{s-k+2}{2}$$

□

Basically, as discussed above and seen in Figure 3.5, if $s-k$ is odd, we have that $\sum_{v:(s,v)\in E} w(s,v) = 1+3+5+\cdots+s-k = (\frac{s-k+1}{2})^2$, while if it is even, $\sum_{v:(s,v)\in E} w(s,v) = 2+4+6+\cdots+s-k = (\frac{s-k}{2})^2 + \frac{s-k}{2}$.

Definition 3.5.20 (Cover constants)**.** *We define the following numbers: $\beta = \frac{\pi^2+6}{6}, \alpha_s = (\frac{s-k+1}{2})^2$ if $s-k$ is odd, $\alpha_s = (\frac{s-k}{2})^2 + \frac{s-k}{2}$ if $s-k$ is even.*

Note that α_s is an integer for all values of $s \geq k$. By applying this definition to the bounds of Lemmata 3.5.18 and 3.5.19 we achieve the following result.

Lemma 3.5.7. *Let $u \in \{k+1, \ldots, n\}$. Then $\sum_{s \leq u} \Delta_s \alpha_s \leq (u-k) \cdot n$.*

Lemma 3.5.8. *Let $\{\delta_s\}_{k+1 \leq s \leq n}$ with $\delta_s \in \mathbb{R}$ be a vertex weight distribution over the strict rounds graph such that for every $u \in \{k+1, \ldots n\}$, $\sum_{s \leq u} \delta_s \alpha_s \leq (u-k) \cdot n$. Then $\sum_{s=k+1}^{n} \delta_s \leq \frac{\pi^2+6}{6} n$.*

Proof. We will show that $\sum_{s=k+1}^{n} \delta_s \alpha_s$ is maximized when $\delta_s = \alpha_s^{-1} \cdot n$ for every s. Let $\{\delta_s\}_{k+1 \leq s \leq n}$ be such that $\sum_{s=k+1}^{n} \delta_s \alpha_s$ if maximized. Assume by contradiction that there exists a $k+1 \leq s \leq n$ such that $\delta_s < \alpha_s^{-1} \cdot n$. Let v be the smallest such s and set $\epsilon = n - \delta_v \alpha_v > 0$. We can then build a different solution γ by setting $\gamma_s = \delta_s$ for every $s \notin \{v, v+1\}$, and setting $\gamma_v = \delta_v + \epsilon \alpha_v^{-1} = \alpha_v^{-1} \cdot n > \delta_v$ and $\gamma_{v+1} = \delta_{v+1} - \epsilon \alpha_{v+1}^{-1}$. Note that for every $u \in \{k+1, \ldots n\}$, $\sum_{s \leq u} \gamma_s \alpha_s \leq (u-k) \cdot n$ as

(1) for $u < v$ it holds that $\sum_{s \leq u} \gamma_s \alpha_s = (u-k) \cdot n$,

(2) for $u = v$ it holds that $\sum_{s \leq u} \gamma_s \alpha_s = (\epsilon \alpha_v^{-1}) \alpha_v + \delta_v \alpha_v + \sum_{s<u} \delta_s \alpha_s = (n - \delta_v \alpha_v) + \delta_v \alpha_v + \sum_{s<u} \delta_s \alpha_s = n + (u-1-k) \cdot n = (u-k) \cdot n$, and

(3) for $u \geq v+1$ if holds that $\sum_{s \leq u} \gamma_s \alpha_s = \sum_{s \leq v-1} \delta_s \alpha_s + \gamma_v \alpha_v + \gamma_{v+1} \alpha_{v+1} + \sum_{v+2 \leq s \leq u} \delta_s \alpha_s = \sum_{s \leq v-1} \delta_s \alpha_s + \delta_v \alpha_v + \epsilon \alpha_{v+1} \alpha_{v+1}^{-1} + \delta_u \alpha_u^{-1} - \epsilon \alpha_u \alpha_u^{-1} + \sum_{v+2 \leq s \leq u} \delta_s \alpha_s = \sum_{s \leq u} \delta_s \alpha_s \leq (u-k) \cdot n$, and

Now note that $\sum_{s=k+1}^{n} \gamma_s \alpha_s > \sum_{s=k+1}^{n} \delta_s \alpha_s$ as the α_s values are decreasing in s, which gives a contradiction to the assumption that δ maximizes $\sum_{s=k+1}^{n} \delta_s \alpha_s$. It now follows that

$$\sum_{s=k+1}^{n} \delta_s \leq \sum_{s=k+1}^{n} \alpha_s^{-1} n \leq \sum_{s-k=1}^{\infty} \alpha_s^{-1} n \leq n \sum_{\ell \in \mathbb{N}} \frac{1}{\ell^2} + n \sum_{\ell \in \mathbb{N}} \frac{1}{\ell^2 + \ell} = \beta n.$$

where the last equation follows by the fact that $\sum_{\ell \in \mathbb{N}} \frac{1}{\ell^2} = \pi^2/6$ and $\sum_{\ell \in \mathbb{N}} \frac{1}{\ell^2+\ell} = \sum_{\ell \in \mathbb{N}} \frac{1}{\ell} - \frac{1}{\ell+1} = 1$. □

Corollary 3.5.9. $\sum_{s=k+1}^{n} \Delta_s \leq \frac{\pi^2+6}{6} n$

Theorem 3.5.10. $t_k^c(\mathcal{F}_n^k) \leq \frac{\pi^2+6}{6} n + 1$

Proof. We have $\sum_{s=k+1}^{n} \Delta_s = \sum_{s=k+1}^{n} t^{(s)} - t^{(s+1)} = t' - t^{(k)}$, which is at most $\frac{\pi^2+6}{6} n$ by Corollary 3.5.9. Setting $t^{(k)} = 1$, we get that $t' \leq \frac{\pi^2+6}{6} n + 1$. Lemma 3.5.12 ensures that A_k is a cover of $[n]$ for t', which means that for every $x \in [n]$, there exists a $i \in [k]$ such that $x \in \mathcal{O}_{t_i+1}^{t'}(a_i)$. Since $\min_i\{t_i\} = t^{(k)} - 1 = 0$, this means that every $x \in [n]$ belongs to $\mathcal{O}_1^{t'}(a_i)$, which implies that $t_k^c(\mathcal{F}_n^k) \leq t'$. □

3.5.2 The Lower Bound

Our lower bound for this problem is very similar to the lower bound in Figure 3.3. The lower bound specific for this problem can be found in Figure 3.7. This lower bound is essentially "trapping" $k-1$ vertices each in its own 1-vertex tree, and repeating the strategy for the lower

bound with i vertices for broadcasting on trees on the last tree. Since none of the first $k-1$ vertices can communicate with the others, cover is only achieved when broadcast is achieved on the last tree. Setting $i = n - k + 1$, we get a lower bound for the Covering on k-forests of $\left\lceil \frac{3n-3k}{2} - 1 \right\rceil$:

Theorem 3.5.21. $t_k^c(\mathcal{T}_n^k) \geq \left\lceil \frac{3n-3k}{2} - 1 \right\rceil$

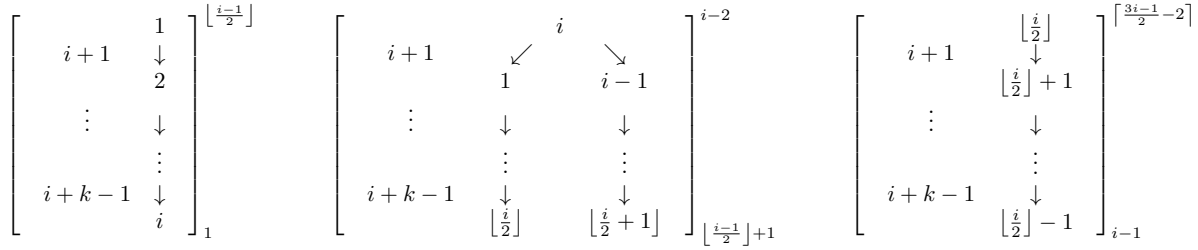


Figure 3.7: The lower bound example for the Covering on k-Forests problem. $[G]_a^b$ means that network G is the communication graph for all rounds between a and b (inclusive). Each of the vertices $i+1, \ldots, i+k-1$ is isolated in every round.

3.6 k-Broadcasting on k-rooted Networks

In this section, we study the case where the adversary has to choose a communication network that has k roots at least in each round. This allows us to enforce not only broadcast, but rather k-broadcast.

The problem is very similar to the one studied in the Section 3.4, and indeed a slight modification of the proofs there work for this problem.

3.6.1 The Upper Bound

By using a technique very similar to Section 3.4, we will show that whenever we have a set $A \subset [n]$ of size at most $k-1$, we can find a process $p \notin A$ that has broadcast, as long as we have waited for a large enough number of rounds.

Definition 3.6.1 (Set of roots of a round)**.** *Let t be a round. We denote by R_t the set of the roots of G_t.*

Recall that $|R_t| \geq k$.

The following two lemmata, very similar to Section 3.4, can be proven by looking at paths going from a root to a carefully chosen node.

Lemma 3.6.2. *Let $t \leq t'$ be rounds, and let x and r be processes such that, $r \in R_t$ and $r \notin \mathcal{I}_{t+1}^{t'}(x)$. Then $\left|\mathcal{I}_t^{t'}(x)\right| > \left|\mathcal{I}_{t+1}^{t'}(x)\right|$.*

Proof. We will show that there exists a process $y \in \mathcal{I}_{t+1}^{t'}(x)$ that has an in-neighbor z in G_t such that $z \notin \mathcal{I}_{t+1}^{t'}(x)$. Then, by Transitivity (Lemma 3.3.4), $z \in \mathcal{I}_t^{t'}(x)$. By Monotonicity (Lemma 3.3.5), we have that $\mathcal{I}_{t+1}^{t'}(x) \subset \mathcal{I}_t^{t'}(x)$, this will show that $\left|\mathcal{I}_t^{t'}(x)\right| > \left|\mathcal{I}_{t+1}^{t'}(x)\right|$.

Let us now find such a z. Consider the path from r to x in G_t. Since $r \notin \mathcal{I}_{t+1}^{t'}(x)$, and trivially $x \in \mathcal{I}_{t+1}^{t'}(x)$, this path must include an edge (z, y) such that $z \notin \mathcal{I}_{t+1}^{t'}(x)$, $y \in \mathcal{I}_{t+1}^{t'}(x)$. □

Lemma 3.6.3. *Let $t \leq t'$ be rounds, and let x and r be processes such that $r \in R_{t'}$, and $r \in \mathcal{O}_t^{t'-1}(x)$. Then $\left|\mathcal{O}_t^{t'}(x)\right| > \left|\mathcal{O}_t^{t'-1}(x)\right|$, unless $\left|\mathcal{O}_t^{t'-1}(x)\right| = n$.*

Proof. Let us look at the case $\left|\mathcal{O}_t^{t'-1}(x)\right| < n$. We will show that there exists a process $y \in \mathcal{O}_t^{t'-1}(x)$ that has an out-neighbor z in $G_{t'}$ such that $z \notin \mathcal{O}_t^{t'-1}(x)$. Then, by Transitivity (Lemma 3.3.4), $z \in \mathcal{O}_t^{t'}(x)$. By Monotonicity, we obviously have $\mathcal{O}_t^{t'-1}(x) \subset \mathcal{O}_t^{t'}(x)$, which implies that $\left|\mathcal{O}_t^{t'}(x)\right| > \left|\mathcal{O}_t^{t'-1}(x)\right|$.

Let us now find such a z. Since $\left|\mathcal{O}_t^{t'-1}(x)\right| < n$, there exists a process a such that $a \notin \mathcal{O}_t^{t'-1}(x)$. Consider the path from r to a in $G_{t'}$. Since $r \in \mathcal{O}_t^{t'-1}(x)$, and $a \notin \mathcal{O}_t^{t'-1}(x)$, this path must include an edge (y, z) such that $z \notin \mathcal{O}_t^{t'-1}(x)$, $y \in \mathcal{O}_t^{t'-1}(x)$. □

We now give a lemma that links the number of in-neighbors of a node to the number of previous rounds whose root are in-neighbors of said node:

Lemma 3.6.4. *Let x be a process, let $t_1 \leq t_2$ be rounds, and let r_t denote a root of R_t for every $t \in \{t_1, \ldots, t_2\}$, then it holds that*

$$\left|\{t : t_1 \leq t \leq t_2, r_t \notin \mathcal{I}_{t_1}^{t_2}(x)\}\right| + 1 \leq \left|\mathcal{I}_{t_1}^{t_2}(x)\right|$$

Proof. Let $A = \{t : t_1 \leq t \leq t_2, r_t \notin \mathcal{I}_{t_1}^{t_2}(x)\}$. Then, in particular, for any $t \in A$, we have $r_t \notin \mathcal{I}_{t+1}^{t_2}(x)$. Then, for all $t \in a$, applying Lemma 3.6.2, we have $\left|\mathcal{I}_t^{t_2}(x)\right| > \left|\mathcal{I}_{t+1}^{t_2}(x)\right|$. Let $k = |A|$ and let $A = \{t^{(1)}, \ldots, t^{(k)}\}$, with $t^{(i)} < t^{(i+1)}$ for $1 \leq i < k$. Then by Monotonicity (Lemma 3.3.5) it follows that

$$\left|\mathcal{I}_{t_1}^{t_2}(x)\right| \geq \left|\mathcal{I}_{t^{(1)}}^{t_2}(x)\right| > \left|\mathcal{I}_{t^{(1)}+1}^{t_2}(x)\right| \geq \left|\mathcal{I}_{t^{(2)}}^{t_2}(x)\right| > \left|\mathcal{I}_{t^{(2)}+1}^{t_2}(x)\right| \geq \ldots > \left|\mathcal{I}_{t^{(k)}+1}^{t_2}(x)\right| \geq 1$$

There are $k = |A|$ inequalities over integers, which concludes the proof. □

We now define the rounds graph avoiding some set A, which is the equivalent of the rounds graph of Section 3.4, this time being very careful not to choose any process from A being used as a root for a round.

Definition 3.6.5 (Rounds graph avoiding a set)**.** *Let A be an arbitrary set of processes with $|A| < k$. We define the* rounds graph avoiding set A *as follows: It contains $2n + \left\lceil\sqrt{2}n\right\rceil + |A|$ nodes, namely*

1. *one* process node *representing each process.*
2. *one* round node *for each of the first $\left\lceil(1+\sqrt{2})n\right\rceil + |A|$ rounds.*

For each round t, let r_t be a vertex of $R_t \setminus A$. The graph contains

1. *a directed edge to every round node t from each process node p with $p \in \mathcal{I}_1^{t-1}(r_t)$.*
2. *a directed edge from every round node $t < \left\lceil \sqrt{2}n \right\rceil + |A|$ to a round node $t' > t$ if $r_t \in \mathcal{I}_1^{t'-1}(r_{t'})$.*

We now find a node in the rounds graph that has out-degree n, that is not a node from A. This node will represent a process that has broadcast.

Lemma 3.6.6. *For any set A of processes with $|A| < k$, the rounds graph avoiding set A contains at least one node of out-degree n that is not a node representing a process of A.*

Proof. As in the proof of Lemma 3.4.5 we apply the pigeonhole principle on the edges, however, this time we will ignore out-edges from nodes representing processes from A. Let us look at round t. If $t \leq \left\lceil \sqrt{2}n \right\rceil + |A|$, then the round node t has in-degree at least t. Indeed, it has in-degree:

$$\begin{aligned}
&\left|\{t' : 1 \leq t' \leq t-1, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| + \left|\mathcal{I}_1^{t-1}(r_t)\right| \\
&\quad \geq 1 + \left|\{t' : 1 \leq t' \leq t-1, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| + \left|\{t' : 1 \leq t' \leq t-1, r_{t'} \notin \mathcal{I}_1^{t-1}(r_t)\}\right| = t
\end{aligned}$$

where we used Lemma 3.6.4 for the inequality. Therefore, it has at least $t-|A|$ in-neighbors not in A. Similarly, if $t > \left\lceil \sqrt{2}n \right\rceil + |A|$, then the round node t has in-degree at least $\left\lceil \sqrt{2}n \right\rceil + |A|$. Indeed, it has in-degree:

$$\begin{aligned}
&\left|\{t' : 1 \leq t' < \left\lceil \sqrt{2}n \right\rceil + |A|, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| + \left|\mathcal{I}_1^{t-1}(r_t)\right| \\
&\geq 1 + \left|\{t' : 1 \leq t' < \left\lceil \sqrt{2}n \right\rceil + |A|, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| + \left|\{t' : 1 \leq t' \leq t-1, r_{t'} \notin \mathcal{I}_1^{t-1}(r_t)\}\right| \\
&\geq 1 + \left|\{t' : 1 \leq t' < \left\lceil \sqrt{2}n \right\rceil + |A|, r_{t'} \in \mathcal{I}_1^{t-1}(r_t)\}\right| \\
&\quad + \left|\{t' : 1 \leq t' < \left\lceil \sqrt{2}n \right\rceil + |A|, r_{t'} \notin \mathcal{I}_1^{t-1}(r_t)\}\right| = \left\lceil \sqrt{2}n \right\rceil + |A|
\end{aligned}$$

where we used Lemma 3.6.4 for the first inequality. Therefore, it has at least $\left\lceil \sqrt{2}n \right\rceil$ in-neighbors not elements of A.

Summing the in-degrees over all the rounds, we get that the number of edges $|E_{\overline{A}}|$ with no endpoint in A is at least:

$$|E_{\overline{A}}| \geq \sum_{t=|A|}^{\left\lceil \sqrt{2}n \right\rceil + |A|} (t - |A|) + n \times \left\lceil \sqrt{2}n \right\rceil = \frac{\left\lceil \sqrt{2}n \right\rceil \left(\left\lceil \sqrt{2}n \right\rceil + 1 \right)}{2} + n \left\lceil \sqrt{2}n \right\rceil > \left\lceil (1+\sqrt{2})n \right\rceil n$$

But only the $n - |A|$ nodes representing the processes and the nodes representing the first $\left\lceil \sqrt{2}n \right\rceil - 1 + |A|$ rounds have out-edges among those counted. The pigeonhole principle asserts then that one of those nodes has an out-degree of at least n. □

Lemma 3.6.7. *For any set A of processes with $|A| < k$, there exists a process $z \notin A$ that has broadcast its identifier to everyone after $\left\lceil (1+\sqrt{2})n \right\rceil + |A|$ rounds.*

Proof. Build the rounds graph avoiding A. We know that in that graph, there is a node y of degree at least n, that is not a process node representing A. Define z as follows: (1) If y represents a process, let z be that process. It follows that $z \notin A$. (2) If y represents a round t, let z be the root r_t, which is not in A by definition of the graph. We will show in either case that z must have broadcast before $\left\lceil (1+\sqrt{2})n \right\rceil + |A|$ rounds.

Let $t_1 < \ldots < t_n$ be the rounds y has out-edges to. By definition, and in both cases (1) and (2), this means $z \in \mathcal{I}_1^{t_i-1}(r_{t_i})$ which is equivalent to $r_{t_i} \in \mathcal{O}_1^{t_i-1}(z)$ for every $i \in [n]$. By Lemma 3.6.3, we thus have, for every $i \in [n]$, that $\left|\mathcal{O}_1^{t_i}(z)\right| > \left|\mathcal{O}_1^{t_i-1}(z)\right|$. Then:

$$\left|\mathcal{O}_1^{t_n}(z)\right| > \left|\mathcal{O}_1^{t_n-1}(z)\right| \geq \left|\mathcal{O}_1^{t_{n-1}}(z)\right| > \left|\mathcal{O}_1^{t_{n-1}-1}(z)\right| \geq \ldots \geq \left|\mathcal{O}_1^{t_1}(z)\right| > \left|\mathcal{O}_1^{t_1-1}(z)\right| \geq 0$$

We have n strict inequalities over non-negative integers, the largest one must be at least n, which implies that z has broadcast. □

This naturally leads to an upper bound for the k-broadcasting on k-rooted networks:

Theorem 3.6.8. $t_k^*(\mathcal{R}_n^k) \leq \left\lceil (1+\sqrt{2})n \right\rceil + k - 1$

Proof. By contradiction, assume that the set A of elements that have broadcast after $\left\lceil (1+\sqrt{2})n \right\rceil + k - 1$ rounds is smaller than k and apply Lemma 3.6.7 to find a process not in A that has broadcast in time less than $\left\lceil (1+\sqrt{2})n \right\rceil + |A|$ rounds, contradicting the assumption. □

3.6.2 The Lower Bound

We build a lower bound example based on the lower bound for broadcasting on trees. A figure and analysis of that example can be found in Figure 3.3, which yields the following result:

Theorem 3.6.9. $t_k^*(\mathcal{R}_n^k) \geq \left\lceil \frac{3n-9k}{2} + 2 \right\rceil$

Proof. The graph for this lower bound can be found in Figure 3.8. The idea is to reduce the k-broadcasting problem to the broadcast problem. More specifically, we use the sequence of networks from the lower bound of Figure 3.3 with i vertices for broadcasting on trees, while replacing each of the 3 vertices that act as root in the three networks of Figure 3.3 by k fully connected vertices, i.e. k vertices such that everyone points to the other $k-1$. Then k-cover is only achieved when broadcast is achieved on the original networks. Setting $i = n - 3k + 3$, we get a lower bound for the k-broadcasting on k-rooted networks of $\left\lceil \frac{3n-9k}{2} + 2 \right\rceil$. □

3.7 Related Work

Broadcasting, gossiping, and other information dissemination problems have been studied by the distributed computing community for decades already [81]. Most classic literature on network broadcast considers a static setting, e.g., where in each round each node can send

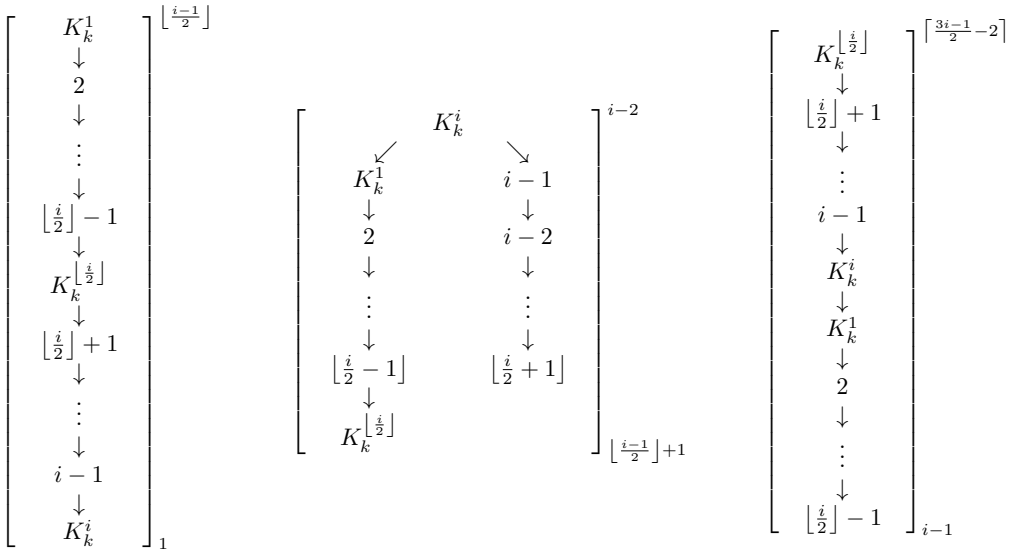


Figure 3.8: The lower bound example for the Covering on k-Forests problem. $[G]_a^b$ means that network G is the communication graph for all rounds between a and b (inclusive). K_k^a is the complete graph that replaces vertex a from the example is Figure 3.3. $K_k^a \to y$ (respectively $y \to K_k^a$) means that an edge is inserted from every $x \in K_k^a$ to y (respectively from y to every $x \in K_k^a$). Similarly, $K_k^a \to K_k^b$ means that an edge is inserted from every $x \in K_k^a$ to every $y \in K_k^b$.

information to one neighbor [87]. This model has also been explored in the context of gossiping, e.g., by Fraigniaud and Lazard [67]. Kuhn, Lynch and Oshman [98] explore the all-to-all data dissemination problem (gossiping) in an undirected dynamic network, where processes do not know beforehand the total number of processes and must decide on that number. Ahmadi, Kuhn, Kutten, Molla and Pandurangan [3] study the message complexity of broadcast in an undirected dynamic setting, where the adversary pays up a cost for changing the network. Broadcast has also been studied in dynamic communication networks which evolve randomly, e.g., by Clementi et al. [41], and in the radio network model [63], just to give a few examples.

A closely related yet different problem to broadcasting is the consensus problem. Our model builds up on the heard-of model first introduced by Charron-Bost and Schiper [34], where authors prove results for the solvability of consensus also considering oblivious message adversaries. Among other results, they give a $\log n$ upper bound for nonsplit graphs, which are graphs for which every pair of nodes has a common in-neighbor. This would result in an $n \log n$ upper bound for rooted trees when combining it with the result of Charron-Bost, Függer and Nowak [33]. Függer, Nowak, and Winkler [68] prove that the time complexity of broadcast is a lower bound for consensus time. A general characterization of oblivious message adversaries on which consensus is solvable, based on broadcastability, has been presented by Coulouma, Godard and Peters in [42]. A time complexity analysis has further been studied by Winkler, Rincon Galeana, Paz, Schmid, and Schmid [127]. Another similar problem is agreement, considered by Santoro and Widmayer [117], where only a k-majority should agree on a value, as opposed to everyone for consensus. Afek, Gafni, Rajsbaum, Raynal and Travers [2] studied a generalization to consensus that is k-set consensus, where each node has to decide on a value such that all the nodes together do not decide on more than k different values.

In this paper, we have studied the broadcasting problem on directed dynamic networks, with an adversary that can choose the communication network at each round among rooted trees. Zeiner, Schwarz, and Schmid [129] give a $n \log n$ upper bound to our exact problem by using graph-theoretic reasoning. They also give a $\left\lceil \frac{3n-1}{2} \right\rceil - 2$ lower bound by providing an explicit example. They further show that under an adversary that can only choose rooted trees with a fixed number of leaves or internal nodes, broadcast time is linear.

There has also been interest in a problem variant which only differs in the pool of networks the adversary can choose a network from for each communication round. Függer, Nowak, and Winkler [68] give an $O(\log \log n)$ upper bound if the adversary can only choose nonsplit graphs. Combined with the result of Charron-Bost, Függer, and Nowak [33] that states that one can simulate $n - 1$ rounds of rooted trees with a round of a nonsplit graph, this gives the previous $O(n \log \log n)$ upper bound for broadcasting on trees. Dobrev and Vrto [47, 46] give specific results when the adversary is restricted to hypercubic and tori graphs with some missing edges.

Bibliographic note: an announcement of this work has been presented at ACM PODC 2022 [61].

3.8 Conclusion

In this paper, we considered an innovative version of the classic broadcast problem where processes communicate across a dynamically changing network, as it often arises in practice (e.g., due to interference). Like in the static setting, the broadcast problem on dynamic networks is related to consensus and leader election: broadcast is a prerequisite for consensus, and hence, the time complexity of broadcast is a lower bound for the consensus and leader election complexity.

Our main contribution is a proof that the broadcast time is at most linear in this setting, which is asymptotically optimal. We further presented several natural generalizations of our model and result.

Our work opens several avenues for future research. In particular, it will be interesting to study the broadcast time also in non-adversarial environments where graphs evolve according to a random process (e.g., due to random node movements). It will also be interesting to further explore the implications of our methods on the closely related consensus problem.

References

[2] Yehuda Afek, Eli Gafni, Sergio Rajsbaum, Michel Raynal, and Corentin Travers. "The *k*-simultaneous consensus problem". In: *Distributed Comput.* 22.3 (2010), pp. 185–195. DOI: 10.1007/s00446-009-0090-8. URL: https://doi.org/10.1007/s00446-009-0090-8.

[3] Mohamad Ahmadi, Fabian Kuhn, Shay Kutten, Anisur Rahaman Molla, and Gopal Pandurangan. "The Communication Cost of Information Spreading in Dynamic Networks". In: *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*. IEEE, 2019, pp. 368–378. DOI: 10.1109/ICDCS.2019.00044. URL: https://doi.org/10.1109/ICDCS.2019.00044.

[33] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. "Approximate Consensus in Highly Dynamic Networks: The Role of Averaging Algorithms". In: *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*. Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Vol. 9135. Lecture Notes in Computer Science. Springer, 2015, pp. 528–539. DOI: 10.1007/978-3-662-47666-6_42. URL: https://doi.org/10.1007/978-3-662-47666-6%5C_42.

[34] Bernadette Charron-Bost and André Schiper. "The Heard-Of model: computing in distributed systems with benign faults". In: *Distributed Comput.* 22.1 (2009), pp. 49–71. DOI: 10.1007/s00446-009-0084-6. URL: https://doi.org/10.1007/s00446-009-0084-6.

[41] Andrea E. F. Clementi, Pierluigi Crescenzi, Carola Doerr, Pierre Fraigniaud, Francesco Pasquale, and Riccardo Silvestri. "Rumor spreading in random evolving graphs". In: *Random Struct. Algorithms* 48.2 (2016), pp. 290–312. DOI: 10.1002/rsa.20586. URL: https://doi.org/10.1002/rsa.20586.

[42] Étienne Coulouma, Emmanuel Godard, and Joseph G. Peters. "A characterization of oblivious message adversaries for which Consensus is solvable". In: *Theor. Comput. Sci.* 584 (2015), pp. 80–90. DOI: 10.1016/j.tcs.2015.01.024. URL: https://doi.org/10.1016/j.tcs.2015.01.024.

[46] Stefan Dobrev and Imrich Vrto. "Optimal Broadcasting in Hypercubes with Dynamic Faults". In: *Inf. Process. Lett.* 71.2 (1999), pp. 81–85. DOI: 10.1016/S0020-0190(99)00093-9. URL: https://doi.org/10.1016/S0020-0190(99)00093-9.

[47] Stefan Dobrev and Imrich Vrto. "Optimal Broadcasting in Tori with Dynamic Faults". In: *Parallel Process. Lett.* 12.1 (2002), pp. 17–22. DOI: 10.1142/S0129626402000781. URL: https://doi.org/10.1142/S0129626402000781.

[60] Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Asymptotically Tight Bounds on the Time Complexity of Broadcast and Its Variants in Dynamic Networks". In: *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*. Ed. by Yael Tauman Kalai. Vol. 251. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 47:1–47:21. DOI: 10.4230/LIPICS.ITCS.2023.47. URL: https://doi.org/10.4230/LIPIcs.ITCS.2023.47.

[61] Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Brief Announcement: Broadcasting Time in Dynamic Rooted Trees is Linear". In: *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*. Ed. by Alessia Milani and Philipp Woelfel. ACM, 2022, pp. 54–56. DOI: 10.1145/3519270.3538460. URL: https://doi.org/10.1145/3519270.3538460.

[63] Faith Ellen, Barun Gorain, Avery Miller, and Andrzej Pelc. "Constant-Length Labeling Schemes for Deterministic Radio Broadcast". In: *ACM Trans. Parallel Comput.* 8.3 (2021), 14:1–14:17. DOI: 10.1145/3470633. URL: https://doi.org/10.1145/3470633.

[67] Pierre Fraigniaud and Emmanuel Lazard. "Methods and problems of communication in usual networks". In: *Discret. Appl. Math.* 53.1-3 (1994), pp. 79–133. DOI: 10.1016/0166-218X(94)90180-5. URL: https://doi.org/10.1016/0166-218X(94)90180-5.

[68] Matthias Függer, Thomas Nowak, and Kyrill Winkler. "On the radius of nonsplit graphs and information dissemination in dynamic networks". In: *Discret. Appl. Math.* 282 (2020), pp. 257–264. DOI: 10.1016/j.dam.2020.02.013. URL: https://doi.org/10.1016/j.dam.2020.02.013.

[81] Sandra Mitchell Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. "A survey of gossiping and broadcasting in communication networks". In: *Networks* 18.4 (1988), pp. 319–349. DOI: 10.1002/net.3230180406. URL: https://doi.org/10.1002/net.3230180406.

[87] Juraj Hromkovi, Ralf Klasing, Burkhard Monien, and Regine Peine. "Dissemination of information in interconnection networks (broadcasting & gossiping)". In: *Combinatorial network theory*. Springer, 1996, pp. 125–212.

[98] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. "Distributed computation in dynamic networks". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. Ed. by Leonard J. Schulman. ACM, 2010, pp. 513–522. DOI: 10.1145/1806689.1806760. URL: https://doi.org/10.1145/1806689.1806760.

[117] Nicola Santoro and Peter Widmayer. "Time is Not a Healer". In: *STACS 89, 6th Annual Symposium on Theoretical Aspects of Computer Science, Paderborn, FRG, February 16-18, 1989, Proceedings*. Ed. by Burkhard Monien and Robert Cori. Vol. 349. Lecture Notes in Computer Science. Springer, 1989, pp. 304–313. DOI: 10.1007/BFb0028994. URL: https://doi.org/10.1007/BFb0028994.

[127] Kyrill Winkler, Hugo Rincon Galeana, Ami Paz, Stefan Schmid, and Ulrich Schmid. "The Time Complexity of Consensus Under Oblivious Message Adversaries". In: *14th Innovations in Theoretical Computer Science (ITCS)*. 2023.

[129] Martin Zeiner, Manfred Schwarz, and Ulrich Schmid. "On linear-time data dissemination in dynamic rooted trees". In: *Discret. Appl. Math.* 255 (2019), pp. 307–319. DOI: 10.1016/j.dam.2018.08.015. URL: https://doi.org/10.1016/j.dam.2018.08.015.

CHAPTER 4

Broadcast and Consensus in Stochastic Dynamic Networks with Byzantine Nodes and Adversarial Edges

This chapter appeard in full in

Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Broadcast and Consensus in Stochastic Dynamic Networks with Byzantine Nodes and Adversarial Edges". In: *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*. Ed. by Dan Alistarh. Vol. 319. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 21:1–21:15. DOI: 10.4230/LIPICS.DISC.2024.21. URL: https://doi.org/10.4230/LIPIcs.DISC.2024.21

4.1 Abstract

Broadcast and Consensus are most fundamental tasks in distributed computing. These tasks are particularly challenging in dynamic networks where communication across the network links may be unreliable, e.g., due to mobility or failures. Over the last years, researchers have derived several impossibility results and high time complexity lower bounds for these tasks. Specifically for the setting where in each round of communication the adversary is allowed to choose one rooted tree along which the information is disseminated, there is a lower as well as an upper bound that is linear in the number n of nodes for Broadcast and for $n \geq 3$ the adversary can guarantee that Consensus never happens. This setting is called the *oblivious message adversary for rooted trees*. Also note that if the adversary is allowed to choose a graph that does not contain a rooted tree, then it can guarantee that Broadcast and Consensus will never happen.

However, such deterministic adversarial models may be overly pessimistic, as many processes in real-world settings are stochastic in nature rather than worst-case.

This paper studies Broadcast on *stochastic* dynamic networks and shows that the situation is very different to the deterministic case. In particular, we show that if information dissemination

occurs along random rooted trees and directed ErdsRényi graphs, Broadcast completes in $O(\log n)$ rounds of communication with high probability. The fundamental insight in our analysis is that key variables are mutually independent.

We then study two adversarial models, (a) one with Byzantine nodes and (b) one where an adversary controls the edges. (a) Our techniques without Byzantine nodes are general enough so that they can be extended to Byzantine nodes. (b) In the spirit of smoothed analysis, we introduce the notion of *randomized oblivious message adversary*, where in each round, an adversary picks $k \leq 2n/3$ edges to appear in the communication network, and then a graph (e.g. rooted tree or directed ErdsRényi graph) is chosen uniformly at random among the set of all such graphs that include these edges. We show that Broadcast completes in a finite number of rounds, which is, e.g., $O(k + \log n)$ rounds in rooted trees.

We then extend these results to All-to-All Broadcast, and Consensus, and give lower bounds that show that most of our upper bounds are tight.

4.2 Introduction

Broadcast and Consensus are two of most fundamental operations in distributed computing which, in large-scale systems, typically have to be performed over a *network*. These networks are likely to be dynamic and change over time due, e.g., to link failures, interference, or mobility. Understanding how information disseminates in such dynamic networks is hence important for developing and analyzing efficient distributed systems.

Over the last years, researchers have derived several important insights into information dissemination in dynamic networks. A natural and popular model assumes an *oblivious*[1] *message adversary* which controls the information flow between a set of n nodes, by dropping an arbitrary set of messages sent by some nodes in each round [42]. Specifically, the adversary is defined by a set of directed communication graphs, one per round, whose edges determine which node can successfully send a message to which other node in a given round. Based on this set of graphs, the oblivious message adversary chooses a sequence of graphs over time, one per round with repetitions allowed, in such a way that the time complexity of the information dissemination task at hand is maximized. This model is appealing because it is conceptually simple and still provides a highly dynamic network model: The set of allowed graphs can be arbitrary, and the nodes that can communicate with one another can vary greatly from one round to the next. It is, thus, well-suited for settings where significant transient message loss occurs, such as in wireless networks. As information dissemination is faster on dense networks, most literature studies oblivious message adversaries on sparse networks, in particular, on rooted trees [60, 129, 42, 69, 70]. In fact, it is easy to see that rooted trees are a minimal necessary requirement for a successful Broadcast and Consensus: if an adversary may choose a graph that does not contain a rooted tree, then it may forever prevent the dissemination of a piece of information.

Unfortunately, information dissemination can be slow in trees: Broadcast can take time linear in the number of nodes under the oblivious message adversary [60, 129], even for constant-height trees (as we show in Appendix 4.9.1); and Consensus can even take super-polynomial

[1] Note that the term oblivious here refers to the property that nodes are oblivious to who their neighbors are. However, our adversary is actually adaptive.

time until termination, if it completes at all [42, 69]. Although this is bad news, one may argue that while the deterministic adversary model is useful in malicious environments, in real-word applications, the dynamics of communication networks is often more stochastic in nature. Accordingly, the worst-case model considered in existing literature may be overly conservative.

This motivates us, in this paper, to study information dissemination, and in particular Broadcast and Consensus tasks, in a scenario where the communication network is stochastic. Initially, we study a purely stochastic scenario where in each round, the communication network is chosen uniformly at random among all rooted trees. We then study several fundamental extensions of this model where the adversary has some limited control. In a first extension, we consider the case where some nodes (up to $\frac{2n}{3}$) may be Byzantine, that is, they may deviate arbitrarily from the protocol (and stop forwarding messages, for example). In a second extension, in the spirit of smoothed analysis, we study a setting where an adversary has some limited control over the communication network; we call this adversary the *randomized oblivious message adversary*. More specifically, we study the setting where first a worst-case adversary chooses k directed edges in the dynamic n-node network for some fixed k with $0 \leq k < \frac{2n}{3} - 1$[2], and then a rooted tree is chosen uniformly at random among the set of all rooted trees that include these edges.

We show that Broadcast completes within time $O(\log n)$ with high probability. We then show that this result even holds with Byzantine nodes. Under our randomized oblivious message adversary, Broadcast completes in $O(k + \log n)$ time with high probability.

It is useful to put our model into perspective with the SI (Susceptible-Infectious) model in epidemics [54]: while in the SI model interactions occur on a network that equals a clique, our model revolves around trees which are chosen by an adversary. This tree structure renders the analytical understanding of the information dissemination process harder, due to the lack of independence between the edges in the network in a particular round. A key insight from our paper is that we can prove the independence of a key variable, namely the increase in the number of "informed" nodes, which is crucial for our analysis. Our proof further relies on stochastic dominance, which makes it robust to the specific adversarial objective, and applies to any adversary definition (e.g., whether it aims to maximize the minimum or the expected number of rounds until the process completes).

We then extend our study to adversaries which are not limited to trees. In particular, we are interested in how the time complexity of Broadcast and Consensus depends on the density of the network. To this end, we consider *directed ErdsRényi graphs*, a directed version of the classic and well-studied random graphs. This graph family is parameterized by the number of edges m and hence allows us to shed light on the impact of the density. Specifically in this model, in each round the network is formed by sampling m edges. We again study two extensions: in the first extension some nodes behave as Byzantine nodes, while in the second extension, up to $k \leq m$ edges are chosen by an adversary, and then the remaining edges are sampled. While results for this model can be found in some cases where m is chosen so that the graph is an expander w.h.p. in each round by using the results from Augustine et al [14], in the case where m is small, our results are novel.

[2]We can relax this condition to $k \leq (1 - \epsilon)n$ for a fixed parameter ϵ, which results in a multiplicative factor of $\frac{1}{\epsilon}$ in the running time.

We show that all our results extend to multiple other problems, namely All-to-All Broadcast, Byzantine Consensus and Reliable Broadcast.

4.2.1 Model

Let n be the number of nodes, and let each node have a unique identifier from $[n]$. Time proceeds in a sequence of rounds $t = 1, 2, \ldots$, such that in each round t the communication network is chosen according to one of the models defined below. In each round, every honest node sends a message to all of its out-neighbors before receiving one from its in-neighbor. There is no message size restriction. We will study the following models of communication:

Uniformly Random Trees. In the *Uniformly Random Trees* model, let $\mathcal{T}_n$ be the set of all directed rooted trees on n nodes (where all edges are pointed away from the root). In each round, the communication network is chosen uniformly at random among graphs in $\mathcal{T}_n$, independently from other rounds. All nodes are honest.

Uniformly Random Trees with Byzantine Nodes. In the *Uniformly Random Trees with Byzantine Nodes* model, in each round, the communication network is chosen uniformly at random among graphs in $\mathcal{T}_n$, independently from other rounds. We have $n - f$ honest nodes, and f nodes are Byzantine, that is, they might behave arbitrarily (and even coordinate to make the protocol fail). We assume access to cryptographic tools that allow nodes to sign and encrypt messages. We restrict $f \leq \frac{2n}{3} - 1$.

Uniformly Random Trees with Adversarial Edges. In the *Uniformly Random Trees with Adversarial Edges* model, in each round, the communication network is chosen as follows: A randomized oblivious message adversary chooses k directed edges, then a graph is chosen uniformly at random among all graphs in $\mathcal{T}_n$ that include those k edges, and the choise is independent from other rounds. All nodes are honest. We restrict $k \leq \frac{2n}{3} - 1$.

Directed ErdsRényi graphs. In the *directed ErdsRényi graphs* model, let $m \in [n^2]$. In each round, the communication network is chosen by uniformly sampling without replacement m edges out of the possible n^2 edges of the graph, independently from other rounds. All nodes are honest.

Directed ErdsRényi graphs with Byzantine Nodes. In the *directed ErdsRényi graphs with Byzantine nodes* model, let $m \in [n^2]$. In each round, the communication network is chosen by uniformly sampling without replacement m edges out of the possible n^2 edges of the graph, independently from other rounds. We have $n - k$ honest nodes, and k nodes are Byzantine, that is, they might behave arbitrarily (and even coordinate to make the protocol fail). We assume access to cryptographic tools that allow nodes to sign and encrypt messages. We restrict $k < \frac{2n}{3}$.

Directed ErdsRényi graphs with Adversarial Edges. In the *directed ErdsRényi graphs with Adversarial Edges* model, let $0 \leq k \leq m \leq n^2$. In each round, the communication network is chosen as follows: A randomized oblivious message adversary chooses k edges,

$m - k$ edges are sampled without replacement out of the remaining $n^2 - k$ edges. All nodes are honest. We restrict $k < \frac{3}{4}n^2$.

In those models, we will study the following problems:

Broadcast. For the *Broadcast*[3] problem, we start by giving a message to *one* (honest) node. Each honest node that received the message will replicate it as many times as needed, and start forwarding it to its neighbors[4]. Then Broadcast *completes* when the message has been forwarded to all other nodes.

All-to-All Broadcast. In the *All-to-All Broadcast* problem, we start by giving a distinct message to *each* node. Each honest node that received a message will replicate it as many times as needed, and start forwarding it as well. Then All-to-All Broadcast *completes* when each honest node receives a copy of every message. In each round, each honest node forwards all the messages it has received in previous rounds to all its out-neighbors.

Consensus. In the *Consensus* problem, we start by giving a value $v_p \in \{0, 1\}$ to each node p, and Consensus completes when each honest node decided on a value in $\{0, 1\}$. This should satisfy the following conditions:

- **Agreement:** No two honest nodes decide differently.
- **Termination:** Every honest node eventually decides.
- **Validity:** The value the honest nodes agree on should be one of the input values v_p.

4.2.2 Our Results

We study Broadcast in the above mentioned models, then apply those results to All-to-All broadcast and Consensus. We prove the following theorems:

Theorem 4.2.1. *For any $c \geq 1$ and $n \geq 5$, Broadcast on Uniformly Random Trees completes within $32 \cdot c \cdot \ln n$ rounds with probability $p > 1 - \frac{1}{n^c}$.*

We also show that these results are asymptotically tight. Indeed, we cannot hope for a similar probability for a number of rounds that is $o(\ln n)$:

Theorem 4.2.2. *If $n \geq 2$, then the probability that Broadcast (and All-to-All Broadcast) on Uniformly Random Trees fails to complete within $\log n$ rounds is at least $\frac{1}{4}$.*

We have similar results for all the combinations of model and problem, which we summarize in Table 4.1.

[3]The Broadcast problem can also be seen as computing the *dynamic eccentricity* of the source node. Other flavors of Broadcast have also been studied under the name *dynamic radius* [68].

[4]This is known as "flooding" or "rumor passing"

	Broadcast	All-to-All Broadcast	Consensus
Uniformly Random Trees (URT)	$O(c \cdot \log n), q \leq n^{-c}$ $\Omega(\log n)$	$O(c \cdot \log n), q \leq n^{1-c}$ $\Omega(\log n)$	$O(c \cdot \log n), q \leq n^{-c}$
URT with Byzantine Nodes	$O(c \cdot \log n), q \leq n^{-c}$ $\Omega(\log n)$	$O(c \cdot \log n), q \leq n^{1-c}$ $\Omega(\log n)$	$O(f \cdot c \cdot \log n), q \leq n^{-c}$
URT with Adversarial Edges	$O(c \cdot (\log n + k)), q \leq n^{-c}$ $\Omega(\log n + k)$	$O(c \cdot (\log n + k)), q \leq n^{1-c}$ $\Omega(\log n + k)$	$O(c \cdot (\log n + k)), q \leq n^{-c}$
Directed ErdsRényi graphs (DER)	$O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right), q \leq n^{-c} \log n$ $O\left(\frac{c \log n}{\log(1+\frac{m}{n})}\right)$ if $\frac{m}{n} \geq \ln n$ with $q \leq n^{-c} \log n$ $\Omega\left(\frac{\log n}{\log(1+m/n)}\right)$	$O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right), q \leq n^{1-c} \log n$ $O\left(\frac{c \log n}{\log(1+\frac{m}{n})}\right)$ if $\frac{m}{n} \geq \ln n$ with $q \leq n^{1-c} \log n$ $\Omega\left(\frac{\log n}{\log(1+m/n)}\right)$	$O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right), q \leq n^{-c} \log n$ $O\left(\frac{c \log n}{\log(1+\frac{m}{n})}\right)$ if $\frac{m}{n} \geq \ln n$ with $q \leq n^{-c} \log n$
DER with Byzantine Nodes	$O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right), q \leq n^{-c} \log n$ $\Omega\left(\frac{\log n}{\log(1+m/n)}\right)$	$O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right), q \leq n^{1-c} \log n$ $\Omega\left(\frac{\log n}{\log(1+m/n)}\right)$	$O\left(f \cdot \left\lceil \frac{c}{m/n} \right\rceil \log n\right), q \leq n^{-c} \log n$
DER with Adversarial Edges	$O\left(\left\lceil \frac{c \cdot (n^2-k)}{(m-k)n} \right\rceil \log n\right)$ with $q \leq n^{-c} \log n$ $\Omega\left(\frac{\log n}{\log(1+m/n)}\right)$	$O\left(\left\lceil \frac{c \cdot (n^2-k)}{(m-k)n} \right\rceil \log n\right)$ with $q \leq n^{1-c} \log n$ $\Omega\left(\frac{\log n}{\log(1+m/n)}\right)$	$O\left(\left\lceil \frac{c \cdot (n^2-k)}{(m-k)n} \right\rceil \log n\right)$ with $q \leq n^{-c} \log n$

Figure 4.1: Our main results, where $c > 0$ is any constant and q is the failure probability.

Applications. Our results have some interesting applications. In an idea similar to Ghaffari, Kuhn and Su's work [73], All-to-All Broadcast allows us, e.g., to implement algorithms that run on a clique in a synchronous setting in our sparser graphs. Indeed, if All-to-All Broadcast needs R rounds to complete with high probability, then each round of communication of a clique can be simulated by R rounds of Uniformly Random Trees with high probability. Essentially, if an algorithm runs in T rounds, with $T \leq n^{c-1}$, in a clique network, we can implement it with high probability in $R \cdot T$ rounds in the Uniformly Random Trees network, which is essentially a logarithmic overhead. In particular, in the Uniformly Random Trees with Byzantine Nodes model, we have:

Theorem 4.2.3. *Let $\mathcal{A}$ be a distributed synchronous algorithm that runs on a static clique in T rounds, where $T \leq \alpha n^x$ for some constant $\alpha, x \in \mathcal{R}_+$, and has a probability of success p. Assume $\mathcal{A}$ is robust to f Byzantine nodes, and $f \leq \frac{2}{3}n - 1$. Then, assuming standard cryptographic tools[5], there exists a distributed algorithm $\mathcal{A}'$ that runs on Uniformly Random Trees in $T \cdot 144 \cdot \log n \cdot c$ rounds, and has a probability of success $p' \geq p(1 - \alpha n^{1+x-c})$, for any $c \geq 1 + x$. Moreover, $\mathcal{A}'$ is robust to f Byzantine nodes.*

In particular, we can apply known results on reliable Broadcast and Byzantine Consensus to show the following results:

Corollary 4.2.4. *For any $c \geq 1$, and $f \leq \frac{2}{3}n - 1$, in the Uniformly Random Trees with f Byzantine nodes, there exists an algorithm for Reliable Broadcast, that is robust to f Byzantine nodes, that runs in $(f+1) \cdot 144 \cdot c \cdot \log n$ rounds, and succeeds with probability $p \geq 1 - n^{2-c}$.*

[5]Specifically, our approach requires authenticated messages. Encryption may also be needed, only if the protocol $\mathcal{A}$ is vulnerable to eavesdropping. Both can be implemented using standard cryptographic tools.

Corollary 4.2.5. *For any $c \geq 1$ and $f < \frac{n}{3}$, in the Uniformly Random Trees with f Byzantine nodes, there exists an algorithm for Byzantine Consensus, that is robust to f Byzantine nodes, that runs in $3(f+1) \cdot 144 \cdot c \cdot \log n$ rounds, and succeeds with probability $p \geq 1 - 2n^{2-c}$.*

Throughout the paper, the filtration of the process is denoted as $\{\mathcal{F}_t\}_{t\in\mathbb{N}}$, that is, $\mathcal{F}_t$ is the amount of information available after timestep t.

Organization The paper is organized as follows. First, we give a new result on counting rooted trees in Section 4.3, which will be useful in our analysis. Afterwards, we explore the Uniformly Random Trees model in Section 4.4. Then, in Section 4.5, we expand our analysis to the Uniformly Random Trees with Byzantine Nodes model. In Section 4.6, we explore the case where the adversary controls k edges in each round. We study the directed ErdsRényi graphs model and its adversarial variants in Section 4.7. We review related work in Section 4.8. Appendix 4.9.1 gives a lower bound for deterministic Broadcast in constant-height trees. Appendix 4.9.2 gives the full details of Section 4.3. In Appendix 4.9.3, we give some probability theory results that are useful throughout the paper. Finally, in Appendix 4.9.4 and 4.9.6, we include omitted proofs from Sections 4.4 and 4.6 respectively, while Appendix 4.9.5 and 4.9.7 give the full details of Sections 4.5 and 4.7.

4.3 Counting Rooted Trees

Given a graph consisting of n vertices together with a directed rooted forest F of e edges on them, Pitman [116] showed in 1999 that there are n^{n-1-e} many directed rooted trees over these vertices that contain F. While useful, this result is not sufficient for our purposes as we need to count the number of trees with a given node v as root.

Thus, we show the following extended result:

Theorem 4.3.1. *Let us be given a directed rooted forest F on n vertices, let $v \in [n]$ be the root of a component in F, and f be the number of vertices of that component (note that we can have $f = 1$ if v is an isolated vertex). Then the number of directed rooted trees T on n vertices, such that F is contained in T, and such that v is the root of T, is $fn^{n-2-|E|}$.*

To show our result, we develop techniques which differ significantly from Pitman's proof. Indeed, Pitman relies on the symmetry of the vertices in the rooted tree. However, for our result, the symmetry is broken as one vertex is different from the others with the new requirement that it is the root. We hence make use of another type of symmetry in the trees in our analysis that is based on group actions.

We first ignore the orientations of the edges in F and find the set A_F of all undirected trees that contain F. We can compute the cardinality of that set with a result by Lu, Mohr and Székely [104]. We then root each of those trees at v. This will give a direction to every edge that might or might not agree with its direction in F. We now want to partition A_F into subsets such that all subsets have the same size and only one tree from each subset has edges that agree with the direction of F. The number we are looking for is then the number of subsets, which is the ratio between the cardinality of A_F and the size of the subsets.

To create the subsets, we introduce a specific group tailored to F, and an action of that group on A_F. It is known that the set of all orbits of the action partition A_F, and we show that exactly one element in each orbit has edges in the same direction as F. To see unicity, we take an element T of A_F that has edges in the same direction as F, and take an element $T' \neq T$ in its orbit, that is there exists a nontrivial group element g such that T' is obtained from T by applying the action of g to T. We show that this action must change the direction of at least one edge of F, and thus T' does not have edges in the same direction as F. For existence, we show that for every $T \in A_F$, we can find a group element g such that, if applied to T, yields a tree that has edges in the same direction as F. We then show how to compute the size of each orbit. This allows us to deduce the number of orbits, which equals the number of trees that we want to count.

The full details of the proof can be found in Appendix 4.9.2.

4.4 The Uniformly Random Trees Model

We now give a precise description of how information flows in Uniformly Random Trees over time. In this section, we will use Theorem 4.9.1, which states that the number of rooted trees on n nodes containing a given directed rooted forest F with e edges is n^{n-1-e}. Since all nodes are equivalent, we will at each step, divide the nodes into two sets: the set I of nodes that have received the message, called *informed* nodes, and the set S of remaining nodes, called *uninformed* nodes. We study how I grows over time.

For the rest of the section, I_t and S_t will, respectively, be the set of nodes that are informed and uninformed after round t. We set $I_0 = \{v_0\}$ and $S_0 = [n] - \{v_0\}$, where v_0 is the node that initially holds the message, $N_t = |I_t|$ to be the number of informed nodes after t rounds, and T_t to be the tree chosen at random in round t. For a tree T, for each node p, $P_T(p)$ is the (unique) parent of node p in T, unless p is the root of T, in which case $P_T(p) = p$. Simplifying the notation, we also use $P_t(p)$ to denote $P_{T_t}(p)$. All skipped proofs can be found in Appendix 4.9.4.

The central claim of the proof is the following lemma, which characterizes how many new nodes get informed in each round, depending on how many were informed after the previous round. This lemma shows that uninformed nodes get informed independently from each other.

Lemma 4.4.1. *For any $t > 0$, $N_{t+1} - N_t$ follows a binomial distribution with parameters $\left(n - N_t, \frac{N_t}{n}\right)$.*

The proof of this lemma shows that every uninformed node has probability $\frac{N_t}{n}$ of having an informed parent in round $t+1$, independently of whether the other uninformed nodes have an uninformed parent.

Proof. Let $I_t = \{i_1, \ldots, i_{N_t}\}$ and $S_t = \{s_1, \ldots, s_{n-N_t}\}$. We then have, for any integer x:

$$\mathbb{P}(N_{t+1} - N_t = x | \mathcal{F}_t) = \sum_{J \in A(S_t, x)} \mathbb{P}\left(\bigcap_{y \in J} (P_{t+1}(y) \in I_t) \bigcap_{y \in S_t \setminus J} (P_{t+1}(y) \notin I_t) \middle| \mathcal{F}_t \right),$$

where $A(S,x)$ with S being a set and x an integer denotes the set of subsets of S of size x. Our goal is to show that the events $P_t(y) \in I_t$ for different $y \in S_t$ are mutually independent. Let us look at the event $\bigcap_{y\in J}(P_t(y) \in I_t)$ for any $J \subseteq S_t$ (note that we do not require that J has a specific size here). We can then write, indexing a on J:

$$\mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(y) \in I_t)\middle|\mathcal{F}_t\right) = \sum_{a\in[N_t]^{|J|}} \mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(y) = i_{a_y})\middle|\mathcal{F}_t\right)$$
$$= \sum_{a\in[N_t]^{|J|}} \frac{\left|\{T \in \mathcal{T}_n : P_T(y) = i_{a_y}, \forall y \in J\}\right|}{|\mathcal{T}_n|}$$

Now consider the forest that is composed of stars whose centers are the i_{a_y} and whose leaves are the nodes $y \in J$. More specifically, consider the forest that contains the edges $(i_{a_y}, y), \forall y \in J$. Note that $\left|\{T \in \mathcal{T}_n : P_T(y) = i_{a_y}, \forall y \in J\}\right|$ equals the number of rooted trees that are compatible with this forest. By Theorem 4.9.1, we have that $\left|\{T \in \mathcal{T}_n : P_T(y) = i_{a_y}, \forall y \in J\}\right| = n^{n-1-|J|}$. This allows us to compute the above probability as follows:

$$\mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(y) \in I_t)\middle|\mathcal{F}_t\right) = \sum_{a\in[N_t]^{|J|}} \frac{n^{n-1-|J|}}{n^{n-1}} = \left(\frac{N_t}{n}\right)^{|J|}$$

This proves that the events $P_{t+1}(y) \in I_t$ for any two $y \in S_t$ are mutually independent (Definition 4.9.26), each having probability $\frac{N_t}{n}$. Going back to the first equation of this proof, we can now compute with Lemma 4.9.27 and using $\Delta_t := N_{t+1} - N_t$ as a shorthand:

$$\mathbb{P}(\Delta_t = x|\mathcal{F}_t) = \sum_{J\in A(S_t,x)} \prod_{y\in J} \mathbb{P}\left(P_{t+1}(y) \in I_t|\mathcal{F}_t\right) \prod_{y\in S_t\setminus J} \mathbb{P}\left(P_{t+1}(y) \notin I_t)|\mathcal{F}_t\right)$$
$$= \binom{n-N_t}{x}\left(\frac{N_t}{n}\right)^x\left(1-\frac{N_t}{n}\right)^{n-N_t-x}$$

□

Our next goal is to show that $N_t = n$ with high probability for all $t \geq 32 \cdot c \cdot \ln n$. To do so we introduce a random variable X_t that we use to lower bound N_t.

Definition 4.4.2 (Lower bound random variable for the number of informed nodes)**.** *Let X_t be the random variable defined recursively: $X_0 = 1$, and*

$$X_{t+1} = X_t + (n - X_t) \cdot \frac{X_t}{n} \qquad \text{if} \quad N_{t+1} - N_t \geq (n - N_t) \cdot \frac{N_t}{n}$$
$$X_{t+1} = X_t \qquad \text{if} \quad N_{t+1} - N_t < (n - N_t) \cdot \frac{N_t}{n}$$

Intuitively, X_t is a lower bound for N_t that increases if and only if $N_{t+1} - N_t$ exceeds its expectation. Therefore, we always have $n \geq N_t \geq X_t \geq 1$ (full proof in Appendix 4.9.4), and we can claim that $N_t = n$ as soon as $X_t > n - 1$. Moreover, we can compute the values of X_t after each increase:

Lemma 4.4.3. *Let $u_t \in \mathbb{N}$ be the t-th round such that $X_{u_t+1} > X_{u_t}$ and let $u_0 = 0$. Then $X_{u_t} = n - n\left(\frac{n-1}{n}\right)^{2^t}$. Moreover, we have that $X_{u_{t+1}} = X_{u_t} + (n - X_{u_t}) \cdot \frac{X_{u_t}}{n}$.*

This allows us to estimate when N_t reaches n, as when $s \geq 2\ln n$, $X_{u_s} > n - 1$.

Lemma 4.4.4. *If $t \geq u_{2\ln n}$, then $N_t = n$.*

All we need to show now is that $X_{t+1} - X_t$ exceeds its expectation often enough. For that, we use a result due to Greenberg and Mohri [78], that will give us an estimate of the probability of X_t strictly increasing in a given round.

Theorem 4.4.5 (Theorem 1 of [78])**.** *For any positive integer m and any probability p such that $p > \frac{1}{m}$, let B be a binomial random variable of parameters (m, p). Then, the following inequality holds:*

$$\mathbb{P}(B \geq mp) > \frac{1}{4}$$

In fact, in Lemma 4.9.25, we are able to relax the condition to $p > \frac{1}{3m}$ while keeping the same inequality. We use this lemma to lower bound the probability of X_t increasing in any round:

Lemma 4.4.6. *If $n > 4$, for every $t \in \mathbb{N}$, we have that $\mathbb{P}(X_{t+1} > X_t) \geq \frac{1}{4}$.*

We now show that, for any $c \geq 1$, over $32 \cdot c \cdot \ln n$ rounds, X_t increases at least $2\ln n$ times with high probability. For that, let $(B_t)_{t\in\mathbb{N}}$ be Bernoulli independent random variables of parameter $\frac{1}{4}$. Let $Z^B_{\leq t} = \sum_{z\in[t]} B_z$ and $Z_{\leq t} = \sum_{z\in[t]} \mathbb{1}(X_{z+1} > X_z)$. By Lemma 1.8.5 of [48] the following trivial corollary follows.

Corollary 4.4.7. *For any $\ell \in \mathbb{N}$, we have that $\mathbb{P}(Z_{\leq t} \leq \ell) \leq \mathbb{P}(Z^B_{\leq t} \leq \ell)$.*

Lemma 4.4.8. *Let $t = 32 \cdot c \cdot \ln n$ for any $c \geq 1$. Then $\mathbb{P}(Z_{\leq t} \leq 2\ln n) \leq \frac{1}{n^c}$.*

Proof. Note that $Z^B_{\leq t}$ is a binomial distribution of parameters $(t, \frac{1}{4})$. Using Hoeffding's inequality (Lemma 4.9.28), we have that:

$$\mathbb{P}(Z^B_{\leq t} \leq 2\ln n) \leq \exp\left(-2t\left(\frac{1}{4} - \frac{2\ln n}{t}\right)^2\right) \leq \exp\left(-2 \cdot 32c\ln n\left(\frac{1}{4} - \frac{2}{16}\right)^2\right) = n^{-c}$$

Corollary 4.4.7 then gives the desired result. □

We now have all the tools to prove Theorem 4.2.1, which we recall here:

Theorem 4.2.1. *For any $c \geq 1$ and $n \geq 5$, Broadcast on Uniformly Random Trees completes within $32 \cdot c \cdot \ln n$ rounds with probability $p > 1 - \frac{1}{n^c}$.*

Proof. By Lemma 4.4.8, we have that, with probability $p \leq 1 - \frac{1}{n^c}$, $X_{t+1} > X_t$ for at least $2\ln n$ many rounds within the $32 \cdot c \cdot \ln n$ first rounds. Recall that $u_{2\ln n}$ is the $2\ln n$-th round where $X_{t+1} > X_t$. We thus have that $\mathbb{P}\left(u_{2\ln n} \leq 32 \cdot c \cdot \ln n\right) \geq 1 - n^{-c}$. But, by Lemma 4.4.4 the event $u_{2\ln n} \leq 32 \cdot c \cdot \ln n$ implies the event $N_{32 \cdot c \cdot \ln n} = n$, therefore $\mathbb{P}\left(N_{32 \cdot c \cdot \ln n} = n\right) \geq 1 - n^{-c}$. □

We now show that this result is asymptotically tight. Indeed, we can show that if at most $\log n$ rounds are allowed, then with probability $q \geq \frac{1}{4}$, Broadcast does not complete:

Theorem 4.2.2. *If $n \geq 2$, then the probability that Broadcast (and All-to-All Broadcast) on Uniformly Random Trees fails to complete within $\log n$ rounds is at least $\frac{1}{4}$.*

Proof. We will only show the result for Broadcast, as the result for All-to-All Broadcast follows immediately. We will first show by induction that $\mathbb{E}(N_t) \leq X_{u_t}$ for every $t \in \mathbb{N}$. We will then conclude using Markov's inequality.

The induction basis is clear as $N_0 = X_0 = 1$. For the induction step, assume that for some $t \in \mathbb{N}$, we have that $\mathbb{E}(N_t) \leq X_{u_t}$. Let us show that this implies that $\mathbb{E}(N_{t+1}) \leq X_{u_{t+1}}$. Indeed, by Lemma 4.4.1, $N_{t+1} - N_t$ has a binomial distribution of parameters $n - N_t$ and $\frac{N_t}{n}$. This implies that:

$$\mathbb{E}[N_{t+1}|\mathcal{F}_t] = N_t + \frac{N_t}{n} \cdot (n - N_t) = 2N_t - \frac{N_t^2}{n}$$

Therefore:

$$\mathbb{E}[N_{t+1}] = \mathbb{E}\left[\mathbb{E}[N_{t+1}|\mathcal{F}_t]\right] = 2\mathbb{E}[N_t] - \frac{\mathbb{E}[N_t^2]}{n}$$

As $Var(N_t) = \mathbb{E}[N_t^2] - \mathbb{E}[N_t]^2 \geq 0$, we have that $-\mathbb{E}[N_t^2] \leq -\mathbb{E}[N_t]^2$. This implies:

$$\mathbb{E}[N_{t+1}] \leq 2\mathbb{E}[N_t] - \frac{\mathbb{E}[N_t]^2}{n}$$

Note that we have that, by Lemma 4.4.3:

$$X_{u_{t+1}} = 2X_{u_t} - \frac{X_{u_t}^2}{n}$$

Since $x \mapsto 2x - \frac{x^2}{n}$ is strictly increasing between 0 and n, with both X_t and $\mathbb{E}[N_t]$ falling in that range (Lemmata 4.9.29 and 4.9.30), the induction hypothesis implies that $2\mathbb{E}[N_t] - \frac{\mathbb{E}[N_t]^2}{n} \leq 2X_{u_t} - \frac{X_{u_t}^2}{n}$. This implies $\mathbb{E}[N_{t+1}] \leq X_{u_{t+1}}$.

We know the value of X_{u_t} from Lemma 4.4.3. We can thus give the upper bound $\mathbb{E}[N_{\log n}] \leq X_{u_{\log n}} = n(1 - ((n-1)/n)^n) \leq n(1 - \frac{1}{4})$, since $n \geq 2$. Using Markov's inequality, we thus have:

$$\mathbb{P}(N_{\log n} \geq n) \leq \frac{\mathbb{E}[N_{\log n}]}{n} = 1 - \frac{1}{4}$$

□

We now use Theorem 4.2.1 result to get a similar result for All-to-All Broadcast. Using a union-bound, we obtain:

Theorem 4.4.9. *For any $c \geq 1$ and $n \geq 5$, All-to-All Broadcast on Uniformly Random Trees completes within $32 \cdot c \cdot \ln n$ rounds with probability $p > 1 - \frac{1}{n^{c-1}}$.*

We now finally show a result on Consensus, which uses the following algorithm:

Algorithm 4.4.10. *The protocol works as follows: each node waits for $32 \cdot c \cdot \ln n$ rounds, during which if it receives the initial value of node 1, it starts forwarding it as well. After the $32 \cdot c \cdot \ln n$ rounds have passed, it outputs that value, or $\perp$ if it hasn't received it.*

Theorem 4.4.11. *For any $c \geq 1$ and $n \geq 5$, There exists a protocol for Consensus on Uniformly Random Trees that satisfies Agreement and Validity, terminates within $32 \cdot c \cdot \ln n$ rounds with probability $p > 1 - \frac{2}{n^c}$, and only requires messages of 1 bit over each edge in each round.*

Proof. Algorithm 4.4.10 is an algorithm where everyone agrees on v_1, the input to node 1, and where only v_1 is passed along. Thus every node outputs either v_1 or $\perp$. However, if v_1 has Broadcast within the first $32 \cdot c \cdot \ln n$ rounds, then everyone outputs v_1. This happens with probability $p \geq 1 - n^{-c}$, by Theorem 4.2.1. □

Note that Algorithm 4.4.10 can be adapted to different variants of Consensus. To keep our presentation concise, we do not explore them further in detail. For example, the version given here satisfies the condition that no node continues to communicate after it has decided on a value, but Consensus does not complete with probability 1 after everyone has decided as some nodes might output $\perp$. A different definition of Consensus could allow each node to send messages after it decides on a value, in which case a different version of the algorithm could be given, where each node can decide as soon as it receives the value v_1.

4.5 Adversarial Nodes: Trees with Byzantine Nodes

In this section, we will discuss the case where some nodes are *Byzantine*, that is, nodes that can arbitrarily deviate from the protocol. These nodes can stop functioning, send wrong messages, and coordinate to make the protocol fail. We will rely on cryptographic tools so that each node can sign and encrypt the message it sends. Then nodes can be confident about the sender of each message and its content and can forward the message along with its unchanged signature to other nodes. We will assume that there are up to f Byzantine nodes, out of a total of n nodes. We require that $f \leq \frac{2}{3}n - 1$. Nodes that are not Byzantine are called honest.

We begin by analyzing Broadcast in this setting. We first give a message to a fixed honest node, and ask the node to forward it to all other honest nodes. Note the difference between this model and the reliable Broadcast model, where the initial message could be from an honest node or a Byzantine node, and where if the initial message is from a Byzantine node, then the message accepted by each honest node must be the same.

In our setting, the best strategy for the Byzantine node is not to forward any message at all. Indeed, they cannot modify the content of a message because they cannot forge any signature, and, thus, their power is limited. Hence, we will analyze this problem as if Byzantine nodes

are just defunct but the process that chooses the communication network, i.e., the random tree, does not know which nodes are Byzantine and, thus, they are part of the network as before, i.e., the tree still consists of n nodes.

As most of the analysis resembles the one of the previous section, all details are delayed to Appendix 4.9.5. We get the main theorem of this section:

Theorem 4.5.1. *For any $c \geq 1$, and $f \leq \frac{2}{3}n - 1$, Broadcast on Uniformly Random Trees with f Byzantine nodes completes within $144 \cdot c \cdot \log n$ rounds with probability $p > 1 - \frac{1}{n^c}$.*

We now use this result to get a similar result for All-to-all Broadcast. Using a union-bound, we obtain:

Theorem 4.5.2. *For any $c \geq 1$, and $f \leq \frac{2}{3}n - 1$, All-to-all Broadcast on Uniformly Random Trees with f Byzantine nodes completes within $144 \cdot c \cdot \log n$ rounds with probability $p > 1 - n^{1-c}$.*

This allows us, e.g., to implement algorithms that run on a clique in a synchronous setting in our sparser graph. Indeed, each round of communication of a clique can be simulated by $32 \cdot c \cdot \tau$ rounds of Uniformly Random Trees with high probability, since All-to-All Broadcast needs $32 \cdot c \cdot \tau$ rounds to complete with high probability. Essentially, if an algorithm runs in T time, with $T \leq n^{c-1}$, in a clique network, we can implement it with high probability in $32 \cdot c \cdot T \cdot \tau$ rounds in the Uniformly Random trees network, which is essentially a logarithmic overhead. The only caveat is that if T is too large, i.e. $T > n^{c-1}$, the probability of at least one of the T All-to-All Broadcast rounds failing can become close to 1. To circumvent this, we restrict ourselves to the case where T is a small enough polynomial in n.

Theorem 4.2.3. *Let $\mathcal{A}$ be a distributed synchronous algorithm that runs on a static clique in T rounds, where $T \leq \alpha n^x$ for some constant $\alpha, x \in \mathcal{R}_+$, and has a probability of success p. Assume $\mathcal{A}$ is robust to f Byzantine nodes, and $f \leq \frac{2}{3}n - 1$. Then, assuming standard cryptographic tools[6], there exists a distributed algorithm $\mathcal{A}'$ that runs on Uniformly Random Trees in $T \cdot 144 \cdot \log n \cdot c$ rounds, and has a probability of success $p' \geq p(1 - \alpha n^{1+x-c})$, for any $c \geq 1 + x$. Moreover, $\mathcal{A}'$ is robust to f Byzantine nodes.*

We now give two applications of this theorem, namely Reliable Broadcast and Byzantine Consensus.

Corollary 4.2.4. *For any $c \geq 1$, and $f \leq \frac{2}{3}n - 1$, in the Uniformly Random Trees with f Byzantine nodes, there exists an algorithm for Reliable Broadcast, that is robust to f Byzantine nodes, that runs in $(f+1) \cdot 144 \cdot c \cdot \log n$ rounds, and succeeds with probability $p \geq 1 - n^{2-c}$.*

Proof. Dolev and Strong [50] have given an algorithm that solves reliable Broadcast, is robust to f Byzantine nodes, and runs in $T = f+1$ rounds. Since $T \leq n$, we can apply Theorem 4.2.3 with $x = 1, \alpha = 1$, and we get the desired result. □

[6]Specifically, our approach requires authenticated messages. Encryption may also be needed, only if the protocol $\mathcal{A}$ is vulnerable to eavesdropping. Both can be implemented using standard cryptographic tools.

Corollary 4.2.5. *For any $c \geq 1$ and $f < \frac{n}{3}$, in the Uniformly Random Trees with f Byzantine nodes, there exists an algorithm for Byzantine Consensus, that is robust to f Byzantine nodes, that runs in $3(f+1) \cdot 144 \cdot c \cdot \log n$ rounds, and succeeds with probability $p \geq 1 - 2n^{2-c}$.*

Proof. Berman, Garay and Perry [22] have given an algorithm (known as the King's algorithm) that solves Byzantine Consensus, is robust to f Byzantine nodes, and runs in $T = 3(f+1)$ rounds. Since $T \leq 2n$, we can apply Theorem 4.2.3 with $x = 1, \alpha = 2$, and we get the desired result. □

4.6 Adversarial Edges: Trees with Adversarial Topology

In this section, we consider a more general model where a parametrized adversary controls a certain number of edges in every round, and the others are chosen randomly. More specifically, in each round, the adversary A chooses k edges such that the resulting graph is a directed rooted forest F, and then a tree is chosen uniformly at random among the rooted trees that are compatible with F. We consider the model where the adversary has access to the randomly chosen trees of all previous rounds, but has no information on the random coin flips of the current and future rounds.

All the proofs missing in this section can be found in Appendix 4.9.6.

We will prove the following theorem:

Theorem 4.6.1. *If the adversary controls k edges in each round, for $k \leq \frac{2}{3}n - 1$, then for any $c \geq 1$, with probability $p \geq 1 - n^{-c}$, Broadcast completes within $O(k + \log n)$ rounds.*

We will in fact show that Broadcast completes within $O(k+\tau)$ rounds, where $\tau = \frac{\log n}{\log(1+\frac{n-k}{2n})} = \Theta(\log n)$.

For the rest of the section, I_t and S_t will, respectively, be the set of nodes that are informed and uninformed after round t. We set $I_0 = \{1\}$ and $S_0 = [n] - \{1\}$, $N_t = |I_t|$ to be the number of informed nodes after t rounds, and T_t to be the tree chosen at random in round t. For a tree T, for each node p, $P_T(p)$ is the (unique) parent of node p in T, unless p is the root of T, in which case $P_T(p) = p$. Simplifying the notation, we also use $P_t(p)$ to denote $P_{T_t}(p)$.

We start by finding the best strategy A could use and then analyze that strategy.

4.6.1 Best Strategy for the Adversary A

In this subsection, we will show that the best strategy for the adversary is to use all the edges to form one tree, with as many uninformed nodes as possible. The main idea is that an uninformed node with an uninformed parent is "protected" in the round, that is, it cannot have an informed parent. Hence the adversary will try to protect as many uninformed nodes as possible by creating a tree connecting them. If the adversary still has edges left after it protected all the nodes it could, they use the remaining edges to connect as many informed nodes as possible to the tree such that there is no edge from an informed to an uninformed node to prevent that any of them becomes a parent of the root of the tree.

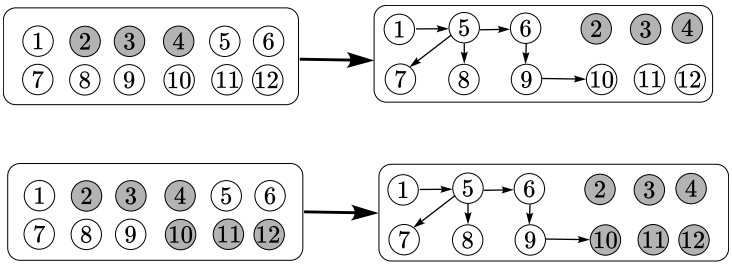


Figure 4.2: The best strategy for the adversary A, with $k = 6$. Shaded nodes are informed nodes. In the top example, nodes $5, 6, 7, 8, 9$ and 10 are safe from being informed, whereas node 1 can still be informed. In the bottom example, nodes $5, 6, 7, 8$, and 9 are safe, whereas node 1 can still be informed. However, node 1 is safe from being informed by node 10.

An illustration of this strategy is shown in Figure 4.2. This section is dedicated to formalizing and proving these ideas. We will use the notion of stochastic dominance. Intuitively, if a strategy yields more informed nodes than another one, then the adversary will choose the latter one. Stochastic dominance is the tool we use to formalize this. Note that we define stochastic dominance for two types of random variables, namely random variables that are real numbers and random variables that are sets. Thus, for any set S, let $\mathcal{P}(S)$ be the set of all subsets of S.

Definition 4.6.2 (Stochastic dominance)**.** *(1) A real random variable Y_1 stochastically dominates another real random variable Y_2, if, for every $x \in \mathcal{R}$, we have that $\mathbb{P}(Y_1 \geq x) \geq \mathbb{P}(Y_2 \geq x)$.*

(2) A random variable Y_1 with values in $\mathcal{P}([n])$ stochastically dominates another random variable Y_2 with values in $\mathcal{P}([n])$, if, for every $x \in \mathbb{N}$, we have that $\mathbb{P}(|Y_1| \geq x) \geq \mathbb{P}(|Y_2| \geq x)$.

With stochastic dominance, we will use a related notion, that is coupling. Coupling is a useful tool to compare two random variables, and in particular, it helps translate probabilistic events into deterministic ones, which are easier to analyze.

Definition 4.6.3 (Coupling)**.** *A coupling of two random variables Y_1, Y_2 is a third random variable $(\hat{Y}_1, \hat{Y}_2)$ such that Y_1 has the same distribution as $\hat{Y}_1$, and Y_2 has the same distribution as $\hat{Y}_2$.*

Next we state two coupling theorems, namely one for random variables that are real numbers and one for random variables that are sets.

Theorem 4.6.4 (Stochastic Dominance and Coupling, Theorem 7.1 of [43])**.** *If a real random variable Y_1 stochastically dominates another real random variable Y_2, then there exists a coupling $(\hat{Y}_1, \hat{Y}_2)$ of Y_1 and Y_2 such that*

$$\mathbb{P}(\hat{Y}_1 \geq \hat{Y}_2) = 1$$

Theorem 4.6.5 (Stochastic Dominance and Coupling, Theorem 7.8 of [43])**.** *If a random variable Y_1 with values in $[n]$ stochastically dominates another random variable Y_2 with values in $[n]$, then there exists a coupling $(\hat{Y}_1, \hat{Y}_2)$ of Y_1 and Y_2 such that*

$$\mathbb{P}\left(\left|\hat{Y}_1\right| \geq \left|\hat{Y}_2\right|\right) = 1$$

The next lemma contains a crucial observation: being greedy in each round is an optimal strategy for the adversary.

Lemma 4.6.6 (Distribution Domination)**.** *Let t be a round. Let E_1, E_2 be two sets of edges the adversaries could choose for round t. Let $N_t^{(1)}$ (resp. $I_t^{(1)}$) be the number (resp. set) of informed nodes after round t if E_1 is chosen, and $N_t^{(2)}$ (resp. $I_t^{(2)}$) if E_2 is chosen. Then if $\mathbb{P}(N_t^{(1)} \geq m) \geq \mathbb{P}(N_t^{(2)} \geq m)$ for every $m \in \mathbb{N}$ (that is, if $N_t^{(1)}$ stochastically dominates $N_t^{(2)}$), then choosing E_2 is a better strategy for the adversary than choosing E_1.*

Intuitively, the way to prove this is to build, for any strategy the adversary might use after choosing E_1, another strategy that would work better if used after choosing E_2. To prove that it is indeed the case, we couple these two strategies to prove that after any round, the number of informed nodes in one strategy stochastically dominates the number of informed nodes in the other one. The full details of the proof can be found in Appendix 4.9.6.

The next step is to show that the adversary will never force an edge from an informed node to an uninformed one. Indeed, intuitively, this means the adversary forces a node to be informed, which is against its interests. To do so, we introduce the notions of *non-increasing* and *increasing* trees, and show that A' will never choose an increasing tree. An illustration is given in Figure 4.3.

Definition 4.6.7 (Non-increasing rooted tree)**.** *A rooted tree U at round t is said to be* non-increasing in round t *if all edges in U whose source is in I_{t-1} have their target in I_{t-1} as well. Otherwise a tree is* (information)-increasing in round t.

To show that the worst-case adversary never uses an increasing tree, we introduce the notion of a *correction* of an increasing tree, which will be non-increasing, and show that choosing the correction is a better strategy for the adversary than choosing the increasing tree.

Definition 4.6.8 (Isomorphism)**.** *We say that a rooted tree U on n nodes is isomorphic to a rooted tree U' on n nodes if there exists a bijection b from $[n]$ to $[n]$ such that for every (directed) edge $(u, v) \in U$, we have that $(b(u), b(v)) \in U'$, and for every (directed) edge $(u, v) \in U'$, we have that $(b^{-1}(u), b^{-1}(v)) \in U$.*

In particular, if r is the root of U, then $b(r)$ is the root of U'.

Definition 4.6.9 (Correction of a tree)**.** *A* correction *of a tree U that is increasing at round t is a tree U' over the same nodes as U that (1) is isomorphic to U, (2) is non-increasing in round t, and (3) whose root is a node $s \in S_{t-1}$ such that $P_U(s) \in I_{t-1}$.*

Intuitively, if a tree is increasing, one can correct it by putting all the informed nodes at the bottom of the tree.

Lemma 4.6.10. *For any increasing tree U, there exists a correction U'.*

The following lemma proves that the worst-case adversary will never choose a set of edges such that one (or more) component is increasing. Indeed, if such components existed, then

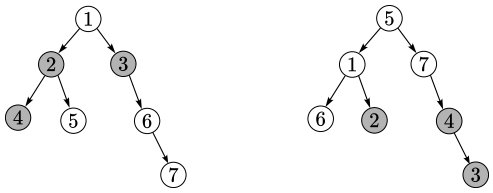


Figure 4.3: Shaded nodes are informed nodes. Left: A tree U that is information increasing. Right: A tree U' that is a correction of U.

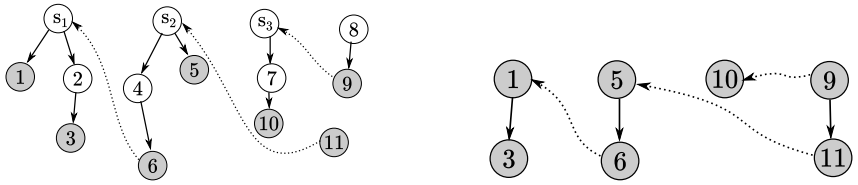


Figure 4.4: Illustration for the proof of Lemma 4.6.12, Case 1. Shaded nodes are informed nodes. Left: Solid lines represent E. Dotted lines are a suitable choice of a. Right: Solid lines represent F associated to E. Dotted lines represent $b(a)$. Any tree rooted at 9 on the right yields a suitable choice of a on the left.

the adversary would have replaced all of them with non-increasing ones, as this will lead to no fewer and potentially more rounds. Therefore, we can assume in the following that all components are non-increasing.

Lemma 4.6.11. *Let t be a round and N_{t-1} be the number of informed nodes after round $t-1$. Let E_1, E_2 be two sets of edges that the adversary could choose for round t such that*

1. *E_1 is a collection of rooted trees such that at least one tree U is information-increasing, and*
2. *E_2 is obtained from E_1 by replacing U with a correction U' of U.*

Let $N_t^{(1)}$ be the number of informed nodes after round t if E_1 is chosen, and let $N_t^{(2)}$ be that number if E_2 is chosen. Then choosing E_2 is a better strategy for the adversary than choosing E_1.

The next step is to show that if the adversary chooses a forest, all edges will be used in one component. For that, we introduce the notion of *merging trees*, and show that if the adversary chooses a forest with 2 or more non-trivial components, then merging two of those non-trivial components will yield a better strategy for the adversary. We start by computing the probability that a set of roots of the forest given by the adversary get informed:

Lemma 4.6.12. *Let t be a round, let E be the set of k edges forming a directed rooted forest over $[n]$ which the adversary chooses in round t such that each component of E is non-increasing, and let $s_1, \ldots, s_x$ be uninformed nodes that are roots of their component (which might have size only 1). Note that $\{s_1, \ldots, s_x\}$ needs not be the set of the roots of all components, simply a collection of some of them. Let $\eta_1, \ldots, \eta_x$ be the number of informed nodes in the component of $s_1, \ldots, s_x$ respectively, and let η be the number of informed nodes outside the components of $s_1, \ldots, s_x$. Then we have that:*

$$\mathbb{P}\left(\cap_{j\in[x]}(P_t(s_j)\in I_{t-1})\middle|\mathcal{F}_{t-1}\right)=\frac{\eta(\eta+\sum_{j\in[x]}\eta_j)^{x-1}}{n^x}=\frac{\eta(N_{t-1})^{x-1}}{n^x}$$

Proof. We have that:

$$\mathbb{P}\left(\cap_{j\in[x]}(P_t(s_j)\in I_{t-1})\middle|\mathcal{F}_{t-1}\right)=\sum_{a\in(I_{t-1})^x}\mathbb{P}\left(\cap_{j\in[x]}(P_t(s_j)=a_j)\middle|\mathcal{F}_{t-1}\right)$$

However, many terms of that sum are equal to 0. Indeed, for example, if a_1 is one of the η_1 informed nodes in the component of s_1, then $\mathbb{P}(P_t(s_1)=a_1)=0$. More generally, if the choice of a is such that $E\cup\bigcup_{j\in[i]}(a_j,s_j)$ contains an (undirected cycle), in other words, is incompatible with a rooted tree, then $\mathbb{P}(P_t(s_1)=a_1)=0$. If, on the other hand, the choice of a is compatible with a rooted tree, then, applying Theorem 4.9.1, we have:

$$\mathbb{P}\left(\cap_{j\in[x]}(P_t(s_j)=a_j)\middle|\mathcal{F}_{t-1}\right)=\frac{\left|T\in\mathcal{T}_n:(E\bigcup_{j\in[x]}(a_j,s_j))\subset T\right|}{|T\in\mathcal{T}_n:E\subset T|}=\frac{n^{n-1-|E|-x}}{n^{n-1-|E|}}=n^{-x}$$

We now have to count how many choices of a are compatible with a rooted tree. Let us call these the *suitable* choices of a. To do so we create a bijection between the set of all suitable choices of a and a simple set of forests consisting only of trees that are line graphs. This basically says that for counting the number of suitable choices, we can ignore the internal structure of each tree.

Case 1: A figure for that case is given in Figure 4.4. Let us first assume that none of the η_j nor η is equal to 0. Let α denote the set of all such values of a, and define β as follows: create a forest F with $x+1$ (directed) line graphs, each line having respectively $\eta_1,\dots,\eta_x,\eta$ nodes. Then β is the set of all rooted trees that are compatible with F, and whose root is the root of the last tree of F.

To determine $|\alpha|$, we show that there is a bijection between α and β and determine $|\beta|$. To create the bijection first take an arbitrary but fixed bijection b that maps every informed node from I_{t-1} to a node from F, such that an informed node from the component of s_j is mapped to a node of the j-th line of F. Recall that each $a\in\alpha$ assigned a parent a_j to each node s_j with $1\le j\le x$. We can map a choice of $a\in\alpha$ to a tree $T\in\beta$ by setting the parent in T of the root of the j-th line to be $b(a_j)$ for every j. Note that this uniquely identifies a tree of β. Conversely, to find a choice $a\in\alpha$ from a tree $T\in B$, set $a_j=b^{-1}(p_j)$ where p_j is the parent of the root of the j-th line of F in T. Now note that β is the set of all rooted trees that are compatible with F, and whose root is the root of the last tree of F. By Theorem 4.3.1, $|\beta|=\eta(\eta+\sum_{j\in[x]}\eta_j)^{x-1}$, which concludes the proof for this case.

Case 2: If $\eta=0$, it is easy to see that no choice of a is compatible with a rooted tree, as a assigns a parent to each root s_j for $1\le j\le x$.

Case 3: If there exists some values of j such that $\eta_j=0$, then assume wlog that $\eta_1=\dots=\eta_\ell=0$, and $\eta_j>0$ for every $j>\ell$. By the same arguments as in Case 1, there will be $\eta(\eta+\sum_{j\in[x]}\eta_j)^{x-\ell-1}$ suitable choices for $(a_{\ell+1},\dots,a_x)$. Once this choice is made, for every $1\le j\le\ell$, a_j can take any value in I_{t-1}, where $|I_{t-1}|=\eta+\sum_{j\in[x]}\eta_j$. Thus, the total number of choices for a is $\eta(\eta+\sum_{j\in[x]}\eta_j)^{x-1}$. □

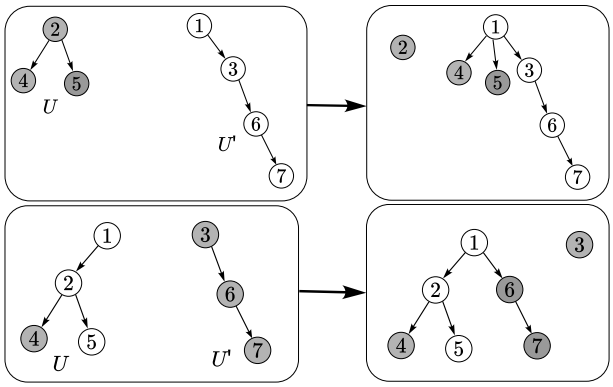


Figure 4.5: Merging examples

The following merge operation combines two trees such as to make a uninformed root the root of the merged tree, if at least one of the roots is uninformed.

Definition 4.6.13 (Merging trees)**.** *We say that we* merge *two non-trivial trees U and U' with respective roots r and r' in round t when we apply the following operation:*

- *If $r \in I_{t-1}$, then for every $p \in U$ with $(r,p) \in U$, replace edge (r,p) with the edge (r',p).*
- *If $r \notin I_{t-1}$, then for every $p \in U'$ with $(r',p) \in U'$, replace edge (r',p) with the edge (r,p).*

Lemma 4.6.14. *Let t be a round and N_{t-1} be the number of informed nodes after round $t-1$. Let E_1, E_2 be two sets of edges that the adversary could choose for round t, as follows: let E_1 be a collection of rooted trees such that every tree is non-increasing, with at least two non-trivial components U with root r and U' with root r', and let E_2 be obtained from E_1 by merging U and U'. Let $N_t^{(1)}$ be the number of informed nodes after round t if E_1 is chosen, and $N_t^{(2)}$ if E_2 is chosen. Then choosing E_2 is a better strategy for the adversary than choosing E_1.*

This lemma implies that the adversary will never choose a set of edges with more than one non-trivial component, i.e., the adversary will choose *one* tree with $k+1$ nodes. We already showed that the adversary will only choose non-increasing components. Therefore, we are left with analyzing the case where the adversary chooses one non-trivial non-increasing tree with $k+1$ nodes.

Lemma 4.6.15. *Let t be a round and N_t be the number of informed nodes after round t. Let U be a non-increasing tree over $k+1$ nodes in round $t+1$. Let σ be the number of uninformed nodes in U and η the number of informed nodes in U. Then the distribution of $N_{t+1} - N_t$ equals the sum of of $n - N_t - \sigma$ independent Bernoulli random variables of parameter $\frac{N_t}{n}$ plus one Bernoulli random variable of parameter $\frac{N_t - \eta}{n}$.*

Corollary 4.6.16. *Let t be a round and N_t be the number of informed nodes after round t. Let U be a non-increasing tree over $k+1$ nodes in round $t+1$ and let η be its number of informed nodes in U. The optimal strategy for the adversary is to minimize η in every round.*

Proof. Recall σ is the number of uninformed nodes in U. Note that we always have $\sigma + \eta = k + 1$. Let us consider two non-increasing trees U and U' over $k+1$ nodes. Let η_1 (resp. σ_1) be the number of informed (resp. uninformed) nodes in U, and η_2 (resp. σ_2) be the number of informed (resp. uninformed) nodes in U'. Assume wlog that $\eta_1 > \eta_2 \geq 0$. Then $\sigma_1 < \sigma_2$. Let $N_{t+1}^{(1)}$ and $N_{t+1}^{(2)}$ be the number of informed nodes after round $t+1$ if the adversary chooses respectively tree U or U'. The distribution of $N_{t+1}^{(1)} - N_t$ is the sum of at least $n - N_t - \sigma_1$ independent Bernoulli variables of parameter $\frac{N_t}{n}$, while $N_{t+1}^{(2)} - N_t$ is the sum of at most $n - N_t - \sigma_2 + 1$ independent Bernoulli variables of parameter at most $\frac{N_t}{n}$. The first distribution clearly dominates the second, and by the Distribution Domination Lemma (Lemma 4.6.6), the result holds. □

This shows that the optimal strategy for the adversary is always to choose the number σ of uninformed nodes in the tree U chosen by the adversary equal to $k+1$, unless the number of informed nodes N_{t-1} is so large that σ is smaller than $k+1$, in which case $\sigma = n - N_{t-1}$, which is the number of uninformed nodes. As the number N_t of informed nodes never decreases, this leads to the following partitioning of the rounds into two phases: one phase which contains all rounds t where the number of uninformed nodes is at least $k+1$, i.e., $n - N_{t-1} \geq k+1$, in which case $\sigma = k+1$, and another phase which contains all rounds t with $n - N_{t-1} < k+1$, in which case $\sigma = n - N_{t-1}$. We will show that the first phase takes $O(\log n)$ rounds, while the second one takes $O(k + \log n)$ rounds.

4.6.2 Phase 1

In phase 1, we have that $N_{t+1} - N_t$ follows a binomial distribution of parameters $(n-k-N_t, \frac{N_t}{n})$. This is exactly the same evolution as the case detailed in Section 4.5, or more precisely the result of Lemma 4.9.33. We hence get the same result as in Theorem 4.5.1:

Lemma 4.6.17. *If $k \leq \frac{2}{3}n - 1$ then, for any $c \geq 1$, Phase 1 ends within $32 \cdot c \cdot \tau$ rounds with probability $p \geq 1 - n^{-c}$, where $\tau = \frac{\log n}{\log\left(1 + \frac{n-k}{2n}\right)}$.*

4.6.3 Phase 2

Phase two starts when there are only k uninformed nodes. This essentially means that the adversary can protect all uninformed nodes but one, as the trees they will choose will have an uninformed root, which might get informed in this round. Note that all uninformed nodes below it will not become informed in the current round.

Lemma 4.6.18. *If $k \leq \frac{2}{3}n - 1$, for any $c \geq 1$, Phase 2 ends within $8c \cdot \max\{\ln n, \frac{kn}{n-k-1}\} \leq 12c \cdot \max\{\ln n, k\}$ rounds with probability $p \geq 1 - n^{-c}$.*

Proof. Recall that in this phase every uninformed node belong to U. Thus, there is only one node, namely the root of U that can be informed through the randomly chosen edges. In each round, by Lemma 4.6.15 with $\sigma = n - N_t$, exactly one node, which as discussed must be the root, gets informed with probability $\frac{n-k-1}{n}$. Assimilating this to a flip of a coin where the coin has probability $\frac{N_t - ((k+1) - |S_t|))}{n} = \frac{n-k-1}{n}$ of landing on heads, and flipping the coin

$8c \cdot \max\{\ln n, \frac{kn}{n-k-1}\}$ times, we are asking what the probability p of the coin landing on heads at least k times is. Again, using Hoeffding's inequality (Lemma 4.9.28), we have that:

$$\begin{aligned}1-p &\leq \exp\left(-2 \times 8c \cdot \max\{\ln n, \frac{kn}{n-k-1}\}\left(\frac{n-k-1}{n} - \frac{k}{8c \cdot \max\{\ln n, \frac{kn}{n-k-1}\}}\right)^2\right) \\ &\leq \exp\left(-2 \times 8c \cdot \ln n \cdot \left(\frac{n-k-1}{n}\right)^2 \left(1-\frac{1}{8c}\right)^2\right) \\ &\leq \exp\left(-2 \times 8c \cdot \ln n \cdot \frac{1}{9}\left(1-\frac{1}{4}\right)^2\right) \leq \exp\left(-c \ln n\right) \leq n^{-c}\end{aligned}$$

Where we use that $k(n-k-1) \geq n-2 \geq 2n$ if $n \geq 4$. □

4.6.4 Combining Phase 1 and 2

We now just have to combine the results for Phases 1 and 2 to show that Broadcast completes in $O(\ln n + k)$ rounds:

Theorem 4.6.19. *If the adversary can control k edges in each round, with $k \leq \frac{2}{3}n - 1$, Broadcast completes within $32 \cdot \tau \cdot c + 12c \cdot \max\{\ln n, k\} = O(c \cdot (\ln n + k))$ rounds with probability $p \geq 1 - 2n^{-c}$.*

Proof. This is a direct result of Lemmata 4.6.17 and 4.6.18 □

In order to understand how tight this bound is, we give a lower bound on how many rounds the adversary can delay Broadcast:

Theorem 4.6.20. *If $n \geq 2$, and the adversary controls k edges in each round, then there exists a strategy for the adversary that delays Broadcast with a Randomized Oblivious Message Adversary to at least $\frac{kn}{2(n-k-1)} \geq \frac{k}{2}$ rounds with probability at least $\frac{1}{4}$.*

Proof. Let us look at an adversary that only uses a non-increasing tree over nodes 1 to $k+1$, where the root is always chosen to be the one with the smallest ID among those that are still uninformed. Let N'_t be the number of informed nodes after t rounds among $[k+1]$. Clearly $N'_0 \leq 1$. By Lemma 4.6.16, the root of the tree has probability at most $\frac{n-k-1}{n}$ of being informed in each round, and thus in expectation, $\mathbb{E}[N'_t] \leq 1 + \frac{t(n-k-1)}{n}$. Therefore, using Markov's inequality, we have that, for $t = \frac{kn}{2(n-k-1)}$:

$$\mathbb{P}(N_t = n) \leq \mathbb{P}(N'_t \geq k+1) \leq \frac{1 + \frac{kn(n-k-1)}{2(n-k-1)n}}{k+1} = \frac{1}{k+1} + \frac{k}{2k+2} \leq \frac{3}{4}.$$

□

Applying a union-bound on the result for Broadcast, we get a result for All-to-All Broadcast:

Theorem 4.6.21. *For any $c \geq 1$ and $n \geq 5$, All-to-All Broadcast on Uniformly Random Trees with adversarial edges completes within $O(c \cdot (\ln n + k))$ rounds with probability $p > 1 - \frac{1}{n^{c-1}}$.*

4.6.5 Consensus

Finally, we see that a direct application of Theorem 4.6.21 gives us a reliable algorithm for Consensus with a Randomized Oblivious Message Adversary of parameter k:

Theorem 4.6.22. *There exists a protocol for Consensus with a Randomized Oblivious Message Adversary that satisfies Agreement and Validity, and terminates in $O(c\cdot(\ln n+k))$ rounds with probability $p \geq 1-\frac{2}{n^c}$, and only requires messages of 1 bit over each edge in each round, as long as $k \leq \frac{2}{3}n-1$.*

Proof. By Theorem 4.6.19, node 1 Broadcasts within $O(c\cdot(\ln n+k))$ rounds with probability $p \geq 1-2n^{-c}$. Therefore, using the same arguments as in the proof of Theorem 4.4.11 it follows that Algorithm 4.4.10 achieves Consensus within $O(c\cdot(\ln n+k))$ rounds with probability $p \geq 1-2n^{-c}$, as long as we let the for loop run for $O(c\cdot(\ln n+k))$ rounds instead of $32\cdot c\cdot\ln n$ rounds. □

4.7 Beyond Trees: Broadcast and Consensus in directed Erdös–Rényi graphs

Directed ErdsRényi graphs consist of m edges chosen uniformly at random among the n^2 potential edges. Intuitively they have less structure than uniformly random trees, which makes the analysis of Broadcast simpler. We present the main ideas below. Note that we also analyze Byzantine nodes and adversarial edges in that model in Section 4.7, but omit these extensions in this overview.

Sampling a directed ErdsRényi graph is equivalent to choosing m edges *without* replacement from the set of all possible edges. We call that Scheme 1. Then we observe, using a coupling argument, that Scheme 1 requires no more rounds than Scheme 2, where in each round m edges are chosen *with* replacement. Finally, to analyze Scheme 2, we basically partition the sequence of rounds of Scheme 2 into $2\lceil(\log n)/2\rceil$ phases, such that for each of the first $\lceil(\log n)/2\rceil$ phases the number of informed nodes doubles in each phase and for each of the last $\lceil(\log n)/2\rceil$ phases the number of uninformed nodes halves in each phase. Note that Broadcast completes after the last phase. Using Hoeffding's inequality for binomial distributions we show that phase i for $1 \leq i \leq \lceil\log n/2\rceil$ requires with high probability at most $O(\max\{\log n, 2^{i-1}\}n/2^{i-1})$ sampled edges, and, thus, $O(\lceil\max\{\log n, 2^{i-1}\}/(2^{i-1}m/n)\rceil)$ rounds, and for $\lceil\log n/2\rceil+1 \leq i \leq 2\lceil\log n/2\rceil$ phase i requires with high probability at most $O(\max\{\log n, 2^{j-2}\}n/2^{j-1})$ sampled edges with with $j := 2\lceil\log n/2\rceil - i$, and, thus, $O(\lceil\max\{\log n, 2^{j-1}\}/(2^{j-1}m/n)\rceil)$ rounds. Summed over all phases this shows that with high probability $O(\lceil n/m\rceil\log n)$ rounds suffice for Scheme 2 to reach Broadcast. We thus get the result:

Theorem 4.7.1. *For any $c \geq 1$, in scheme 2, and therefore scheme 1, Broadcast completes within $O\left(\left\lceil\frac{cn}{m}\right\rceil\log n\right)$ rounds with probability $p \geq 1-n^{-c}\log n$.*

Note that the analysis extends to the setting when the graph in each round contains *at least* m edges. We also show that a lower bound that implies that this upper bound is tight for $m \leq n$.

Theorem 4.7.2. *In scheme 1, and thus in scheme 2, Broadcast fails to complete within* $\frac{\log(n)-1}{\log(1+m/n)}$ *rounds with probability at least* $\frac{1}{2}$.

We also give somewhat different analysis where the number of informed resp. uninformed nodes does not double, but increases by $(1 + m/n)$ that is tight for $m \geq n \ln n$.

Theorem 4.7.3. *For any* $c \geq 1$ *and* $m \in [n^2]$ *such that* $m/n \geq \ln n$*, in scheme 2 and in scheme 1, Broadcast completes within* $O\left(\frac{c \cdot \log n}{\log(1+m/n)}\right)$ *rounds with probability* $p \geq 1 - n^{-c} \log n$.

We also extend those results to Consensus, Byzantine nodes and with adversarial edges. All details are delayed to Appendix 4.9.7.

4.8 Related Work

Information dissemination in general and Broadcasting and Consensus in particular are fundamental topics in distributed computing. In contrast to this paper, most classic literature on network Broadcast as well as on related tasks such as gossiping and Consensus, considers a static setting, e.g., where in each round each node can send information to one neighbor [87, 67].

Especially the Byzantine setting has received much attention in the literature. Important results include Dolev and Strong [50] on reliable Broadcast which is robust to f Byzantine nodes, and runs in $T = f + 1$ rounds, or Berman, Garay and Perry [22] on King's algorithm that solves reliable Broadcast, is robust to f Byzantine nodes, and runs in $T = 3(f + 1)$ rounds. To just name a few.

In terms of dynamic networks, Kuhn, Lynch and Oshman [98] explore the all-to-all data dissemination problem (gossiping) in an undirected setting, where nodes do not know beforehand the total number of nodes and must decide on that number. Dutta, Pandurangan, Rajaraman, Sun and Viola [55] generalize the model to when not all nodes need to forward their message, but only k tokens must be forwarded. Augustine, Pandurangan, Robinson and Upfal [14] show that if the graph is an expander in every round, broadcast is complete within $O(\log n)$ rounds, even if a small enough constant fraction of nodes get churned in each round. Ahmadi, Kuhn, Kutten, Molla and Pandurangan [3] study the message complexity of Broadcast also in an undirected dynamic setting, where the adversary pays up a cost for changing the network.

In dynamic networks, the oblivious message adversary is a commonly considered model, especially for Broadcast and Consensus problems, first introduced by Charron-Bost and Schiper [34]. The Broadcast problem under oblivious message adversaries has been studied for many years. A first key result for this problem was the $n \log n$ upper bound by Zeiner, Schwarz, and Schmid [129] who also gave a $\left\lceil \frac{3n-1}{2} \right\rceil - 2$ lower bound. Another important result is by Függer, Nowak, and Winkler [68] who presented an $O(\log \log n)$ upper bound if the adversary can only choose nonsplit graphs; combined with the result of Charron-Bost, Függer, and Nowak [33] that states that one can simulate $n - 1$ rounds of rooted trees with a round of a nonsplit graph, this gives the previous $O(n \log \log n)$ upper bound for Broadcasting on trees. Dobrev and Vrto [47, 46] give specific results when the adversary is restricted to hypercubic and tori graphs with some missing edges. El-Hayek, Henzinger, and Schmid [61, 60] recently settled

the question about the asymptotic time complexity of Broadcast by giving a tight $O(n)$ upper bound, also showing the upper bound still holds in more general models. Regarding Consensus, Coulouma, Godard and Peters in [42] presented a general characterization on which dynamic graphs Consensus is solvable, based on Broadcastability. Winkler, Rincon Galeana, Paz, Schmid, and Schmid [69] recently presented an explicit decision procedure to determine if Consensus is possible under a given adversary, enabling a time complexity analysis of Consensus under oblivious message adversaries, both for a centralized decision procedure as well as for solving distributed Consensus. They also showed that reaching Consensus under an oblivious message adversary can take exponentially longer than Broadcasting.

In contrast to the above works, in this paper we study a more randomized message adversary, considering a stochastic model where adversarial graphs are partially chosen uniformly at random. While a randomized perspective on dynamic networks is natural and has been considered in many different settings already, existing works on random dynamic communication networks, e.g., on the radio network model [63], on rumor spreading [41], as well as on epidemics [54], do not consider oblivious message adversaries. Note, however, that the information dissemination considered in this paper is similar to the SI model for virus propagation, with results having implications in both directions [65]. For example, Doerr and Fouz [49] introduced an information dissemination protocol inspired by epidemics. More generally, randomized information dissemination protocols can be well-understood from an epidemiological point-of-view, and are very similar to the SI model which has been very extensively studied. In contrast to the typical SI models considered in the literature [110], however, our model in this paper revolves around tree communication structures which introduce additional technical challenges. Furthermore, existing literature often provides results in expectation, while we in this paper provide tail bounds.

Many papers have tried to bridge the gap between the deterministic and random case, using smoothed analysis. In [106], Meir, Paz and Schwartzman study the broadcast problem in noisy networks, under different definitions on noise. In particular, if in each round the graph given by the adversary is replaced by a graph chosen uniformly at random among graphs at hamming distance at most k from the original graph, in the case where the adversary can suggest any connected graph, then Broadcast is reduced from n rounds to $O(\min\{n, n\sqrt{\frac{\log n}{k}}\})$ rounds, in the case of an adaptive adversary. If the adversary is oblivious, then Dinitz, Fineman, Gilbert and Newport [45] showed that it is further reduced to $O(n^{2/3}/k^{1/3} \times \log n)$.

4.9 Appendix

4.9.1 Lower Bound for Deterministic Broadcast in Constant Height Trees

In this section, we consider a very similar model to [60], the only difference being that the adversary is restricted to choosing trees of height at most 2.

Model We are given n nodes, and these nodes can communicate in synchronous rounds. Each node has a distinct I.D., and aims to share this I.D. with as many nodes as possible. In the beginning, each node only knows its own I.D.. An adversary chooses for each round a

directed network along which nodes can communicate, among a set A of allowed networks. In each round, each node sends all I.D.s it has received in previous rounds to each one of its out-neighbors. The adversary's goal it to maximize the number of rounds until broadcast, that is, until one I.D. has been received by everyone. The question is: how many rounds can the adversary delay broadcast, depending on A?

Authors in [60] have shown that if A is the set of rooted trees, then the adversary can delay broadcast for a linear number of rounds. Since a linear number of rounds is easily achievable by the adversary simply by taking a line graph L, and using L as the communication network in each round, one would think that the height of the trees allowed play an important role to determine broadcast time. We give in Figure 4.6 a counter example, where A is the set of rooted trees of height at most 2, and where broadcast needs at least a linear number of rounds.

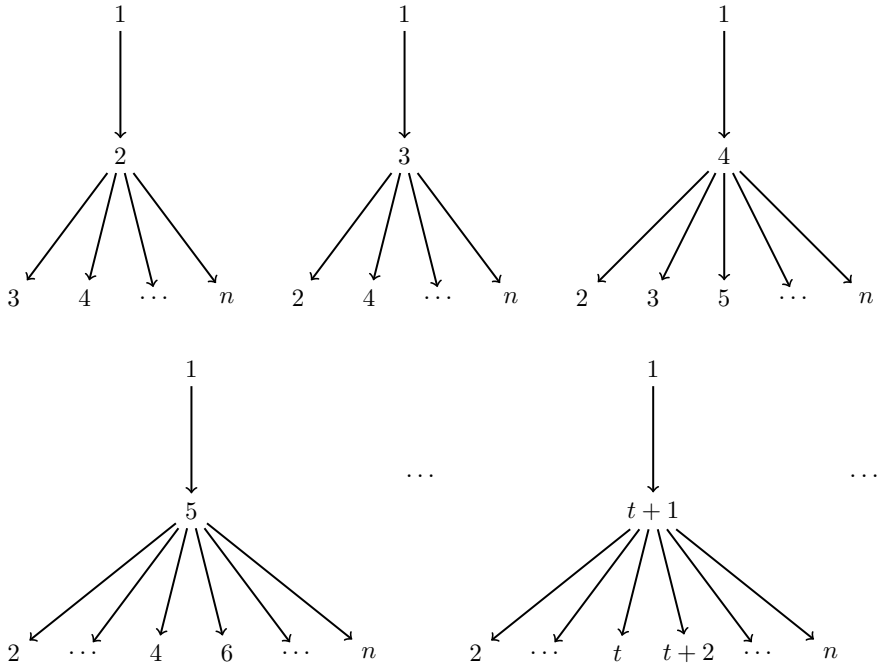


Figure 4.6: Lower Bound for (deterministic) Broadcast when the adversary is restricted to trees of height at most 2.

In this example, in round t, for $t < n-2$, the adversary chooses the tree rooted at node 1, with edges $(1, t+1)$ and $(t+1, i)$ for every $i \in [n] \setminus \{1, t+1\}$. Since node 1 never has an in-neighbor, broadcast completes when the I.D. of node 1 is shared to every node. It is easy to see that this only happens after round $n-2$.

4.9.2 Counting Trees

In this section, we will present previously known and new results on the number of rooted trees that satisfy given properties. This will be helpful for computing probabilities in later sections. Namely, we are particularly interested in the two following results:

Theorem 4.9.1 (Lemma 1 of [116]). *Let us be given a directed rooted forest F on n vertices, and let $|E|$ be the number of edges in F. Then, the number of directed rooted trees T over n vertices, such that $F \subseteq T$, is $n^{n-1-|E|}$.*

Theorem 4.3.1. *Let us be given a directed rooted forest F on n vertices, let $v \in [n]$ be the root of a component in F, and f be the number of vertices of that component (note that we can have $f = 1$ if v is an isolated vertex). Then the number of directed rooted trees T on n vertices, such that F is contained in T, and such that v is the root of T, is $fn^{n-2-|E|}$.*

We will also prove Theorem 4.3.1. To do so, we start by recalling Cayley's formula [31]:

Theorem 4.9.2 (Cayley's formula). *The number of undirected trees on n vertices is n^{n-2}.*

As a corollary of this theorem, we can compute the number of rooted trees on n vertices, as choosing a rooted tree is equivalent to choosing an undirected tree, and then choosing a root:

Corollary 4.9.3. *The number of rooted trees on n vertices is n^{n-1}.*

Throughout this section we use F to denote an *undirected or directed* forest and $C_1, C_2, \ldots, C_m$ of $f_1, \ldots, f_m$ vertices with integer $m \geq 1$ to denote the connected components of (the undirected version of) F. The next theorem on undirected trees gives the number of undirected trees which respect a set of fixed edges. It was shown by Lu, Mohr and Székely [104].

Theorem 4.9.4 (Lemma 6 of [104]). *Let us be given an undirected forest F on n vertices, with connected components $C_1, C_2, \ldots, C_m$ of $f_1, \ldots, f_m$ vertices with integer $m \geq 1$. Let $|E|$ be the number of edges in F. Then, the number of undirected trees T on n vertices, such that $F \subseteq T$, is:*

$$\left(\prod_{i \in [m]} f_i \right) n^{n-2-|E|}$$

We also recall the definition of a directed rooted forest, illustrated in Figure 4.7:

Definition 4.9.5 (Directed rooted forest). *A directed rooted forest is a collection of disjoint directed rooted trees.*

For simplicity, we will always require that $\sum_{i \in [m]} f_i = n$, which is always achievable by putting isolated vertices in trivial components. We will also assume that $v \in C_1$. For any directed graph G, $u(G)$ will represent its undirected version (Illustrated in Figure 4.7). For any directed rooted tree T, its root is denoted by $r(T)$. We will also use the following bijection. Recall that $\mathcal{T}_n$ is the set of all directed rooted trees on n vertices. We use T_n to denote the set of all undirected trees on n vertices.

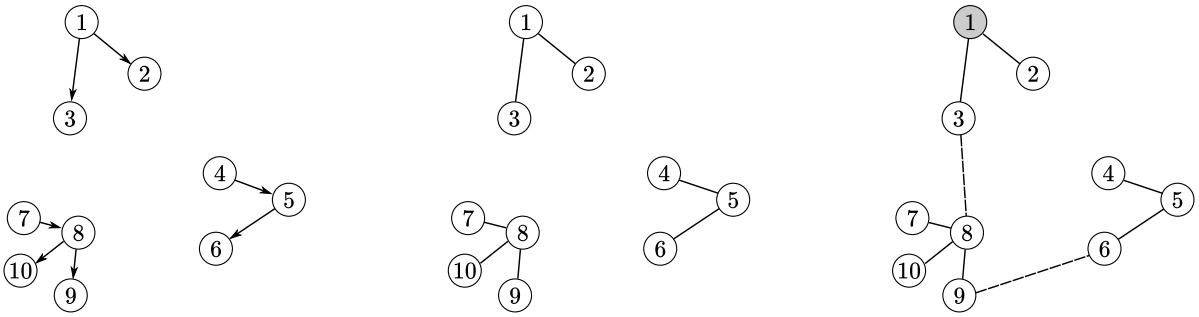


Figure 4.7: Left: A directed rooted forest F. Middle: $u(F)$. Right: A directed rooted tree T in A_F, as seen as an undirected tree with a choice of a root. Note that $F \not\subset T$.

Definition 4.9.6 (Bijection between rooted trees and undirected trees with a choice of a root)**.** *Let T_n be the set of all undirected trees on n vertices. We define π to be the following bijection:*

$$\begin{aligned} \pi \colon \mathcal{T}_n &\to T_n \times [n] \\ T &\mapsto (u(T), r(T)) \end{aligned}$$

To prove Theorem 4.3.1, we will first look at all the trees rooted at v that agree with F if edge directions are ignored. Choosing such a tree is equivalent to choosing an undirected tree that contains F, then choosing v as the root. This results in $\left(\prod_{i\in[m]} f_i\right) n^{n-2-E}$ trees. However, while all of them agree with F on the undirected edges, the direction of those edges will not correspond for a majority of them. We will then partition this set of trees such that only one element of each set of the partition agrees with F on the directed edges, and counting the number of sets in the partition will yield the desired result. To do so, we will use group actions.

Definition 4.9.7 (Group action)**.** *If G is a group with identity element e, and X is a set, then a (left) group action α of G on X is a function*

$$\alpha \colon G \times X \to X$$

that satisfies the following two axioms:

- *Identity: $\alpha(e, x) = x, \forall x \in X$, where e is the identity element of G.*
- *Compatibility: $\alpha(g, \alpha(h, x)) = \alpha(gh, x), \forall g, h \in G, \forall x \in X$*

Definition 4.9.8 (Rotations)**.** *Let $k > 0$ be an integer and let R_k be the group of all rotations of $[k]$, that is, the set of functions:*

$$\begin{aligned} \sigma_i^k \colon \mathbb{Z}/k\mathbb{Z} &\to \mathbb{Z}/k\mathbb{Z} \\ x &\mapsto (x + i) \mod k \end{aligned}$$

Definition 4.9.9 (Undirected-compatible)**.** *Let F be a forest with vertices in $[n]$ (rooted and directed or undirected), and T a tree with vertices in $[n]$ (rooted and directed or undirected). We say that they are* undirected-compatible *if $u(F) \subseteq u(T)$, where $u(G)$ represents the undirected version of graph G. If F and T are both rooted and directed or both undirected, we say that they are* compatible *if $F \subseteq T$.*

Definition 4.9.10 (Set of undirected-compatible trees to a forest)**.** *Let us be given a directed rooted forest F with vertices in $[n]$. A_F is the set of directed rooted trees on n vertices, rooted at v, that are undirected-compatible with F.*

An example of a tree in A_F is given in Figure 4.7. The following lemma follows almost immediately from Theorem 4.9.4.

Lemma 4.9.11. *Let F be a directed rooted forest with n vertices and E edges. Then $|A_F| = \left(\prod_{i\in[m]} f_i\right) n^{n-2-|E|}$.*

Proof. Let B_F be the set of all undirected rooted trees that are undirected-compatible with F. π induces a bijection between A_F and $B_F \times \{v\}$. Therefore, $|A_F| = |B_F| \cdot 1$. By Theorem 4.9.4, $|B_F| = \left(\prod_{i\in[m]} f_i\right) n^{n-2-|E|}$. □

Definition 4.9.12 (Bijection between a set of f_i elements and $\mathbb{Z}/f_i\mathbb{Z}$)**.** *For any $i \in [m]$, there exists a bijection between $\mathbb{Z}/f_i\mathbb{Z}$ and C_i. Let b_i be that bijection.*

Let $R = R_{f_2} \times \ldots \times R_{f_k}$, an illustration of an element of R is given in Figure 4.8. Note that R is a group as a cartesian product of groups. We now define a group action of R on A_F. This group action will allow us to partition A_F as desired.

Definition 4.9.13 (Group action of R on A_F)**.** *Given a forest F with connected components C_i with $1 \leq i \leq m$ and corresponding bijections b_i, let α be the group action of R on A_F defined as follows: Let $\sigma = (\sigma_{a_2}^{f_2}, \ldots, \sigma_{a_m}^{f_m})$ for some $(a_2, \ldots, a_m) \in \mathbb{Z}/f_2\mathbb{Z} \times \cdots \times \mathbb{Z}/f_m\mathbb{Z}$ be an element of R and let $T \in A_F$. Then $\alpha(\sigma, T)$ is obtained from T by making the following modifications to $\pi(T) = (u(T), v)$:*

For every $i \in \{2, \ldots m\}$, there is one (and only one) path from v to C_i in $u(T)$. Let (x, y) be the only edge on that path such that $x \notin C_i, y \in C_i$. Replace edge (x, y) with edge $(x, b_i\sigma_{a_i}^{f_i} b_i^{-1}(y))$.

To prove that this is indeed a group action, we need to verify (1) that $\alpha(\sigma, T)$ is indeed in A_F, (2) that the identity element $e = (\sigma_0^{f_1}, \ldots, \sigma_0^{f_m})$ of R verifies $\alpha(e, T) = T$ for any $T \in A_F$, and (3) that for any two $\sigma, \tau \in R$, for any $T \in A_F$, we have $\alpha(\sigma, \alpha(\tau, T)) = \alpha(\sigma\tau, T)$. The second condition being trivial as $\sigma_0^{f_i}$ is the identity function for any value of f_i, we only prove the other two.

Lemma 4.9.14. *$\alpha(\sigma, T) \in A_F$.*

Proof. Let us first show that $u(\alpha(\sigma, T))$ is an undirected tree. As it has $n - 1$ edges, we only need to show that it is connected. Let u be a vertex. We need to show that it can be reached from v. Let P be the (only) path from v to u in T, written as a sequence of vertices. Then we can split up P into $P = P_1 P_2 \ldots P_z$, where each P_j is a sequence of vertices that all belong to the same C_i for some $i \in [m]$. We will now replace each of the P_j by another path to make a path from v to u in $u(\alpha(\sigma, T))$.

Consider every edge (x, y) where x is the last vertex of P_j for some j, and y is the first vertex of P_{j+1}. There exists some k such that $y \in C_k$. Then $P_1 P_2 \ldots P_j y$ is the path from $r(T)$ to C_k in $u(T)$. Then $(x, b_k^{-1}\sigma_{a_k}^{f_k} b_k(y)) \in u(\alpha(\sigma, T))$. Replace y by $b_k^{-1}\sigma_{a_k}^{f_k} b_k(y)$ in P.

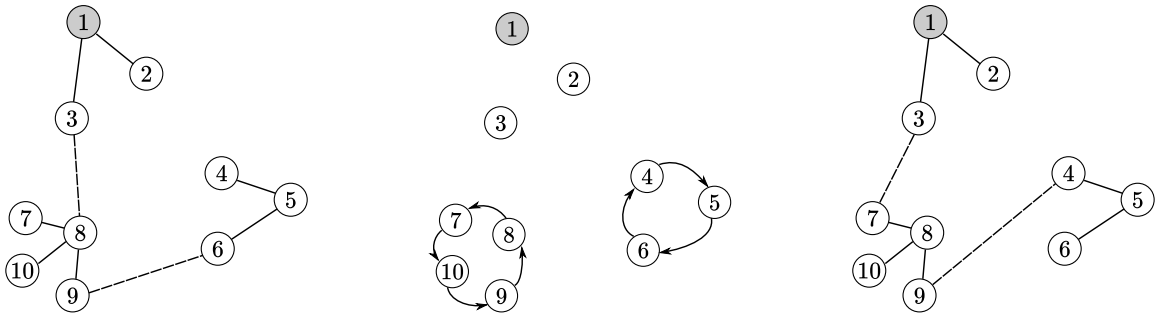


Figure 4.8: Left: An element T in A_F. Middle: An element σ of R. Right: The result $\alpha(\sigma, T)$ of the action α of σ on T. Note that $F \subseteq \alpha(\sigma, T)$, where F is the example from Figure 4.7, with this particular choice of σ.

Let us now look at a particular P_j, and let i be such that all of the vertices of P_j belong to C_i, then its first vertex has been changed to another vertex of C_i, while all others are unchanged. Hence, the first and last vertex still belong to C_i. As C_i is connected in $u(\alpha(\sigma, T))$ since no edge inside C_i has been modified, there exists a path P'_j in $u(\alpha(\sigma, T))$ that connects that first and last vertex of P_j. We can thus replace P_j by P'_j.

The new path now correctly connects v and u in $u(\alpha(\sigma, T))$, which shows that it is connected. Hence $\alpha(\sigma, T)$ is a tree. Since no edge in any particular C_i has been modified, $\alpha(\sigma, T)$ is compatible with F. □

Lemma 4.9.15. *For any $T \in A_F$, and $\sigma, \tau \in R$ we have that $\alpha(\sigma, \alpha(\tau, T)) = \alpha(\sigma\tau, T)$.*

Proof. Let $\sigma = (\sigma_{a_2}^{f_2}, \ldots, \sigma_{a_m}^{f_m})$ and $\tau = (\sigma_{c_2}^{f_2}, \ldots, \sigma_{c_m}^{f_m})$. And then, for every $i \in \{2, \ldots, m\}$, the path from v to C_i in T will include the edge (x, y) such that $x \notin C_i, y \in C_i$, then the corresponding edge is $(x, b_i \sigma_{c_i}^{f_i} b_i^{-1}(y))$ in $\alpha(\tau, T)$, and $(x, b_i \sigma_{a_i}^{f_i} \sigma_{c_i}^{f_i} b_i^{-1}(y))$ in $\alpha(\sigma\tau, T)$. Hence, it is $(x, b_i \sigma_{a_i}^{f_i} b_i^{-1} b_i \sigma_{c_i}^{f_i} b_i^{-1}(y))$ in $\alpha(\sigma, \alpha(\tau, T))$. We thus have $\alpha(\sigma, \alpha(\tau, T)) = \alpha(\sigma\tau, T)$. □

As we plan to use Lagrange's theorem for group actions, we now compute the *stabilizer* of a tree T, which is the set of all rotations that do not modify the tree:

Lemma 4.9.16. $R_T := \{\sigma \in R : \alpha(\sigma, T) = T\} = \{e\}$, *for every $T \in A_F$.*

Proof. Let $\sigma \in R$ be a rotation such that $\alpha(\sigma, T) = T$. For every $i \in \{2, \ldots m\}$, we look at the path from v to C_i in both T and $\alpha(\sigma, T)$. These two paths must be the same. However, if the first element of that path in T that is in C_i is some vertex y, then in $\alpha(\sigma, T)$, it is $b_i \sigma_{a_i}^{f_i} b_i^{-1}(y)$. We conclude that $b_i \sigma_{a_i}^{f_i} b_i^{-1}(x) = x$ and thus $a_i = 0$.

We therefore have that $a_i = 0$ for every $i \in \{2, \ldots, m\}$, which proves that $\sigma = (\sigma_0^{f_2}, \ldots, \sigma_0^{f_m}) = e$. □

We now take a look at the orbit $R \cdot T$ of a tree $T \in A_F$. The group action ensures that the orbits in A_F form a partition of A_F.

Theorem 4.9.17 (Lagrange's Theorem, Corollary 10.23 of [119])**.** *Let G be a group, X a set and α a group action of G on X. Let x be an element of X, $G_x := \{g \in G : \alpha(g, x) = x\}$ and $G.x := \{y \in X : \exists g \in G, y = \alpha(g, x)\}$. Then we have that:*

$$|G.x| = \frac{|G|}{|G_x|}$$

Lemma 4.9.18. *Let, for every $T \in A_F$, $R \cdot T := \{T' \in A_F : \exists \sigma \in R, \alpha(\sigma, T) = T'\}$. Then $|R \cdot T| = \prod_{i \in \{2,\dots,m\}} f_i$.*

Proof. By Theorem 4.9.17, we have that $|R \cdot T| = \frac{|R|}{R_T} = \frac{\prod_{i \in \{2,\dots,m\}} f_i}{1}$. □

We now show that exactly one tree in each orbit is compatible with F.

Lemma 4.9.19. *Let $T \in A_F$. Then there exists exactly one $T' \in R \cdot T$ such that T' is compatible with F.*

Proof. Let $T' \in R \cdot T$ be a tree such that T' is compatible with F, and let σ be the rotation such that $T' = \alpha(\sigma, T)$. Let, for every $i \in [m]$, r_i be the root of C_i in F.

For every $i \in \{2, \dots, m\}$, look at the path from v to C_i in T, and its corresponding path in T', computed similarly to the proof of Lemma 4.9.14. In T', the first vertex of that path in C_i must be r_i, but it also is $b_i \sigma_{a_i}^{f_i} b_i^{-1}(y)$, where y is the first vertex of the path in T. Hence $a_i = b_i^{-1} r_i - b_i^{-1}(y)$.

These conditions uniquely determine σ, and, thus, T'. Conversely, setting σ with each a_i defined as above gives a tree T' that is compatible with F. □

We can now prove Theorem 4.3.1, which we recall below:

Theorem 4.3.1. *Let us be given a directed rooted forest F on n vertices, let $v \in [n]$ be the root of a component in F, and f be the number of vertices of that component (note that we can have $f = 1$ if v is an isolated vertex). Then the number of directed rooted trees T on n vertices, such that F is contained in T, and such that v is the root of T, is $f n^{n-2-|E|}$.*

Proof. Consider set A_F as defined in Definition 4.9.10. We know that every directed rooted spanning tree T in K_n such that F is contained by T, and such that v is the root of T, is in A_F. We can partition A_F in orbits of the group action defined in Definition 4.9.13. By Lemma 4.9.18, each orbit has $\prod_{i \in \{2,\dots,m\}} f_i$ elements, and thus we have $\frac{|A_F|}{\prod_{i \in \{2,\dots,m\}} f_i}$ orbits, which is equal to $f_1 n^{n-2-|E|}$ by Lemma 4.9.11. Lemma 4.9.19 ensures that exactly one element in each orbit is a directed rooted spanning tree T in K_n such that F is contained by T. Note that $f_1 = f$ to conclude the proof. □

4.9.3 Probabilities tools

Lemma 4.9.20. *Let $X_1, \dots, X_m$ and $Y_1, \dots Y_m$ be binary random variables such that for every $I \subseteq [m]$ we have that $\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{i\notin I} (X_i = 0)) = \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{i\notin I} (Y_i = 0))$, then the probability distribution of $\sum_{i\in[m]} X_i$ is equal to the probability distribution of $\sum_{i\in[m]} Y_i$.*

Proof. We have, for every $k \in [m]$:

$$\begin{aligned}\mathbb{P}\left(\sum_{i\in[m]} X_i = k\right) &= \sum_{|I|=k} \mathbb{P}\left(\cap_{i\in I}(X_i = 1) \cap_{i\notin I} (X_i = 0)\right) \\ &= \sum_{|I|=k} \mathbb{P}\left(\cap_{i\in I}(Y_i = 1) \cap_{i\notin I} (Y_i = 0)\right) \\ &= \mathbb{P}\left(\sum_{i\in[m]} Y_i = k\right)\end{aligned}$$

□

Lemma 4.9.21. *Let $X_1, \dots, X_m$ and $Y_1, \dots Y_m$ be binary random variables such that for every $\ell \in \mathbb{N}$, $\sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{i\notin I} (X_i = 0)) = \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{i\notin I} (Y_i = 0))$, then the probability distribution of $\sum_{i\in[m]} X_i$ is equal to the probability distribution of $\sum_{i\in[m]} Y_i$.*

Proof. We have, for every $k \in [m]$:

$$\begin{aligned}\mathbb{P}\left(\sum_{i\in[m]} X_i = k\right) &= \sum_{|I|=k} \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{i\notin I} (X_i = 0)) \\ &= \sum_{|I|=k} \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{i\notin I} (Y_i = 0)) \\ &= \mathbb{P}\left(\sum_{i\in[m]} Y_i = k\right)\end{aligned}$$

□

Lemma 4.9.22. *Let $X_1, \dots, X_m$ and $Y_1, \dots Y_m$ be binary random variables such that for every $I \subset [m], \mathbb{P}(\cap_{i\in I}(X_i = 1)) = \mathbb{P}(\cap_{i\in I}(Y_i = 1))$, then the probability distribution of $\sum_{i\in[m]} X_i$ is equal to the probability distribution of $\sum_{i\in[m]} Y_i$.*

Proof. We start by proving by induction on the size of J, $\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J} (X_j = 0)) = \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J} (Y_j = 0))$ for any $I, J \subseteq [n]$ such that $I \cap J = \varnothing$. This is clear for $|J| = 0$.

Let $I, J \subseteq [n]$ such that $I \cap J = \varnothing$ and $|J| > 0$. Let a be an element of J. Then we have:

$$\begin{aligned}&\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0)) \\ &\qquad = \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J} (X_j = 0)) + \mathbb{P}(\cap_{i\in I\cup\{a\}}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0))\end{aligned}$$

Similarly:

$$\mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0)) \\ = \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J} (Y_j = 0)) + \mathbb{P}(\cap_{i\in I\cup\{a\}}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))$$

By induction hypothesis, we have:

$$\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0)) = \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0)) \\ \mathbb{P}(\cap_{i\in I\cup\{a\}}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0)) = \mathbb{P}(\cap_{i\in I\cup\{a\}}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))$$

Hence:

$$\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J} (X_j = 0)) = \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J} (Y_j = 0))$$

The result follows from Lemma 4.9.20, when we take $J = [m] \setminus I$ □

Lemma 4.9.23. *Let $X_1, \ldots, X_m$ and $Y_1, \ldots Y_m$ be binary random variables such that $\sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(X_i = 1)) = \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(Y_i = 1))$ for every $\ell \in \mathbb{N}$, then the probability distribution of $\sum_{i\in[m]} X_i$ is equal to the probability distribution of $\sum_{i\in[m]} Y_i$.*

Proof. We start by proving by induction on k, that for every $\ell, k \in \mathbb{N}, \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J_I} (X_j = 0)) = \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J_I} (Y_j = 0))$ for any choice of $J_I \subseteq [n]$ such that $I \cap J_I = \varnothing$ and $|J_I| = k$. This is clear for $k = 0$.

For the induction case, let us assume, that for $k > 1$, we have that for every $\ell \in \mathbb{N}, \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J_I} (X_j = 0)) = \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J_I} (Y_j = 0))$ for any choice of $J_I \subseteq [n]$ such that $I \cap J_I = \varnothing$ and $|J_I| = k - 1$.

Let us fix ℓ, and for every $I \subseteq [m]$ such that $|I| = \ell$, let $J_I \subseteq [m]$ be such that $I \cap J_I = \varnothing$ and $|J_I| = k > 0$. Let a_I be an element of J_I. Then we have:

$$\sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J_I\setminus\{a_I\}} (X_j = 0)) \\ = \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J_I} (X_j = 0)) + \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I\cup\{a_I\}}(X_i = 1) \cap_{j\in J_I\setminus\{a\}} (X_j = 0))$$

Similarly:

$$\sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J_I\setminus\{a_I\}} (Y_j = 0)) \\ = \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J_I} (Y_j = 0)) + \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I\cup\{a_I\}}(Y_i = 1) \cap_{j\in J_I\setminus\{a_I\}} (Y_j = 0))$$

By induction hypothesis, we have:

$$\sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J_I\setminus\{a_I\}} (X_j = 0)) = \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J_I\setminus\{a_I\}} (Y_j = 0)) \\ \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I\cup\{a_I\}}(X_i = 1) \cap_{j\in J_I\setminus\{a_I\}} (X_j = 0)) = \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I\cup\{a_I\}}(Y_i = 1) \cap_{j\in J\setminus\{a_I\}} (Y_j = 0))$$

Hence:

$$\sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J_I} (X_j = 0)) = \sum_{|I|=\ell} \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J_I} (Y_j = 0))$$

This concludes the induction step.

The result follows from Lemma 4.9.21, when we take for every I, $J_I = [m] \setminus I$, for $k = m - \ell$. □

Lemma 4.9.24. *Let $X_1, \ldots, X_m$ and $Y_1, \ldots Y_m$ be binary random variables, $\alpha \in \mathcal{R}, \alpha \geq 1$ and $r \in \mathbb{N}$ such that for any $I \subseteq [m] \setminus \{r\}, \mathbb{P}(\cap_{i\in I}(X_i = 1)) = \mathbb{P}(\cap_{i\in I}(Y_i = 1))$, and $\mathbb{P}(\cap_{i\in I\cup\{r\}}(X_i = 1)) = \alpha\mathbb{P}(\cap_{i\in I\cup\{r\}}(Y_i = 1))$ then $\sum_{i\in[m]} X_i$ stochastically dominates $\sum_{i\in[m]} Y_i$.*

Proof. We start by proving by induction on the size of J, for every $I, J \subset [m] \setminus \{r\}$ such that $I \cap J = \varnothing, \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J} (X_j = 0)) = \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J} (Y_j = 0))$. This is clear for $|J| = 0$.

Let $I, J \subseteq [n]$ such that $I \cap J = \varnothing$ and $|J| > 0$. Let a be an element of J. Then we have:

$$\begin{aligned}&\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0))\\ &\qquad = \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J} (X_j = 0)) + \mathbb{P}(\cap_{i\in I\cup\{a\}}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0))\end{aligned}$$

Similarly:

$$\begin{aligned}&\mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))\\ &\qquad = \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J} (Y_j = 0)) + \mathbb{P}(\cap_{i\in I\cup\{a\}}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))\end{aligned}$$

By induction hypothesis, we have:

$$\begin{aligned}\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0)) &= \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))\\ \mathbb{P}(\cap_{i\in I\cup\{a\}}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0)) &= \mathbb{P}(\cap_{i\in I\cup\{a\}}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))\end{aligned}$$

Hence:

$$\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J} (X_j = 0)) = \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J} (Y_j = 0))$$

We then show by induction on the size of J, that for every $I, J \subset [m]$ such that $r \in I$, $I \cap J = \varnothing, \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J} (X_j = 0)) = \alpha\mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J} (Y_j = 0))$ for any $I, J \subseteq [n]$. This is clear for $|J| = 0$.

Let $I, J \subseteq [n]$ such that $r \in I$, $I \cap J = \varnothing$ and $|J| > 0$. Let a be an element of J. Then we have:

$$\begin{aligned}&\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0))\\ &\qquad = \mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J} (X_j = 0)) + \mathbb{P}(\cap_{i\in I\cup\{a\}}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0))\end{aligned}$$

Similarly:

$$\begin{aligned}&\mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))\\ &\qquad = \mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J} (Y_j = 0)) + \mathbb{P}(\cap_{i\in I\cup\{a\}}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))\end{aligned}$$

By induction hypothesis, we have:

$$\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0)) = \alpha\mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))$$
$$\mathbb{P}(\cap_{i\in I\cup\{a\}}(X_i = 1) \cap_{j\in J\setminus\{a\}} (X_j = 0)) = \alpha\mathbb{P}(\cap_{i\in I\cup\{a\}}(Y_i = 1) \cap_{j\in J\setminus\{a\}} (Y_j = 0))$$

Hence:

$$\mathbb{P}(\cap_{i\in I}(X_i = 1) \cap_{j\in J} (X_j = 0)) = \alpha\mathbb{P}(\cap_{i\in I}(Y_i = 1) \cap_{j\in J} (Y_j = 0))$$

We now show that for any $x \in \mathbb{N}$, we have that $\mathbb{P}\left(\sum_{i\in[m]} X_i \geq x\right) \geq \mathbb{P}\left(\sum_{i\in[m]} Y_i \geq x\right)$. Indeed, we have:

$$\begin{aligned}
\mathbb{P}\left(\sum_{i\in[m]} X_i \geq x\right) &= \sum_{|I|=x:r\in I} \mathbb{P}\left(\cap_{i\in I}(X_i = 1) \cap_{j\in[m]\setminus I} (X_j = 0)\right) \\
&+ \sum_{I\subset[m]:r\notin I,|I|\geq x} \mathbb{P}\left(\cap_{i\in I}(X_i = 1) \cap_{j\in[m]\setminus I} (X_j = 0)\right) \\
&+ \mathbb{P}\left(\cap_{i\in I\cup\{r\}}(X_i = 1) \cap_{j\in[m]\setminus(I\cup\{r\})} (X_j = 0)\right) \\
&= \alpha \sum_{|I|=x:r\in I} \mathbb{P}\left(\cap_{i\in I}(Y_i = 1) \cap_{j\in[m]\setminus I} (Y_j = 0)\right) \\
&+ \sum_{I\subset[m]:r\notin I,|I|\geq x} \mathbb{P}\left(\cap_{i\in I}(X_i = 1) \cap_{j\in[m]\setminus(I\cup\{r\})} (X_j = 0)\right) \\
&\geq \sum_{|I|=x:r\in I} \mathbb{P}\left(\cap_{i\in I}(Y_i = 1) \cap_{j\in[m]\setminus I} (Y_j = 0)\right) \\
&+ \sum_{I\subset[m]:r\notin I,|I|\geq x} \mathbb{P}\left(\cap_{i\in I}(Y_i = 1) \cap_{j\in[m]\setminus(I\cup\{r\})} (Y_j = 0)\right) \\
&= \mathbb{P}\left(\sum_{i\in[m]} Y_i \geq x\right)
\end{aligned}$$

□

Lemma 4.9.25. *Let X be a random variable that has a binomial distribution of parameters (m, p). Then if $0 < p \leq \frac{1}{m}$, we have that $\mathbb{P}(X \geq mp) \geq \frac{1}{4}$ as long as $p \geq 1 - \left(\frac{3}{4}\right)^{\frac{1}{m}}$.*

In particular, it suffices for p to be larger than $\frac{1}{3m}$.

Proof. If $p \leq \frac{1}{m}$ then $0 < mp \leq 1$ which means that the events $X \geq mp$ and $X \geq 1$ are the same since the binomial distribution takes only integer values. Hence:

$$\begin{aligned}
\mathbb{P}(X \geq mp) = \mathbb{P}(X \geq 1) &= 1 - \mathbb{P}(X = 0) = 1 - (1 - p)^m \\
&\geq 1 - \left(1 - 1 + \left(\frac{3}{4}\right)^{\frac{1}{m}}\right)^m \geq \frac{1}{4}
\end{aligned}$$

As the function $\frac{1}{3m} - 1 + \left(\frac{3}{4}\right)^{\frac{1}{m}}$ is positive for $m = 1$ and strictly decreasing towards 0 with increasing m, it is always positive and thus the second claim holds. □

Definition 4.9.26 (Mutually independent events)**.** *Let $A_1, \ldots, A_n$ be events. They are said to be mutually independent if and only if, for every subset $I \subseteq [n]$, we have that:*

$$\mathbb{P}\left(\cap_{i\in I} A_i\right) = \prod_{i\in I} \mathbb{P}(A_i)$$

Lemma 4.9.27. *If $A_1, \ldots, A_n$ are mutually independent events, then we have that, for every subsets $I, J \subseteq [n]$, $I \cap J = \varnothing$:*

$$\mathbb{P}\left(\cap_{i\in I} A_i \cap_{j\in J} \overline{A_j}\right) = \prod_{i\in I} \mathbb{P}(A_i) \prod_{j\in J} (1 - \mathbb{P}(A_j))$$

Proof. We will show by induction on the size of J that this holds for every $I \subseteq [n]$ such that $I \cap J = \varnothing$. It is clear for the case $|J| = 0$.

Let J be a nonempty subset of $[n]$ and I a subset of $[n]$ such that $I \cap J = \varnothing$. Let $a \in J$. Then we have that, by the induction hypothesis:

$$\mathbb{P}\left(\cap_{i\in I} A_i \cap_{j\in J\setminus\{a\}} \overline{A_j}\right) = \prod_{i\in I} \mathbb{P}(A_i) \prod_{j\in J\setminus\{a\}} (1 - \mathbb{P}(A_j))$$

However, we also have that:

$$\mathbb{P}\left(\cap_{i\in I} A_i \cap_{j\in J\setminus\{a\}} \overline{A_j}\right) = \mathbb{P}\left(\cap_{i\in I} A_i \cap_{j\in J} \overline{A_j}\right) + \mathbb{P}\left(\cap_{i\in I\cup\{a\}} A_i \cap_{j\in J\setminus\{a\}} \overline{A_j}\right)$$

Again, by the induction hypothesis, we have that

$$\mathbb{P}\left(\cap_{i\in I\cup\{a\}} A_i \cap_{j\in J\setminus\{a\}} \overline{A_j}\right) = \prod_{i\in I\cup\{a\}} \mathbb{P}(A_i) \prod_{j\in J\setminus\{a\}} (1 - \mathbb{P}(A_j))$$

Piecing everything together, we get that:

$$\begin{aligned}
\mathbb{P}\left(\cap_{i\in I} A_i \cap_{j\in J} \overline{A_j}\right) &= \mathbb{P}\left(\cap_{i\in I} A_i \cap_{j\in J\setminus\{a\}} \overline{A_j}\right) - \mathbb{P}\left(\cap_{i\in I\cup\{a\}} A_i \cap_{j\in J\setminus\{a\}} \overline{A_j}\right) \\
&= \prod_{i\in I} \mathbb{P}(A_i) \prod_{j\in J\setminus\{a\}} (1 - \mathbb{P}(A_j)) - \prod_{i\in I\cup\{a\}} \mathbb{P}(A_i) \prod_{j\in J\setminus\{a\}} (1 - \mathbb{P}(A_j)) \\
&= \prod_{i\in I} \mathbb{P}(A_i) \prod_{j\in J} (1 - \mathbb{P}(A_j))
\end{aligned}$$

□

Lemma 4.9.28 (Hoeffding's inequality for binomial distributions [85])**.** *Let Y be a binomial random variable with parameters (t, p). We then have, for any $x \leq tp$:*

$$\mathbb{P}(Y \leq x) \leq \exp\left(-2t\left(p - \frac{x}{t}\right)^2\right)$$

4.9.4 Ommitted Lemmas and Proofs from Section 4.4

Lemma 4.9.29. *For every $t \in \mathbb{N}$, we have that $n \geq N_t \geq X_t$.*

Proof. Note that N_t cannot go higher than n because it is the number of nodes informed after round t, which is at most n.

We will prove the rest by induction on t. For the induction basis note that by definition $N_0 = 1 = X_0$. For the induction step let us assume that $n \geq N_t \geq X_t$ for some $t \in \mathbb{N}$. Consider first the case that $N_{t+1} - N_t < (n - N_t) \cdot \frac{N_t}{n}$. Since no informed node can become uninformed, we have that $N_{t+1} \geq N_t \geq X_t = X_{t+1}$, as desired. Next consider the case that $N_{t+1} - N_t \geq (n - N_t) \cdot \frac{N_t}{n}$. Then $N_{t+1} \geq N_t + (n - N_t) \cdot \frac{N_t}{n}$ and $X_{t+1} = X_t + (n - X_t) \cdot \frac{X_t}{n}$. As the function $x \mapsto x + (n - x)\frac{x}{n}$ is strictly increasing for $x \leq n$, this proves that $N_{t+1} \geq X_{t+1}$, as desired. □

Lemma 4.9.30. *For every $t \in \mathbb{N}$, we have that $X_t \geq 1$.*

Proof. We will again show this by induction. For the induction basis note that by definition $1 = X_0$. For the induction step let us assume that $X_t \geq 1$ for some $t \in \mathbb{N}$. We then have two cases, either $X_{t+1} = X_t$ and the result holds trivially, or $X_{t+1} = X_t + (n - X_t) \cdot \frac{X_t}{n}$. Since $1 \leq X_t \leq n$, we have that $X_{t+1} \geq X_t \geq 1$. □

Lemma 4.9.31. *For every $t \in \mathbb{N}$, we have that $n > X_t$, if $n > 1$.*

Proof. We show this claim by induction on t. As $n > 1$ and $X_1 = 1$, it is trivially true for $t = 1$. Assume it is true for $t \in \mathbb{N}$. Then $X_{t+1} \leq X_t + (n - X_t) \cdot \frac{X_t}{n} = n(\frac{X_t}{n} + \frac{n - X_t}{n}\frac{X_t}{n}) < n$, where the last inequality holds by noting that $(\frac{X_t}{n} + \frac{n - X_t}{n}\frac{X_t}{n})$ is a convex combination of 1 and $\frac{X_t}{n}$, the latter of which being strictly smaller than 1. □

Essentially, this means that X_t never reaches n, and thus that X_{t+1} is always strictly larger than X_t if $N_{t+1} - N_t \geq (n - N_t) \cdot \frac{N_t}{n}$:

Corollary 4.9.32. *We have that $X_{t+1} > X_t$ if and only if $N_{t+1} - N_t \geq (n - N_t) \cdot \frac{N_t}{n}$.*

Lemma 4.4.3. *Let $u_t \in \mathbb{N}$ be the t-th round such that $X_{u_t+1} > X_{u_t}$ and let $u_0 = 0$. Then $X_{u_t} = n - n\left(\frac{n-1}{n}\right)^{2^t}$. Moreover, we have that $X_{u_{t+1}} = X_{u_t} + (n - X_{u_t}) \cdot \frac{X_{u_t}}{n}$.*

Proof. We show the claim by induction on t. By definition of u_0 we have that $X_{u_0} = 1$. Thus the induction basis $X_{u_0} = 1 = n - n\left(\frac{n-1}{n}\right)^{2^0}$ follows.

For the induction step assume next the result is true for some $t \in \mathbb{N}$. Note that for every $t \in \mathbb{N}$, it holds that $X_{u_t+1} = X_{u_t} + (n - X_{u_t}) \cdot \frac{X_{u_t}}{n}$. Indeed, we have that $X_{u_t+1} = X_{u_{t+1}-1} = \cdots = X_{u_t+1} = X_{u_t} + (n - X_{u_t}) \cdot \frac{X_{u_t}}{n}$. Thus,

$$\begin{aligned}
X_{u_{t+1}} &= X_{u_t} + (n - X_{u_t}) \cdot \frac{X_{u_t}}{n} = n - n\left(\frac{n-1}{n}\right)^{2^t} \\
&\quad + \left(n - n + n\left(\frac{n-1}{n}\right)^{2^t}\right) \frac{n - n\left(\frac{n-1}{n}\right)^{2^t}}{n} \\
&= n - n\left(\frac{n-1}{n}\right)^{2^t} + \left(\frac{n-1}{n}\right)^{2^t}\left(n - n\left(\frac{n-1}{n}\right)^{2^t}\right) \\
&= n - n\left(\frac{n-1}{n}\right)^{2^{t+1}}
\end{aligned}$$

□

Lemma 4.4.4. *If $t \geq u_{2\ln n}$, then $N_t = n$.*

Proof. Let $Y_{u_i} = \frac{n - X_{u_i}}{n}$. Then $X_{u_t} > n-1$ iff $Y_{u_t} < 1/n$. Further $Y_{u_0} = \frac{n-1}{n}$ and $Y_{u_{t+1}} = \frac{n - X_{u_{t+1}}}{n} = \left(\frac{n-1}{n}\right)^{2^{t+1}} = (Y_{u_t})^2$. Now note that $Y_{u_t} < 1/n$ iff $\left(\frac{n-1}{n}\right)^{2^t} < 1/n$ iff $2^t < \frac{\log(1/n)}{\log(\frac{n-1}{n})} = \frac{\log n}{\log n - \log(n-1)}$. Now note that $\log n - \log(n-1) > 1/n$ and, thus, $\frac{\log n}{\log n - \log(n-1)} < n \log n \leq n^2$. It follows that for $t \leq 2\log(n)$ it holds that $X_{u_t} > n-1$ and, hence, $N_{u_t} \geq X_{u_t} > n-1$. As N_{u_t} is an integer, the result follows. □

Lemma 4.4.6. *If $n > 4$, for every $t \in \mathbb{N}$, we have that $\mathbb{P}(X_{t+1} > X_t) \geq \frac{1}{4}$.*

Proof. By Corollary 4.9.38, we have that $X_{t+1} > X_t$ if and only if $N_{t+1} - N_t \geq (n - N_t) \cdot \frac{N_t}{n}$. Thus, $\mathbb{P}(X_{t+1} > X_t) = \sum_{\ell \in [n]} \mathbb{P}\left(N_{t+1} - N_t \geq (n - N_t) \cdot \frac{N_t}{n} \middle| N_t = \ell\right) \mathbb{P}(N_t = \ell)$. Hence we only have to show that $\mathbb{P}\left(N_{t+1} - N_t \geq (n - N_t) \cdot \frac{N_t}{n} \middle| N_t = \ell\right) \geq \frac{1}{4}$ for every $\ell = N_t$. Recall that $N_{t+1} - N_t$ follows a binomial distribution with parameters $m = n - N_t$ and $p = \frac{N_t}{n}$.

If $N_t = n$, the result holds trivially. Otherwise, if $pm \geq 1$, then the result holds by applying Theorem 4.4.5. If on the other hand $pm \leq 1$, then we note that $pm = \frac{(n - N_t)N_t}{n}$ which is at its minimum for $N_t = 1$. Therefore $pm \geq \frac{n-1}{n} \geq \frac{1}{3}$ if $n \geq 2$. The result then holds by applying Lemma 4.9.25. □

Theorem 4.4.9. *For any $c \geq 1$ and $n \geq 5$, All-to-All Broadcast on Uniformly Random Trees completes within $32 \cdot c \cdot \ln n$ rounds with probability $p > 1 - \frac{1}{n^{c-1}}$.*

Proof. Let $N_t^{(i)}$ be the random variable that represents the number of nodes that are informed after round t of the message given to node i. By Theorem 4.2.1, we know that $\mathbb{P}\left(N^{(i)}_{32 \cdot c \cdot \ln n} < n\right) \leq n^{-c}$ for every $i \in [n]$. Using a union-bound, we get that:

$$\mathbb{P}\left(\bigcup_{i \in [n]} N^{(i)}_{32 \cdot c \cdot \ln n} < n\right) \leq n^{-c+1}$$

And thus:

$$\mathbb{P}\left(\bigcap_{i \in [n]} N^{(i)}_{32 \cdot c \cdot \ln n} = n\right) = 1 - \mathbb{P}\left(\bigcup_{i \in [n]} N^{(i)}_{32 \cdot c \cdot \ln n} < n\right) \geq 1 - n^{-c+1}$$

□

4.9.5 Adversarial Nodes: Trees with Byzantine Nodes

In this section, we will discuss the case where some nodes are *Byzantine*, that is, nodes that can arbitrarily deviate from the protocol. These nodes can stop functioning, send wrong messages, and coordinate to make the protocol fail. We will rely on cryptographic tool so that each node can sign and encrypt the message it sends. Then nodes can be confident about the sender of each message and its content and can forward the message along with its unchanged signature to other nodes. We will assume that there are up to f Byzantine nodes, out of a total of n nodes. We require that $f \leq \frac{2}{3}n - 1$. Nodes that are not Byzantine are called honest.

We begin by analyzing Broadcast in this setting. We first give a message to a fixed honest node, and ask the node to forward it to all other honest nodes. Note the difference between this model and the reliable Broadcast model, where the initial message could be from an honest node or a Byzantine node, and where if the initial message is from a Byzantine node, then the message accepted by each honest node must be the same.

In our setting, the best strategy for the Byzantine node is not to forward any message at all. Indeed, they cannot modify the content of a message because they cannot forge any signature, and, thus, their power is limited. Hence, we will analyze this problem as if Byzantine nodes are just defunct but the process that chooses the communication network, i.e., the random tree, does not know which nodes are Byzantine and, thus, they are part of the network as before, i.e., the tree still consists of n nodes.

As in the previous section, I_t and S_t will, respectively, be the set of informed and uninformed nodes after round t. We set $I_0 = \{v\}$ and $S_0 = [n - f] - \{v\}$, where v is the node that initially holds the message, $N_t = |I_t|$ to be the number of informed and honest nodes after t rounds, and T_t to be the tree chosen at random in round t. For a tree T, for each node p, $P_T(p)$ is the (unique) parent of node p in T, unless p is the root of T, in which case $P_T(p) = p$. Simplifying the notation, we also use $P_t(p)$ to denote $P_{T_t}(p)$. We use $A(S, x)$ where S is a set and x an integer to represent the set of subsets of S of size x.

For the rest of the paper, we will denote $\tau = \frac{\log n}{\log\left(1+\frac{n-f}{2n}\right)}$. Note that $\log n \leq \tau \leq 4.5 \log n$.

Again, the central lemma will characterize how many new nodes get informed in each round, depending on how many were informed after the previous round. This lemma shows that uninformed nodes get informed independently from each other.

Lemma 4.9.33. *For any $t > 0$, $N_{t+1} - N_t$ follows a binomial distribution with parameters $\left(n - f - N_t, \frac{N_t}{n}\right)$.*

This lemma proves that every uninformed node has probability $\frac{N_t}{n}$ of having an informed parent in round $t + 1$, regardless of whether the other uninformed nodes have an uninformed parent.

Proof. Let $I_t = \{i_1, \ldots, i_{N_t}\}$ and $S_t = \{s_1, \ldots, s_{n-f-N_t}\}$. We then have, for any integer x:

$$\mathbb{P}(N_{t+1} - N_t = x | \mathcal{F}_t) = \sum_{J \in A(S_t, x)} \mathbb{P}\left(\bigcap_{y \in J} (P_{t+1}(y) \in I_t) \bigcap_{y \in S_t \setminus J} (P_{t+1}(y) \notin I_t) \middle| \mathcal{F}_t \right)$$

Our goal is to show that the events $P_t(y) \in I_t$ for different $y \in S_t$ are mutually independent. Let us look at the event $\bigcap_{y \in J}(P_t(y) \in I_t)$ for any $J \subseteq S_t$ (note that we do not require that J has a specific size here). We can then write, indexing a on J:

$$\begin{aligned}\mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(y) \in I_t)\middle|\mathcal{F}_t\right) &= \sum_{a\in[N_t]^{|J|}} \mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(y) = i_{a_y})\middle|\mathcal{F}_t\right) \\ &= \sum_{a\in[N_t]^{|J|}} \frac{\left|\{T \in \mathcal{T}_n : P_T(y) = i_{a_y}, \forall y \in J\}\right|}{|\mathcal{T}_n|}\end{aligned}$$

Now consider the forest that is composed of stars whose centers are the i_{a_y} and whose leaves are the nodes y. More specifically, consider the forest that contains the edges $(i_{a_y}, y), \forall y \in J$. Note that $\left|\{T \in \mathcal{T}_n : P_T(y) = i_{a_y}, \forall y \in J\}\right|$ equals the number of rooted trees that are compatible with this forest. By Theorem 4.9.1, we have that $\left|\{T \in \mathcal{T}_n : P_T(y) = i_{a_y}, \forall y \in J\}\right| = n^{n-1-|J|}$. This allows us to compute the above probability as follows:

$$\mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(y) \in I_t)\middle|\mathcal{F}_t\right) = \sum_{a\in[N_t]^{|J|}} \frac{n^{n-1-|J|}}{n^{n-1}} = \left(\frac{N_t}{n}\right)^{|J|}$$

This proves that the events $P_{t+1}(y) \in I_t$ for any two $y \in S_t$ are mutually independent (Definition 4.9.26), each having probability $\frac{N_t}{n}$. Going back to the first equation of this proof, we can now compute, using Lemma 4.9.27:

$$\begin{aligned}\mathbb{P}(N_{t+1} - N_t = x|\mathcal{F}_t) &= \sum_{J\in A(S_t,x)} \prod_{y\in J} \mathbb{P}\left(P_{t+1}(y) \in I_t|\mathcal{F}_t\right) \prod_{y\in S_t\setminus J} \mathbb{P}\left(P_{t+1}(y) \notin I_t)|\mathcal{F}_t\right) \\ &= \binom{n-f-N_t}{x}\left(\frac{N_t}{n}\right)^x\left(1-\frac{N_t}{n}\right)^{n-f-N_t-x}\end{aligned}$$

□

Our next goal is to show that $N_t = n - f$ with high probability for all $t \geq 32 \cdot \tau \cdot c$, for any $c \geq 1$, and $\tau = \frac{\log n}{\log(1+\frac{n-f}{2n})}$. To do so we introduce a random variable X_t that we use to lower bound N_t and we show right below that $X_t \leq N_t$ for all t.

Definition 4.9.34 (Lower bound random variable for the number of informed nodes with byzantine nodes)**.** *Let X_t be the random variable that is defined as follows:*

$$\begin{aligned}X_0 &= 1 \\ X_{t+1} &= X_t + (n-f-X_t)\cdot\frac{X_t}{n} && \text{if} \quad N_{t+1} - N_t \geq (n-f-N_t)\cdot\frac{N_t}{n} \\ X_{t+1} &= X_t && \text{if} \quad N_{t+1} - N_t < (n-f-N_t)\cdot\frac{N_t}{n}\end{aligned}$$

Lemma 4.9.35. *For every $t \in \mathbb{N}$, we have that $n - f \geq N_t \geq X_t$.*

Proof. Note that N_t cannot go higher than $n - f$ because it is the number of honest nodes informed after round t, which is at most $n - f$.

We will prove the rest by induction on t. For the induction basis note that by definition $N_0 = 1 = X_0$. For the induction step let us assume that $n - f \geq N_t \geq X_t$ for some $t \in \mathbb{N}$. Consider first the case that $N_{t+1} - N_t < (n - f - N_t) \cdot \frac{N_t}{n}$. Since no informed node can become uninformed, we have that $N_{t+1} \geq N_t \geq X_t = X_{t+1}$, as desired. Next consider the case that $N_{t+1} - N_t \geq (n - f - N_t) \cdot \frac{N_t}{n}$. Then $N_{t+1} \geq N_t + (n - f - N_t) \cdot \frac{N_t}{n}$ and $X_{t+1} = X_t + (n - f - X_t) \cdot \frac{X_t}{n}$. As the function $x \mapsto x + (n - f - x)\frac{x}{n}$ is strictly increasing for $x \leq n - f$, this proves that $N_{t+1} \geq X_{t+1}$, as desired. □

Lemma 4.9.36. *For every $t \in \mathbb{N}$, we have that $X_t \geq 1$.*

Proof. We will again show this by induction. For the induction basis note that by definition $1 = X_0$. For the induction step let us assume that $X_t \geq 1$ for some $t \in \mathbb{N}$. We then have two cases, either $X_{t+1} = X_t$ and the result holds trivially, or $X_{t+1} = X_t + (n - f - X_t) \cdot \frac{X_t}{n}$. Since $1 \leq X_t \leq n - f$, we have that $X_{t+1} \geq X_t \geq 1$. □

Next we show that X_t never reaches $n - f$ if there are at least 2 honest nodes.

Lemma 4.9.37. *For every $t \in \mathbb{N}$, we have that $n - f > X_t$, if $n - f > 1$.*

Proof. We show this claim by induction on t. As $n - f > 1$ and $X_1 = 1$, it is trivially true for $t = 1$. Assume it is true for $t \in \mathbb{N}$. Then $X_{t+1} \leq X_t + (n - f - X_t) \cdot \frac{X_t}{n} = (n - f)(\frac{X_t}{n-f} + \frac{n-f-X_t}{n-f}\frac{X_t}{n}) < n - f$, where the last inequality holds by noting that $(\frac{X_t}{n-f} + \frac{n-f-X_t}{n-f}\frac{X_t}{n})$ is a convex combination of 1 and $\frac{X_t}{n}$, the latter of which being strictly smaller than 1. □

It follows that X_{t+1} is always strictly larger than X_t if $N_{t+1} - N_t \geq (n - f - N_t) \cdot \frac{N_t}{n}$:

Corollary 4.9.38. *We have that $X_{t+1} > X_t$ if and only if $N_{t+1} - N_t \geq (n - f - N_t) \cdot \frac{N_t}{n}$.*

Next we introduce a variable that remembers the rounds that increase X_t.

Definition 4.9.39 (Strict improvements)**.** *Let $u_t \in \mathbb{N}$ be the t-th round such that $X_{u_t+1} > X_{u_t}$ and let $u_0 = 0$.*

We now show that once X_t has been increased in $\Theta(\log n)$ rounds, it reaches $n - f$, i.e., all honest nodes have received the message. Recall that $\tau = \frac{\log n}{\log\left(1+\frac{n-f}{2n}\right)}$.

Lemma 4.9.40. *For all $t \geq u_{2\tau}$, $N_t = n - f$.*

Proof. Since N_t is non-decreasing and upper-bounded by $n - f$, it suffices to show that $N_{u_{2\tau}} = n - f$. We will do so by using its lower bound $X_{u_{2\tau}}$.

We first show that $X_{u_\tau} \geq \frac{n-f}{2}$. Indeed, for any t, if $X_{u_t} \leq \frac{n-f}{2}$, then :

$$X_{u_{t+1}} = X_{u_t} + (n - f - X_{u_t})\frac{X_{u_t}}{n} = X_{u_t}\left(1 + \frac{n - f - X_{u_t}}{n}\right) \geq X_{u_t}\left(1 + \frac{n-f}{2n}\right)$$

And X_{u_t} thus only needs at most τ steps to increase from 1 to $\frac{n-f}{2}$.

We now show that $X_{u_{2\tau}} > n - f - 1$. Indeed, for any t, if $X_{u_t} \geq \frac{n-f}{2}$, then we have:

$$\begin{aligned} n - f - X_{u_{t+1}} &= n - f - X_{u_t} - (n - f - X_{u_t})\frac{X_{u_t}}{n} \\ &= (n - f - X_{u_t})\left(1 - \frac{X_{u_t}}{n}\right) \leq (n - f - X_{u_t})\left(1 - \frac{n-f}{2n}\right) \end{aligned}$$

And $n - f - X_{u_t}$ thus only needs at most τ' steps to decrease from $\frac{n-f}{2}$ to $\frac{1}{2}$, where:

$$\tau' = \frac{\log(n-f)}{\log\left(\frac{1}{1-\frac{n-f}{2n}}\right)} = \frac{\log(n-f)}{\log\left(\frac{2n}{n+f}\right)} \leq \frac{\log(n-f)}{\log\left(\frac{3n-f}{2n}\right)} = \tau$$

where the inequality holds since $\frac{3n-f}{2n} = \frac{2n+n-f}{n+f+n-f} \leq \frac{2n}{n+f}$.

Overall we have that $X_{u_{2\tau}} \geq X_{u_{\tau+\tau'}} > n-f-1$. Since $n-f \geq N_{u_{2\tau}} \geq X_{u_{2\tau}}$ by Lemma 4.9.35, and since $N_{u_{2\tau}} \in \mathbb{N}$, we have that $N_{u_{2\tau}} = n - f$. □

Lemma 4.9.41. *If $f \leq \frac{2}{3}n - 1$, for every $t \in \mathbb{N}$, we have that $\mathbb{P}(X_{t+1} > X_t) \geq \frac{1}{4}$*

Proof. By Corollary 4.9.38, we have that $X_{t+1} > X_t$ if and only if $N_{t+1} - N_t \geq (n-f-N_t)\cdot\frac{N_t}{n}$. Thus, $\mathbb{P}(X_{t+1} > X_t) = \sum_{\ell\in[n]} \mathbb{P}\left(N_{t+1} - N_t \geq (n - f - N_t)\cdot\frac{N_t}{n} | N_t = \ell\right)\mathbb{P}(N_t = \ell)$. It thus suffices to show that for every ℓ, $\mathbb{P}\left(N_{t+1} - N_t \geq (n - f - N_t)\cdot\frac{N_t}{n} | N_t = \ell\right) \geq \frac{1}{4}$. Recall that $N_{t+1} - N_t$ follows a binomial distribution with parameters $m = n - f - N_t$ and $p = \frac{N_t}{n}$.

If $N_t = n - f$, the result holds trivially. Otherwise, if $pm \geq 1$, then the result holds by applying Theorem 4.4.5. If on the other hand $pm \leq 1$, then we note that $pm = \frac{(n-f-N_t)N_t}{n}$ which is at its minimum for $N_t = 1$. Therefore $pm \geq \frac{n-f-1}{n} \geq \frac{1/3n}{n} = \frac{1}{3}$, where we used $f \leq \frac{2}{3}n - 1$. The result then holds by applying Lemma 4.9.25.

□

Let $(B_t)_{t\in\mathbb{N}}$ be Bernoulli independent random variables of parameter $\frac{1}{4}$. Let $Z^B_{\leq t} = \sum_{z\in[t]} B_z$ and $Z_{\leq t} = \sum_{z\in[t]} \mathbb{1}(X_{z+1} > X_z)$.

Corollary 4.9.42. *For any $\ell \in \mathbb{N}$, we have that $\mathbb{P}(Z_{\leq t} \leq \ell) \leq \mathbb{P}(Z^B_{\leq t} \leq \ell)$.*

Using the corollary we can now show that with high probability in the first $32 \cdot c \cdot \tau$ X_t will increase at least 2τ often. This then immediately leads to our result.

Lemma 4.9.43. *Let $t = 32 \cdot c \cdot \tau$ for any $c \geq 1$. Then $\mathbb{P}(Z_{\leq t} \leq 2\tau) \leq \frac{1}{n^c}$.*

Proof. Note that $Z^B_{\leq t}$ is a binomial distribution of parameters $(t, \frac{1}{4})$. Using Hoeffding's inequality, we have that:

$$\mathbb{P}(Z^B_{\leq t} \leq 2\tau) \leq \exp\left(-2t\left(\frac{1}{4} - \frac{2\tau}{t}\right)^2\right) \leq \exp\left(-2 \cdot 32c \ln n \left(\frac{1}{4} - \frac{2}{16}\right)^2\right) = n^{-c}$$

Where we used that $\tau \leq \ln n$. Corollary 4.9.42 then gives the desired result. □

We now have all the tools to prove the main theorem of this section, which we state here:

Theorem 4.5.1. *For any $c \geq 1$, and $f \leq \frac{2}{3}n - 1$, Broadcast on Uniformly Random Trees with f Byzantine nodes completes within $144 \cdot c \cdot \log n$ rounds with probability $p > 1 - \frac{1}{n^c}$.*

Proof. By Lemma 4.9.43, we have that, with probability $p \leq 1 - \frac{1}{n^c}$, $X_{t+1} > X_t$ for at least 2τ many rounds within the $32 \cdot c \cdot \tau$ first rounds. Recall that $u_{2\tau}$ is the 2τ-th round where $X_{t+1} > X_t$. We thus have that $\mathbb{P}(Z_{\leq 32 \cdot c \cdot \tau} > 2\tau) = \mathbb{P}\left(u_{2\tau} \leq 32 \cdot c \cdot \tau\right) \geq 1 - n^{-c}$. But, by Lemma 4.9.40 the event $u_{2\tau} \leq 32 \cdot c \cdot \tau$ implies the event $N_{32 \cdot c \cdot \tau} = n - f$, therefore $\mathbb{P}\left(N_{32 \cdot c \cdot \tau} = n - f\right) \geq 1 - n^{-c}$. Upper-bounding $\tau \leq 4.5 \log n$ gives the desired result. □

We now use this result to get a similar result for All-to-all Broadcast. Using a union-bound, we obtain:

Theorem 4.5.2. *For any $c \geq 1$, and $f \leq \frac{2}{3}n - 1$, All-to-all Broadcast on Uniformly Random Trees with f Byzantine nodes completes within $144 \cdot c \cdot \log n$ rounds with probability $p > 1 - n^{1-c}$.*

Proof. Let $N^{(i)}_t$ be the random variable that represents the number of nodes that are informed after round t of the message given to node i. By Theorem 4.5.1, we know that $\mathbb{P}\left(N^{(i)}_{32 \cdot c \cdot \tau} < n - f\right) \leq n^{-c}$ for every $i \in [n]$. Using a union-bound, we get that:

$$\mathbb{P}\left(\bigcup_{i \in [n]} N^{(i)}_{32 \cdot c \cdot \tau} < n - f\right) \leq n^{-c+1}$$

And thus:

$$\mathbb{P}\left(\bigcap_{i \in [n]} N^{(i)}_{32 \cdot c \cdot \tau} = n - f\right) = 1 - \mathbb{P}\left(\bigcup_{i \in [n]} N^{(i)}_{32 \cdot c \cdot \tau} < n - f\right) \geq 1 - n^{-c+1}$$

□

This allows us, e.g., to implement algorithms that run on a clique in a synchronous setting in our sparser graph. Indeed, each round of communication of a clique can be simulated by $32 \cdot c \cdot \tau$ rounds of Uniformly Random Trees with high probability, since All-to-All Broadcast needs $32 \cdot c \cdot \tau$ rounds to complete with high probability. Essentially, if an algorithm runs in T time, with $T \leq n^{c-1}$, in a clique network, we can implement it with high probability in $32 \cdot c \cdot T \cdot \tau$ rounds in the Uniformly Random trees network, which is essentially a logarithmic overhead. The only caveat is that if T is too large, i.e. $T > n^{c-1}$, the probability of at least one of the T All-to-All Broadcast rounds failing can become close to 1. To circumvent this, we restrict ourselves to the case where T is a small enough polynomial in n.

Theorem 4.2.3. *Let $\mathcal{A}$ be a distributed synchronous algorithm that runs on a static clique in T rounds, where $T \leq \alpha n^x$ for some constant $\alpha, x \in \mathcal{R}_+$, and has a probability of success p. Assume $\mathcal{A}$ is robust to f Byzantine nodes, and $f \leq \frac{2}{3}n - 1$. Then, assuming standard cryptographic tools[7], there exists a distributed algorithm $\mathcal{A}'$ that runs on Uniformly Random Trees in $T \cdot 144 \cdot \log n \cdot c$ rounds, and has a probability of success $p' \geq p(1 - \alpha n^{1+x-c})$, for any $c \geq 1 + x$. Moreover, $\mathcal{A}'$ is robust to f Byzantine nodes.*

Proof. The algorithm $\mathcal{A}'$ works as follows: Each round of $\mathcal{A}$ will be simulated using $32 \cdot c \cdot \tau$ rounds of Uniformly Random Trees. Each round in $\mathcal{A}$ can be seen as (1) a *computation step*, where each node decides what message to send to every other node, and (2) a *communication step*, where each node sends the messages and receives the messages the other nodes have sent it. In $\mathcal{A}'$ the computation step is unchanged, except that each node uses the PKI to sign and encrypt the messages. The communication step on the other hand is extended over $32 \cdot c \cdot \tau$ rounds, in which every (honest) node simply forwards all received messages to all its out-neighbors. Theorem 4.5.2 ensures that with probability at most n^{1-c}, at least one honest node fails to send a message to all other honest nodes. By a union bound over the T rounds, the probability that at least one message fails to be delivered is at most αn^{1+x-c}. By taking its complement, the probability that all messages are delivered in time before the next round's computation step is $p' \geq 1 - \alpha n^{x+1-c}$. □

We now give two applications of this theorem, namely Reliable Broadcast and Byzantine Consensus.

Corollary 4.2.4. *For any $c \geq 1$, and $f \leq \frac{2}{3}n - 1$, in the Uniformly Random Trees with f Byzantine nodes, there exists an algorithm for Reliable Broadcast, that is robust to f Byzantine nodes, that runs in $(f + 1) \cdot 144 \cdot c \cdot \log n$ rounds, and succeeds with probability $p \geq 1 - n^{2-c}$.*

Proof. Dolev and Strong [50] have given an algorithm that solves reliable Broadcast, is robust to f Byzantine nodes, and runs in $T = f+1$ rounds. Since $T \leq n$, we can apply Theorem 4.2.3 with $x = 1, \alpha = 1$, and we get the desired result. □

Corollary 4.2.5. *For any $c \geq 1$ and $f < \frac{n}{3}$, in the Uniformly Random Trees with f Byzantine nodes, there exists an algorithm for Byzantine Consensus, that is robust to f Byzantine nodes, that runs in $3(f + 1) \cdot 144 \cdot c \cdot \log n$ rounds, and succeeds with probability $p \geq 1 - 2n^{2-c}$.*

Proof. Berman, Garay and Perry [22] have given an algorithm (known as the King's algorithm) that solves reliable Broadcast, is robust to f Byzantine nodes, and runs in $T = 3(f + 1)$ rounds. Since $T \leq 2n$, we can apply Theorem 4.2.3 with $x = 1, \alpha = 2$, and we get the desired result. □

4.9.6 Ommitted proofs of Section 4.6

Lemma 4.6.6 (Distribution Domination)**.** *Let t be a round. Let E_1, E_2 be two sets of edges the adversaries could choose for round t. Let $N_t^{(1)}$ (resp. $I_t^{(1)}$) be the number (resp. set) of*

[7] Specifically, our approach requires authenticated messages. Encryption may also be needed, only if the protocol $\mathcal{A}$ is vulnerable to eavesdropping. Both can be implemented using standard cryptographic tools.

informed nodes after round t if E_1 is chosen, and $N_t^{(2)}$ (resp. $I_t^{(2)}$) if E_2 is chosen. Then if $\mathbb{P}(N_t^{(1)} \geq m) \geq \mathbb{P}(N_t^{(2)} \geq m)$ for every $m \in \mathbb{N}$ (that is, if $N_t^{(1)}$ stochastically dominates $N_t^{(2)}$), then choosing E_2 is a better strategy for the adversary than choosing E_1.

Proof. Saying that $N_t^{(1)}$ stochastically dominates $N_t^{(2)}$ is equivalent to saying that $I_t^{(1)}$ stochastically dominates $I_t^{(2)}$. By Theorem 4.6.5, there exists a coupling $(\hat{I}_t^{(1)}, \hat{I}_t^{(2)})$ of $I_t^{(1)}$ and $I_t^{(2)}$ such that $\mathbb{P}\left(\left|\hat{I}_t^{(1)}\right| \geq \left|\hat{I}_t^{(2)}\right|\right) = 1$. Consider that coupling and let β be a bijection from $[n]$ to $[n]$ such that $\beta(\hat{I}_t^{(2)}) \subseteq \hat{I}_t^{(1)}$. By slight abuse of notation we naturally extend β to also give a bijection between edges, by setting $\beta(u,v) = (\beta(u), \beta(v))$. For any $t' \geq t$, let $E_1^{t'}$ and $E_2^{t'}$ be respectively the edges chosen by the adversary in round t' after choosing E_1, respectively E_2, in round t. Let $T_1^{t'}$ and $T_2^{t'}$ be respectively the trees in round t' containing $E_1^{t'}$ and $E_2^{t'}$, respectively.

We introduce a coupling $(\hat{T}_1^{t'}, \hat{T}_2^{t'})$ such that if $E_1^{t'} = \beta(E_2^{t'})$ then $\mathbb{P}(\hat{T}_1^{t'} = \beta(\hat{T}_2^{t'})) = 1$ as follows: If $E_1^{t'} = \beta(E_2^{t'})$, then note that β induces a bijection between $\mathcal{T}_n^{(2)} = \{T \in \mathcal{T}_n : E_2^{t'} \subseteq T\}$ and $\mathcal{T}_n^{(1)} = \{T \in \mathcal{T}_n : E_1^{t'} \subseteq T\}$. In this case to define the coupling we choose $\hat{T}_2^{t'}$ uniformly at random from $\mathcal{T}_n^{(2)}$ and set $\hat{T}_1^{t'} = \beta(\hat{T}_2^{t'})$. Thus, $\mathbb{P}(\hat{T}_1^{t'} = \beta(\hat{T}_2^{t'})) = 1$.

Otherwise, i.e., if $E_1^{t'} \neq \beta(E_2^{t'})$, to define the coupling we choose $\hat{T}_2^{t'}$ uniformly at random from $\mathcal{T}_n^{(2)}$ and, independently, $\hat{T}_1^{t'}$ uniformly at random from $\mathcal{T}_n^{(1)}$.

Let $\hat{I}_1^{t'}$ and $\hat{I}_2^{t'}$ be the set of informed nodes we get after round t' if the trees in round t' were $\hat{T}_1^{t'}$, respectively $\hat{T}_2^{t'}$. We will show that $\mathbb{P}\left(\beta(\hat{I}_2^{t'}) \subseteq \hat{I}_1^{t'}\right) = 1$ for all $t' \geq t$.

Let us now assume that the sequence $(E_1^{t'})_{t'>t}$ is an optimal strategy for the adversary after choosing E_1 in round t and let us call this sequence (including E_1 in round t) σ_1. Then consider the sequence σ_2 which chooses E_2 in round t then $E_2^{t'} := \beta^{-1}(E_1^{t'})$ in rounds $t' \geq t$ and compare it with σ_1. Thus, $E_1^{t'} = \beta(E_2^{t'})$ for all $t' > t$, which implies that $\mathbb{P}(\hat{T}_1^{t'} = \beta(\hat{T}_2^{t'})) = 1$.

Recall that σ_1 is optimal after choosing E_1 in round t. We show by induction that σ_2 is a better overall strategy for the adversary than σ_1. Indeed, we can show by induction on the number of rounds $t' \geq t$ that $\mathbb{P}\left(\beta(\hat{I}_2^{t'}) \subseteq \hat{I}_1^{t'}\right) = 1$. The induction basis is trivial as $\beta(\hat{I}_2^t) \subseteq \hat{I}_1^t$. For the induction step, note that for any $v \in \hat{I}_2^{t'}$, either (1) $v \in \hat{I}_2^{t'-1}$ which, using the induction assumption implies with probability 1 that $\beta(v) \in \beta(\hat{I}_2^{t'-1}) \subseteq \hat{I}_1^{t'-1} \subseteq \hat{I}_1^{t'}$, or (2) $P_{\hat{T}_2^{t'}}(v) \in \hat{I}_2^{t'-1}$.

As $\mathbb{P}(\hat{T}_1^{t'} = \beta(\hat{T}_2^{t'})) = 1$, Case (2) implies that with probability 1, $\beta(P_{\hat{T}_2^{t'}}(v)) = P_{\hat{T}_1^{t'}}(\beta(v))$. As $P_{\hat{T}_2^{t'}}(v) \in \hat{I}_2^{t'-1}$ and by induction, with probability 1, $\beta(\hat{I}_2^{t'-1}) \subseteq \hat{I}_1^{t'-1}$, it follows that with probability 1, it holds that $P_{\hat{T}_1^{t'}}(\beta(v)) = \beta(P_{\hat{T}_2^{t'}}(v)) \in \hat{I}_1^{t'-1}$ and, thus, $\beta(v) \in \hat{I}_1^{t'}$. Hence, in both cases, with probability 1, $\beta(I_2^{t'}) \in \hat{I}_1^{t'}$.

Now note that $\mathbb{P}\left(\beta(\hat{I}_2^{t'}) \subseteq \hat{I}_1^{t'}\right) = 1$ implies that if t' is the smallest round at which Broadcast completes after the adversary chooses E_2, that is, if $\left|\hat{I}_2^{t'-1}\right| = n$, then Broadcast completes in a no later round if the adversary chooses E_1. □

Lemma 4.6.10. *For any increasing tree U, there exists a correction U'.*

Proof. Let $V(U)$ be the set of nodes of U and let $|V(U) \cap S_{t-1}| = \ell$. To show the lemma we will give a bijection b that maps the ℓ uninformed nodes of $V(U)$ to the ℓ first nodes of U in bfs-order (on U) and the informed nodes to the remaining nodes of U such that one of the nodes $u \in S_{t-1}$ with $P_U(u) \in I_{t-1}$ becomes the root of U'. The resulting tree will be the correction U'. As a result of this bijection every uninformed node of U' has only uninformed ancestors and, thus, U' is non-increasing. By construction, U and U' are isomorphic.

More formally, if U is increasing, then there exists an edge (i, s) such that $i \in I_{t-1}, s \in S_{t-1}$. Let π be a bijection from $[|V(U)|]$ to $V(U)$ such that $\pi(1) = s, \{\pi(2), \ldots, \pi(\ell)\} \subset S_{t-1}$, and $\{\pi(\ell+1), \ldots, \pi(|V(U)|)\} \subseteq I_{t-1}$. Furthermore, let ρ be a bijection from $[|V(U)|]$ to $V(U)$ such that $\rho(j)$ is the j-th node encountered in a breadth-first traversal starting at the root of U. Then let $b = \pi \circ \rho^{-1}$. We will show next that the tree U', whose set of edges is $\{(b(u), b(v)) : (u, v) \in U\}$ is a correction of U. We first need to argue that U' is a tree. This is the case as U' is a graph over the same nodes as U and for every $(b(u), b(v)) \in U'$, u is encountered in a BFS before v in U, thus $\rho^{-1}(u) < \rho^{-1}(v)$. Now we are ready to show that U' fulfills the conditions of being a correction of U: (1) It follows that U' cannot contain a cycle. As U' is a graph on $|U|$ nodes with $|U| - 1$ edges, it follows that U' is a tree. Additionally, U' isomorphic to U as b gives the required bijection between U and U'.

(2) Furthermore, $\rho^{-1}(u) < \rho^{-1}(v)$ implies that if $\rho^{-1}(u) \geq \ell$, we also have $\rho^{-1}(v) \geq \ell$ for any ℓ. Specifically for $\ell = |S_{t-1}|$ this means that if $b(u) \in I_{t-1}$, then $b(v) \in I_{t-1}$. Thus U' is non-increasing in round t.

(3) By construction we made sure that $s \in S_{t-1}$ with $P_U(s) \in I_{t-1}$ is the root of U'. □

Lemma 4.6.11. *Let t be a round and N_{t-1} be the number of informed nodes after round $t-1$. Let E_1, E_2 be two sets of edges that the adversary could choose for round t such that*

1. *E_1 is a collection of rooted trees such that at least one tree U is information-increasing, and*
2. *E_2 is obtained from E_1 by replacing U with a correction U' of U.*

Let $N_t^{(1)}$ be the number of informed nodes after round t if E_1 is chosen, and let $N_t^{(2)}$ be that number if E_2 is chosen. Then choosing E_2 is a better strategy for the adversary than choosing E_1.

Proof. We will build a bijection π from $\mathcal{T}_n^{(2)} = \{T \in \mathcal{T}_n : E_2 \subseteq T\}$ to $\mathcal{T}_n^{(1)} = \{T \in \mathcal{T}_n : E_1 \subseteq T\}$ such that for every $s \in S_{t-1}$ and any $T \in \mathcal{T}_n^{(2)}$ with $P_T(s) \in I_{t-1}$, we have that $P_{\pi(T)}(s) \in I_{t-1}$. Hence, $\pi(T)$ has more uninformed nodes that become informed than T. We will use this property to show that $N_t^{(1)}$ stochastically dominates $N_t^{(2)}$.

To do so, let b be the bijection that achieves the isomorphism of the proof of Lemma 4.6.10 from U to U'. $\pi(T)$ is constructed in a way such that all nodes have the same parents as in T, unless they are in U'. More specifically, we let $\pi(T) := \pi_b(T)$ where $\pi_b(T)$ is the tree obtained from T by replacing every edge $(u, v) \in T$ as follows:

- if $u, v \in U'$, then replace it with the edge $(b^{-1}(u), b^{-1}(v))$.

- if $u \notin U'$, $v \notin U'$, then keep it the same.
- if $u \in U'$, $v \notin U'$, then keep it the same.
- if $u \notin U', v \in U'$, then replace it with $(u, b^{-1}(v))$.

We clearly have that $U \subseteq E_1 \subset \pi(T)$ and $U' \subseteq E_2 \subset T$. Also, for any node v, the path from the root to v in T can be transformed into a path from the root in $\pi(T)$ by replacing the subpath $P = u_0, \ldots, u_\ell$ that is in U' with the path from $b^{-1}(u_0)$ to u_ℓ in U. Hence $\pi(T) \in \mathcal{T}_n^{(1)}$. Since $\pi_{b^{-1}}$ is clearly an inverse of π_b, we have that π is a bijection.

Let $s \in S_{t-1}$ be such that $P_T(s) \in I_{t-1}$. If $s \notin U'$, then it has the same parent in T and in $\pi(T)$. If $s \in U'$, which is a non-increasing tree, then by the fact that the parent of s in T belongs to I_{t-1} it follows that s is the root of U', and, thus, that its parent does not belong to U'. By the definition of a correction it follows that the parent of s in U is informed. As $U \subseteq \pi(T)$ the parent of s in $\pi(T)$ is a node of I_{t-1}.

We, therefore, have that, for every $x \in \mathbb{N}$:

$$\begin{aligned}
\mathbb{P}(N_t^{(2)} - N_{t-1} \geq x | \mathcal{F}_{t-1}) &= \frac{\left|\{T \in \mathcal{T}_n^{(2)} : |\{s \in S_{t-1} : P_T(s) \in I_{t-1}\}| \geq x\}\right|}{\left|\mathcal{T}_n^{(2)}\right|} \\
&\leq \frac{\left|\{T \in \mathcal{T}_n^{(2)} : \left|\{s \in S_{t-1} : P_{\pi(T)}(s) \in I_{t-1}\}\right| \geq x\}\right|}{\left|\mathcal{T}_n^{(1)}\right|} \\
&\leq \frac{\left|\{T \in \mathcal{T}_n^{(1)} : |\{s \in S_{t-1} : P_T(s) \in I_{t-1}\}| \geq x\}\right|}{\left|\mathcal{T}_n^{(1)}\right|} \\
&= \mathbb{P}(N_t^{(1)} - N_{t-1} \geq x | \mathcal{F}_t t - 1)
\end{aligned}$$

The lemma now follows from the Distribution Domination Lemma (Lemma 4.6.6). □

Lemma 4.6.14. *Let t be a round and N_{t-1} be the number of informed nodes after round $t-1$. Let E_1, E_2 be two sets of edges that the adversary could choose for round t, as follows: let E_1 be a collection of rooted trees such that every tree is non-increasing, with at least two non-trivial components U with root r and U' with root r', and let E_2 be obtained from E_1 by merging U and U'. Let $N_t^{(1)}$ be the number of informed nodes after round t if E_1 is chosen, and $N_t^{(2)}$ if E_2 is chosen. Then choosing E_2 is a better strategy for the adversary than choosing E_1.*

Proof. We will show that for any $x \in \mathbb{N}$, we have that $\mathbb{P}(N_t^{(1)} - N_{t-1} = x | \mathcal{F}_{t-1}) \geq \mathbb{P}(N_t^{(2)} - N_{t-1} = x | \mathcal{F}_{t-1})$. Then the result will follow from the Distribution Domination Lemma.

In the following let S be a set of uninformed nodes $s_1, \ldots, s_{|S|}$, let η_j for $1 \leq j \leq |S|$ be the number of informed nodes in the connected component of s_j in E_1 and let $\eta(S)$ be the number of informed nodes that do not belong to the connected component of any s_j.

We will analyze the value of $\mathbb{P}(\cap_{s \in S} P_t(s) \in I_{t-1} | \mathcal{F}_{t-1})$ when the adversary chooses E_1, and when it chooses E_2. Then two cases can arise: Either the value of $\sum_{|S|=\ell} \mathbb{P}(\cap_{s \in S} P_t(s) \in I_{t-1} | \mathcal{F}_{t-1})$ is equal whether the adversary chooses E_1 or E_2, and this for every ℓ, and the

result will follow Lemma 4.9.23, or $\mathbb{P}(\cap_{s\in S}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1})$ will be the same whether the adversary chooses E_1 or E_2 for every set S except if S includes a particular node, where there will be a constant factor difference between the two values, and the result will then follow from Lemma 4.9.24.

As all trees in E_1 (respectively E_2) are non-increasing, the parent in E_1 (respectively E_2) of every non-root node $s \in S$ is uniformed. Thus, if there exists a node in S that is not a root of E_1 (respectively E_2) then $\mathbb{P}(\cap_{s\in S}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = 0$. Hence, we only need to analyze the setting where all nodes of S are roots in E_1.

Case A: Let us first consider the case $r \in I_{t-1}$ in which case the merge of U and U' makes all children of r to children of r'. In that case, $r \notin S$ and let $\gamma = |U| - 1$. We have two subcases: (A1) If $r' \notin S$, then the number of informed nodes in none of the components with roots in S, $\eta(S)$, remains unchanged. It follows from Lemma 4.6.12 that $\mathbb{P}(\cap_{s\in S}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1})$ is the same whether the adversary chooses E_1 or E_2. (A2) If $r' \in S$, wlog assume that $r' = s_1$. Then we have that $\mathbb{P}(\cap_{s\in S}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = \frac{\eta(S)(N_{t-1})^{|S|-1}}{n^{|S|}}$ if the adversary chooses E_1, while $\mathbb{P}(\cap_{s\in S}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = \frac{(\eta(S)-\gamma)(N_{t-1})^{|S|-1}}{n^{|S|}}$ if the adversary chooses E_2. Applying Lemma 4.9.24 where we set $X_s = P_t(s) \in I_{t-1}$ if the adversary chooses E_1, and $Y_s = P_t(s) \in I_{t-1}$ if the adversary chooses E_2, and $\alpha = \frac{\eta(S)}{\eta(S)-\gamma}$, we have that $N_t^{(1)} - N_{t-1} = \sum_{s\in S_{t-1}} X_s$ stochastically dominates $N_t^{(2)} - N_{t-1} = \sum_{s\in S_{t-1}} Y_s$. The result follows from the Distribution Domination Lemma.

Case B: Let us now look at the case where $r \notin I_{t-1}$. In this case the merge of U and U' makes all children of U' children of U.

We consider again two cases: (B1) If $r' \in I_{t-1}$, then this case is symmetric to Case (A1) and the same proof as above applies.

(B2) If $r' \notin I_{t-1}$, for any $\ell \in \mathbb{N}$, we have that:

$$\begin{aligned}\sum_{|S|=\ell} \mathbb{P}(\cap_{s\in S}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = &\sum_{|S|=\ell:r,r'\notin S} \mathbb{P}(\cap_{s\in S}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) \\ &+ \sum_{|S|=\ell:r,r'\in S} \mathbb{P}(\cap_{s\in S}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) \\ &+ \sum_{|S|=\ell-1:r,r'\notin S} \mathbb{P}(\cap_{s\in S\cup\{r\}}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) \\ &+ \mathbb{P}(\cap_{s\in S\cup\{r'\}}P_t(s) \in I_{t-1}|\mathcal{F}_{t-1})\end{aligned}$$

We need to analyze the three sums.

For the first two sums, where both r and r' or neither belong to S, the number of informed nodes in none of the components with root in S, $\eta(S)$, is not different in E_1 and in E_2 and, thus, Lemma 4.6.12 implies that $\mathbb{P}(\cap_{s\in S}P_t(s) \in I_{t-1})$ does not change whether the adversary chooses E_1 or E_2.

For the third sum, let γ, γ' be respectively the number of informed nodes in the component of r, r' in E_1. Let us first consider the case where the adversary chooses E_1. We have, by

Lemma 4.6.12:

$$\mathbb{P}(\cap_{s\in S\cup\{r\}} P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = \frac{(\eta(S)-\gamma)(N_{t-1})^{|S|}}{n^{|S|+1}}$$

and

$$\mathbb{P}(\cap_{s\in S\cup\{r'\}} P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = \frac{(\eta(S)-\gamma')(N_{t-1})^{|S|}}{n^{|S|+1}}$$

therefore:

$$\mathbb{P}(\cap_{s\in S\cup\{r\}} P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) + \mathbb{P}(\cap_{s\in S\cup\{r'\}} P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = \frac{(2\eta(S)-\gamma-\gamma')(N_{t-1})^{|S|}}{n^{|S|+1}}$$

If the adversary chooses E_2, then r has $\gamma+\gamma'$ informed nodes in its component in E_2, while r' has 0 of them. by Lemma 4.6.12:

$$\mathbb{P}(\cap_{s\in S\cup\{r\}} P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = \frac{(\eta(S)-\gamma-\gamma')(N_{t-1})^{|S|}}{n^{|S|+1}}$$

and

$$\mathbb{P}(\cap_{s\in S\cup\{r'\}} P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = \frac{\eta(S)(N_{t-1})^{|S|}}{n^{|S|+1}}$$

therefore:

$$\mathbb{P}(\cap_{s\in S\cup\{r\}} P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) + \mathbb{P}(\cap_{s\in S\cup\{r'\}} P_t(s) \in I_{t-1}|\mathcal{F}_{t-1}) = \frac{(2\eta(S)-\gamma-\gamma')(N_{t-1})^{|S|}}{n^{|S|+1}}$$

Therefore, $\sum_{|S|=\ell}\mathbb{P}(\cap_{s\in S}P_t(s)\in I_{t-1})$ has the same value whether the adversary chooses E_1 or E_2. Applying Lemma 4.9.23 where we set $X_s = P_t(s)\in I_{t-1}$ if the adversary chooses E_1, and $Y_s = P_t(s)\in I_{t-1}$ if the adversary chooses E_2, we have that $N_t^{(1)} - N_{t-1} = \sum_{s\in S_{t-1}} X_s$ and $N_t^{(2)} - N_{t-1} = \sum_{s\in S_{t-1}} Y_s$ have the same distribution. The result follows from the Distribution Domination Lemma.

□

Lemma 4.6.15. *Let t be a round and N_t be the number of informed nodes after round t. Let U be a non-increasing tree over $k+1$ nodes in round $t+1$. Let σ be the number of uninformed nodes in U and η the number of informed nodes in U. Then the distribution of $N_{t+1}-N_t$ equals the sum of of $n-N_t-\sigma$ independent Bernoulli random variables of parameter $\frac{N_t}{n}$ plus one Bernoulli random variable of parameter $\frac{N_t-\eta}{n}$.*

Proof. Let $I_t = \{i_1, \ldots, i_{N_t}\}$ and $S_t = \{s_1, \ldots, s_{n-N_t}\}$ such that $i_1, \ldots, i_\eta$ are nodes of U, $s_1, \ldots, s_{\sigma-1}$ are uninformed nodes of U that are not the root, and s_σ is the root of U. As U is non-increasing $s_1, \ldots s_{\sigma-1}$ cannot get informed in round $t+1$. As the parent of s_σ does not belong to U, it cannot belong to $i_1, \ldots, i_\eta$. We will show that the events *uninformed node s gets informed in round $t+1$* for different uninformed nodes $s \in [\sigma, n-N_t]$ are mutually independent. To do so we take some $J \subseteq [\sigma, n-N_t]$ and analyze the event $\bigcap_{y\in J}(P_t(s_y) \in I_t)$, We distinguish two cases.

Case 1: If $\sigma \notin J$, then it holds that

$$\begin{aligned}\mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(s_y) \in I_t)\middle|\mathcal{F}_t\right) &= \sum_{a\in[N_t]^{|J|}} \mathbb{P}\left(\bigcap_{s_y\in J}(P_{t+1}(s_y) = i_{a_y})\middle|\mathcal{F}_t\right) \\ &= \sum_{a\in[N_t]^{|J|}} \frac{\left|\{T' \in \mathcal{T}_n : P_U(s_y) = i_{a_y}, \forall y \in J \wedge U \subset T'\}\right|}{|\{T' \in \mathcal{T}_n : U \subset T'\}|}\end{aligned}$$

By Theorem 4.9.1, we have that $|\{T' \in \mathcal{T}_n : U \subset T'\}| = n^{n-1-k}$, and $\left|\{T' \in \mathcal{T}_n : P_U(s_y) = i_{a_y}, \forall y \in J \wedge U \subset T'\}\right| = n^{n-1-k-|J|}$. Therefore it follows that:

$$\mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(s_y) \in I_t)\middle|\mathcal{F}_t\right) = \sum_{a\in[N_t]^{|J|}} \frac{n^{n-1-|J|}}{n^{n-1}} = \left(\frac{N_t}{n}\right)^{|J|}$$

Case 2: If $\sigma \in J$, we have to take extra care of node s_σ:

$$\begin{aligned}\mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(b_y) \in I_t)\middle|\mathcal{F}_t\right) &= \sum_{a\in[N_t]^{|J|-1}\times[\eta+1,N_t]} \mathbb{P}\left(\bigcap_{s_y\in J}(P_{t+1}(b_y) = i_{a_y})\middle|\mathcal{F}_t\right) \\ &= \sum_{a\in[N_t]^{|J|-1}\times[\eta+1,N_t]} \frac{\left|\{T' \in \mathcal{T}_n : P_U(s_y) = i_{a_y}, \forall y \in J \wedge U \subset T'\}\right|}{|\{T' \in \mathcal{T}_n : U \subset T'\}|}\end{aligned}$$

By Theorem 4.9.1, we have that $|\{T' \in \mathcal{T}_n : U \subset T'\}| = n^{n-1-k}$, and $\left|\{T' \in \mathcal{T}_n : P_U(s_y) = i_{a_y}, \forall y \in J \wedge U \subset T'\}\right| = n^{n-1-k-|J|}$. Therefore we have that:

$$\mathbb{P}\left(\bigcap_{y\in J}(P_{t+1}(s_y) \in I_t)\middle|\mathcal{F}_t\right) = \sum_{a\in[N_t]^{|J|-1}\times[\eta+1,N_t]} \frac{n^{n-1-|J|}}{n^{n-1}} = \left(\frac{N_t}{n}\right)^{|J|-1}\frac{N_t-\eta}{n}$$

This proves that the events $P_{t+1}(s_y) \in I_t$ are mutually independent for every $y \geq \sigma$, each having probability $\frac{N_t}{n}$, except if $y = \sigma$, which has probability $\frac{N_t-\eta}{n}$.

□

4.9.7 Beyond Trees: Broadcast and Consensus in directed ErdsRényi graphs

In this section, we will study the case where in each round, the communication graph is a directed ErdsRényi Graph with m edges. Choosing such a graph can be seen as choosing without replacement m edges in the graph. To do so, we will analyze three different schemes.

In *scheme 1*, in each round, m edges are chosen uniformly at random *without* replacement among the n^2 possible edges. This is equivalent to our model.

In *scheme 2*, in each round, m edges are chosen uniformly at random *with* replacement among the n^2 possible edges. This can only result in more rounds than scheme 1 as fewer disjoint edges are chosen in each round compared to scheme 1.

In *scheme 3*, we start with a unique informed node 1. There are two types of phases for a total of $2\left\lceil \log \frac{n}{2} \right\rceil$ phases. In each phase i with $1 \leq i \leq \left\lceil \log \frac{n}{2} \right\rceil$, we have $N_i = 2^{i-1}$ informed nodes in I_i at the beginning of the phase and the goal is to double that number within the phase. We set $E = \varnothing$ and add to E one edge at a time, sampled with replacement, until $|I_i \cup \mathcal{N}_E(I_i)| = \min\{2^i, \left\lceil \frac{n}{2} \right\rceil\}$. We then set $I_{i+1} = I_i \cup \mathcal{N}_E(I_i)$. Note that at the end of phase $i = \left\lceil \log \frac{n}{2} \right\rceil$, $N_{i+1} = \left\lceil \frac{n}{2} \right\rceil$. Note that this scheme is independent of m.

Then in each phase $i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ with $0 \leq j < \left\lceil \log \frac{n}{2} \right\rceil$, we initially have N_i informed nodes in I_i, and $\min\{2^j, \left\lfloor \frac{n}{2} \right\rfloor\}$ uninformed nodes in S_i and the goal is to halve the number of uninformed nodes in each phase. We set $E = \varnothing$ and add to E, one edge at a time sampled with replacement, until $|S_i \setminus \mathcal{N}_E(I_i)| = 2^{j-1}$. We then set $I_{i+1} = I_i \cup \mathcal{N}_E(I_i)$.

Let us intuitively compare scheme 2 and scheme 3 and assume initially that $m = 1$. Then scheme 2 in each round chooses an edge uniformly at random, forwards information along it if its source is an informed node, and then moves on to the following round. Scheme 3 on the other hand, will continue to sample edges until enough progress can be made along those edges, and then forward information along those edges all at once, before moving to the next phase. Overall, scheme 3 will sample more edges than scheme 2, as in scheme 2 any progress is made as soon as possible, whereas in scheme 3 progress is only made when checkpoints are reached.

In this section, we will expand this intuition for any $m \in [n^2]$, give upper bounds on the number of edges sampled by scheme 3, then use those results to get an upper bound on the number of rounds needed by scheme 2, and use it to give an upper bound for scheme 1. Note that in scheme 3, we only count the number of edges sampled, and we will not be talking about rounds in that scheme. We thus start by analyzing scheme 3:

Lemma 4.9.44. *For any $c \geq 1$, any phase $i \leq \left\lceil \log \frac{n}{2} \right\rceil$ needs at most $8 \cdot c \cdot \max\{\ln n, 2^{i-2}\} \frac{n}{2^{i-2}}$ sampled edges to complete with probability $p \geq 1 - n^{-c}$.*

Proof. We first note that in phase i with $i \leq \left\lceil \log \frac{n}{2} \right\rceil$, the probability that an edge that is sampled increases $I_i \cup \mathcal{N}_E(I_i)$ is at least $\frac{2^{i-2}}{n}$. Indeed, $|I_i \cup \mathcal{N}_E(I_i)| \leq \frac{n}{2}$ (otherwise the phase would have ended) and there are $|I_i| \cdot |[n] \setminus (I_i \cup \mathcal{N}_E(I_i))| \geq 2^{i-2} n$ edges that can increase $I_i \cup \mathcal{N}_E(I_i)$. Thus, picking any of those edges, out of n^2 possible ones, has probability at least $\frac{2^{i-2}}{n}$.

Next, we remark that if we sample $\frac{n}{2^{i-2}}$ edges, then the probability that at least one of those edges increases $I_i \cup \mathcal{N}_E(I_i)$ is at least $\frac{1}{2}$. Indeed, the probability that all of those edges do not make $I_i \cup \mathcal{N}_E(I_i)$ larger is $(1 - \frac{2^{i-2}}{n})^{\frac{n}{2^{i-2}}} \leq e^{-1} \leq \frac{1}{2}$. We will, thus, group the sampled edges into disjoint "buckets" of $\frac{n}{2^{i-2}}$ consecutively sampled edges.

We then use the fact that we only need 2^{i-1} edges that increase $I_i \cup \mathcal{N}_E(I_i)$ to end phase i. If we take $8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}$ buckets of $\frac{n}{2^{i-2}}$ edges each, then applying Hoeffding's inequality, we have:

$$\mathbb{P}(|I_i \cup \mathcal{N}_E(I_i)| \leq 2^i - 1) \leq \exp\left(-2 \cdot 8c \cdot \max\{\ln n, 2^{i-2}\}\left(\frac{1}{2} - \frac{2^{i-1}}{8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}}\right)^2\right)$$
$$\leq \exp\left(-16 \cdot c \cdot \ln n \left(\frac{1}{2} - \frac{1}{4}\right)^2\right) = n^{-c}$$

Which proves that with probability $p \geq 1 - n^{-c}$, phase i ends after sampling at most $8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}\frac{n}{2^{i-2}}$ edges. □

We have a symmetric result:

Lemma 4.9.45. *For any $c \geq 1$, any phase i with $i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ for $1 \leq j < \left\lceil \log \frac{n}{2} \right\rceil$ needs at most $8 \cdot c \cdot \max\{\ln n, 2^{j-2}\}\frac{n}{2^{j-2}}$ sampled edges to complete with probability $p \geq 1 - n^{-c}$.*

Proof. We first note that, by the stopping condition for phase $\left\lceil \frac{n}{2} \right\rceil$, for any phase i with $i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ for $1 \leq j < \left\lceil \log \frac{n}{2} \right\rceil$, it holds that $N_i \geq \left\lceil \frac{n}{2} \right\rceil$. We first show that in phase i, the probability that a sampled edge decreases $S_i \setminus \mathcal{N}_E(I_i)$ is at least $\frac{2^{j-2}}{n}$. Indeed, $|[n] \setminus (I_i \cup \mathcal{N}_E(I_i))| = |S_i \setminus \mathcal{N}_E(I_i)| > 2^{j-1}$ (otherwise the phase would have ended) and, thus, there are $N_i \cdot |[n] \setminus (I_i \cup \mathcal{N}_E(I_i))| \geq 2^{j-2}n$ edges that would decrease $S_i \setminus \mathcal{N}_E(I_i)$. Thus, picking any of those edges, out of n^2 possible ones, has probability at least than $\frac{2^{j-2}}{n}$.

The rest of the proof is symmetrical to the previous one. □

Lemma 4.9.46. *For any $t \in \mathbb{N}$, $p \in [0, 1]$, if Broadcast completes in scheme 2 within t rounds with probability at least p, then the same result holds for scheme 1.*

Proof. We will couple schemes 1 and 2 as follows: One can see sampling without replacement of m edges as sampling with replacement of as many edges as needed until the number of different edges sampled is equal to m. Indeed, to sample m edges without replacement, one first must choose an edge uniformly at random, then another edge uniformly at random among the remaining edges, and so on until we have sampled m edges. If we sample *with* replacement until we have m different edges, then whenever we have sampled $i \in [m-1]$ edges, the next new edge is chosen by sampling edges with replacement until a new edge is selected. This new edge is thus chosen uniformly at random among remaining edges.

For each round t, we sample m edges with replacement and call the resulting set of edges $E_t^{(2)}$. This is the set of edges for scheme 2. To build $E_t(1)$, the set of edges for scheme 1, we start with $E_t(1) = E_t^{(2)}$, and add to sampled edges with replacement until $\left|E_t^{(1)}\right| = m$. The

set $E_t(1)$ sampled that way follows the distribution of m sampled edges without replacement. We thus have that, with probability 1, $E_t^{(2)} \subseteq E_t^{(1)}$.

We now show that in each round t, we have that $\mathbb{P}(I_t^{(2)} \subseteq I_t^{(1)}) = 1$, where $I_t^{(i)}$ is the set of informed nodes in scheme i after round t. Indeed, by induction, this is trivial for $t = 0$. Let's assume it is true for some t. Then let $v \in I_{t+1}^{(2)}$. Then we either have $v \in I_t^{(2)} \subseteq I_t^{(1)} \subseteq I_{t+1}^{(1)}$, or that $v \notin I_t^{(2)}$, and thus there exist a node $u \in I_t^{(2)}$ such that edge $(u, v) \in E_{t+1}^{(2)}$. However, $u \in I_t^{(1)}$ by induction hypothesis, and $(u, v) \in E_{t+1}^{(2)} \subseteq E_{t+1}^{(1)}$. Therefore $v \in I_{t+1}^{(1)}$.

This proves that with probability 1, Broadcast completes in scheme 1 no later than it completes in scheme 2, and thus the result holds. □

This allows us to prove the following result on scheme 2:

Theorem 4.7.1. *For any $c \geq 1$, in scheme 2, and therefore scheme 1, Broadcast completes within $O\left(\left\lceil \frac{cn}{m} \right\rceil \log n\right)$ rounds with probability $p \geq 1 - n^{-c} \log n$.*

Proof. To see this, we are going to simulate scheme 3 with scheme 2. The main idea is that if a phase (of scheme 3) takes x edges, sampled with replacement, to end with high probability, and in scheme 2 each round samples y edges with replacement, then in $\left\lceil \frac{x}{y} \right\rceil$ rounds, scheme 2 samples at least x edges, and, thus, we can simulate the phase in scheme 3 with $\left\lceil \frac{x}{y} \right\rceil$ rounds of scheme 2. The only difference is that scheme 2 groups the edges into rounds to make intermediate progress, whereas scheme 3 only forwards the information at the end of the phase, all at once. This implies that in scheme 2 each phase is faster than the corresponding one in scheme 3, and any upper bound we get with this analysis will thus be an upper bound on the number of rounds scheme 2 needs to complete Broadcast.

We first start with the phases $i \leq \left\lceil \log \frac{n}{2} \right\rceil$ such that $\ln n \leq 2^{i-2}$ and the phases $i > \left\lceil \log \frac{n}{2} \right\rceil$ with $i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ for $j \geq 1$ such that $\ln n \leq 2^{j-2}$. In that case, phase i needs at most $8 \cdot c \cdot n$ sampled edges to end with probability larger than $1 - n^{-c}$. We need at most $\left\lceil \frac{8 \cdot c \cdot n}{m} \right\rceil = \left\lceil \frac{8 \cdot c}{m/n} \right\rceil$ rounds to gather that many edges, and thus phase i ends in $\left\lceil \frac{8 \cdot c}{m/n} \right\rceil$ rounds with probability greater than $1 - n^{-c}$. There are at most $\lceil \log n \rceil$ such phases, and thus over all phases we require at most $O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$.

Let us now analyze the phases $i \leq \left\lceil \log \frac{n}{2} \right\rceil$ where $\ln n > 2^{i-2}$ and symmetrically phases $i > \left\lceil \log \frac{n}{2} \right\rceil$, with $i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ for $j \geq 1$ such that $\ln n > 2^{j-2}$. In that case, phase i needs at most $8 \cdot c \cdot \ln n \cdot \frac{n}{2^{i-2}}$ sampled edges to end with probability larger than $1 - n^{-c}$. We need at most $\left\lceil \frac{8 \cdot c \cdot \ln n \cdot \frac{n}{2^{i-2}}}{m} \right\rceil = \left\lceil \frac{8 \cdot c \cdot \ln n}{m/n \cdot 2^{i-2}} \right\rceil$ rounds to gather that many edges, and thus phase i ends in $\left\lceil \frac{8 \cdot c \cdot \ln n}{m/n \cdot 2^{i-2}} \right\rceil$ rounds with probability greater than $1 - n^{-c}$. Summing the number of rounds over all such phases we get:

$$\sum_i \left\lceil \frac{8 \cdot c \cdot \ln n}{m/n \cdot 2^{i-2}} \right\rceil \leq \sum_i \left(\frac{8 \cdot c \cdot \ln n}{m/n \cdot 2^{i-2}} + 1\right) \leq \frac{32 \cdot c \cdot \ln n}{m/n} + \log n = O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$$

The probability of success $p \geq 1 - n^{-c} \log n$ is simply a union bound on the number of phases. □

This result is particularly interesting if $m \leq cn$. We can also show the following result if $m \geq n \ln n$, which is more interesting in that particular case.

Theorem 4.7.3. *For any $c \geq 1$ and $m \in [n^2]$ such that $m/n \geq \ln n$, in scheme 2 and in scheme 1, Broadcast completes within $O\left(\frac{c \cdot \log n}{\log(1+m/n)}\right)$ rounds with probability $p \geq 1 - n^{-c} \log n$.*

Proof. We show that bound for scheme 2. With Lemma 4.9.46 the bound for scheme 1 immediately follows. Again, we introduce scheme 3, however, we modify it so that the goal of each phase is not to multiply (respectively, divide) the number of informed (respectively, uninformed) nodes by 2, but instead, it is to multiply (respectively, divide) it by $(1 + m/n)$. As a result, we get a total number of $O\left(\frac{\log n}{\log(1+m/n)}\right)$ phases.

In this case, as formally discussed below, each phase necessitates $16 \cdot c \cdot m$ edges to complete with high probability, but each round provides m edges. Therefore each phase consists of $\lceil 16c \rceil$ rounds.

Formally, in phase $i \leq \frac{\log(n/2)}{\log(1+m/n)}$, we start with $(1 + m/n)^{i-1}$ informed nodes, and at least $\frac{n}{2} \cdot (1 + m/n)^{i-1}$ edges out of the n^2 potential edges can inform an uninformed node. Thus, each sampled edge has probability at least $\frac{1}{2n} \cdot (1 + m/n)^{i-1}$ of informing an uninformed node. Hence, by the same argument as in Lemma 4.9.44, we need to sample $\frac{2n}{(1+m/n)^{i-1}}$ edges to inform a new node with probability at least $\frac{1}{2}$.

To go from $(1 + m/n)^{i-1}$ informed nodes to $(1 + m/n)^i$ informed nodes, we need to inform $m/n(1+m/n)^{i-1}$ uninformed nodes. If we sample $8c \cdot m/n(1+m/n)^{i-1}$ buckets of $\frac{2n}{(1+m/n)^{i-1}}$ edges each(for a total of $16c \cdot m$ edges), we get by Hoeffding's inequality that the probability that not enough edges inform a new node is:

$$\mathbb{P}(|I_i \cup \mathcal{N}_E(I_i)| \leq (1+\frac{m}{n})^i - 1) \leq \exp\left(-2 \cdot 8c \cdot m/n(1+m/n)^{i-1}\left(\frac{1}{2} - \frac{m/n(1+m/n)^{i-1}}{8c \cdot \frac{m}{n}(1+\frac{m}{n})^{i-1}}\right)^2\right)$$
$$\leq \exp\left(-16 \cdot c \cdot \ln n \left(\frac{1}{2} - \frac{1}{4}\right)^2\right) = n^{-c}$$

□

We also show a lower bound:

Theorem 4.7.2. *In scheme 1, and thus in scheme 2, Broadcast fails to complete within $\frac{\log(n)-1}{\log(1+m/n)}$ rounds with probability at least $\frac{1}{2}$.*

Proof. We will show by induction that, in scheme 1, $\mathbb{E}(|I_t|) \leq (1 + m/n)^t$ for every $t \in \mathbb{N}$. We will then apply Markov's inequality to conclude.

Let us first compute $\mathbb{E}\left(|I_t| \,\middle|\, |I_{t-1}| = x\right)$. Let $v \notin I_{t-1}$ be an uninformed node, and let e be an incoming edge to v such that its tail is in I_{t-1}. Then the probability that this edge is picked is $\frac{m}{n^2} := \rho$. By a union bound over the x edges (u, v) such that $u \in I_{t-1}$, we

have that $\mathbb{P}(v \in I_t \Big| |I_{t+1}| = x) \leq \rho x$. Denoting X_v the variable $v \in I_t$, we then have that $\mathbb{E}(X_v \Big| |I_{t+1}| = x) \leq \rho x$.

Summing that expectation over all $v \in [n] \setminus I_{t-1}$, we have that:

$$\mathbb{E}\left(|I_t| \,\middle|\, |I_{t-1}| = x\right) = x + \sum_{v \in [n]-I_{t-1}} \mathbb{E}(X_v) = x + \sum_{v \in [n]-I_{t-1}} \rho x \leq x + (n-x)\rho x \leq x(1+m/n) \quad (4.1)$$

We can now prove our claim by induction. For the induction basis, we clearly have $\mathbb{E}(I_0) = 1 = (1+m/n)^0$. For the induction step, assume that for some $t \geq 1$, we have that $\mathbb{E}(I_{t-1}) \leq (1+m/n)^{t-1}$. Then:

$$\mathbb{E}\left(|I_t|\right) = \mathbb{E}\left(\mathbb{E}(|I_t| \,\middle|\, |I_{t-1}|)\right) \leq \mathbb{E}\left(|I_{t-1}|\left(1+m/n\right)\right) \leq (1+m/n)^t$$

Where the first inequality holds by Equation 4.1 and the second inequality holds by the induction hypothesis.

Therefore, we have that $\mathbb{E}\left(\left|I_{\frac{\log(n)-1}{\log(1+m/n)}}\right|\right) \leq \frac{n}{2}$. Using Markov's inequality, we then have that:

$$\mathbb{P}\left(\left|I_{\frac{\log(n)-1}{\log(1+m/n)}}\right| \geq n\right) \leq \frac{\mathbb{E}\left(\left|I_{\frac{\log(n)-1}{\log(1+m/n)}}\right|\right)}{n} \leq \frac{1}{2}$$

□

Using a union-bound in the same way as for the uniformly random trees model, we have a result on All-to-All Broadcast:

Theorem 4.9.47. *For any $c \geq 1$, $n \geq 5$ $m \in [n^2]$, All-to-All Broadcast on directed Erdsrényi graphs completes within $O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$ rounds with probability $p > 1 - n^{c-1}\log n$. Moreover, if $m/n \geq \ln n$, All-to-All Broadcast on directed ErdsRényi graphs completes within $O\left(\frac{c \cdot \log n}{\log(1+m/n)}\right)$ rounds with probability $p > 1 - n^{c-1}\log n$.*

Finally, using Algorithm 4.4.10 for Consensus, we have:

Theorem 4.9.48. *For any $c \geq 1$, $n \geq 5$ $m \in [n^2]$, there exists a protocol for Consensus on directed ErdsRényi graphs that satisfies Agreement and Validity, terminates within $O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$ rounds with probability $p > 1 - \frac{1}{n^c}$, and only requires messages of 1 bit over each edge in each round. Moreover, if $m/n \geq \ln n$, then we get the better bound $O\left(\frac{c \cdot \log n}{\log(1+m/n)}\right)$ for the number of rounds, with the same probability of success.*

4.9.8 Byzantine Nodes in directed ErdsRényi graphs

We now analyze what happens if some nodes deviate arbitrarily from the protocol. More specifically, we allow up to $f < \frac{2n}{3}$ nodes, the *Byzantine nodes*, to coordinate to delay Broadcast as much as possible. Moreover, we give every node access to a cryptographic tools, so

that nodes can sign messages, and ensure any message they receive, even if forwarded, has been sent "as is" from the not who signed the message. As in Section 4.5, the best strategy Byzantine nodes can thus have is to stop forwarding messages. To analyze the problem, we consider the three schemes as above:

In *scheme 1*, in each round, m edges are chosen uniformly at random *without* replacement among the n^2 possible edges. This is equivalent to our model.

In *scheme 2*, in each round, m edges are chosen uniformly at random *with* replacement among the n^2 possible edges. This can only result in more rounds than scheme 1 as fewer disjoint edges are chosen in each round compared to scheme 1.

In *scheme 3*, we start with a unique informed node 1. We then run $\left\lceil \log \frac{n-f}{2} \right\rceil$ phases. In phase i, we have $N_i = 2^{i-1}$ honest informed nodes I_i. We set $E = \varnothing$ and add one edge at a time, sampled with replacement, to E until $|I_i \cup \mathcal{N}_E(I_i)| = \min\{2^i, \left\lceil \frac{n-f}{2} \right\rceil\}$. We then set $I_{i+1} = I_i \cup \mathcal{N}_E(I_i)$.

We then run $\left\lceil \log \frac{n-f}{2} \right\rceil$ other phases. In phase $i = 2\left\lceil \log \frac{n-f}{2} \right\rceil - j$, we have N_i honest informed nodes I_i, and honest uninformed nodes S_i, with $|S_i| = \min\{2^j, \left\lfloor \frac{n-f}{2} \right\rfloor\}$. We set $E = \varnothing$ and add to E, one edge at a time, sampled with replacement until $|S_i \setminus \mathcal{N}_E(I_i)| = 2^{j-1}$. We then set $I_{i+1} = I_i \cup \mathcal{N}_E(I_i)$.

As above, we start by analyzing scheme 3:

Lemma 4.9.49. *For any $c \geq 1$, any phase $i \leq \left\lceil \log \frac{n-f}{2} \right\rceil$ needs at most $24 \cdot c \cdot \max\{\ln n, 2^{i-2}\} \frac{n}{2^{i-2}}$ sampled edges to complete with probability $p \geq 1 - n^{-c}$.*

Proof. We first note that in phase i, the probability that an edge being sampled makes $I_i \cup \mathcal{N}_E(I_i)$ larger is at least $\frac{2^{i-2}}{3n}$. Indeed, $|I_i \cup \mathcal{N}_E(I_i)| \leq \frac{n-f}{2}$ (otherwise the phase would have ended) and there are $|I_i| \cdot |[n] \setminus (I_i \cup \mathcal{N}_E(I_i))| \geq 2^{i-2}(n-f)$ edges that would make $I_i \cup \mathcal{N}_E(I_i)$ larger. Therefore picking any of those edges, out of n^2 possible ones, has probability at least $\frac{2^{i-2}(n-f)}{n^2} \geq \frac{2^{i-2}}{3n}$.

Next, we remark that if we sample $\frac{3n}{2^{i-2}}$ edges, then the probability that at least one of those edges makes $I_i \cup \mathcal{N}_E(I_i)$ larger is at least $\frac{1}{2}$. Indeed, the probability that all of those edges do not make $I_i \cup \mathcal{N}_E(I_i)$ larger is $(1 - \frac{2^{i-2}}{3n})^{\frac{3n}{2^{i-2}}} \leq e^{-1} \leq \frac{1}{2}$. We will, thus, group the edges into "buckets" of $\frac{3n}{2^{i-2}}$ edges.

We then use the fact that we only need 2^{i-1} edges that make $I_i \cup \mathcal{N}_E(I_i)$ larger to end phase i. If we take $8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}$ buckets of $\frac{3n}{2^{i-2}}$ edges, then applying Heoffding's inequality, we have:

$$\begin{aligned}\mathbb{P}(|I_i \cup \mathcal{N}_E(I_i)| < 2^i) &\leq \exp\left(-2 \cdot 8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}\left(\frac{1}{2} - \frac{2^{i-1}}{8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}}\right)^2\right) \\ &\leq \exp\left(-16 \cdot c \cdot \ln n \left(\frac{1}{2} - \frac{1}{4}\right)^2\right) = n^{-c}\end{aligned}$$

Which proves that with probability $p \geq 1 - n^{-c}$, phase i ends after sampling at most $24 \cdot c \cdot \max\{\ln n, 2^{i-2}\} \frac{n}{2^{i-2}}$ edges. □

We have a symmetric result:

Lemma 4.9.50. *For any $c \geq 1$, any phase $i > \left\lceil \log \frac{n-f}{2} \right\rceil, i = 2\left\lceil \log \frac{n-f}{2} \right\rceil - j$ needs at most $24 \cdot c \cdot \max\{\ln n, 2^{j-2}\} \frac{n}{2^{j-2}}$ samplings to complete with probability $p \geq 1 - n^{-c}$.*

Proof. We first note that in phase i, the probability that an edge being sampled makes $S_i \setminus \mathcal{N}_E(I_i)$ smaller is at least $\frac{2^{j-2}}{3n}$. Indeed, $|S_i \setminus \mathcal{N}_E(I_i)| \geq 2^{j-1}$ (otherwise the phase would have ended) and there are $|I_i| \cdot |[n] \setminus (I_i \cup \mathcal{N}_E(I_i))| \geq 2^{j-2}(n-f)$ edges that would make $S_i \setminus \mathcal{N}_E(I_i)$ smaller. Therefore picking any of those edges, out of n^2 possible ones, has probability at least $\frac{2^{j-2}(n-f)}{n^2} \geq \frac{2^{j-2}}{3n}$.

The rest of the proof is symmetrical to the previous one. □

This allows us to prove the following result on scheme 2:

Theorem 4.9.51. *For any $c \geq 1$, in scheme 2, and therefore scheme 1, Broadcast completes within $O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$ rounds with probability $p \geq 1 - n^{-c} \log n$.*

Proof. To see this, we are going to simulate scheme 3 with scheme 2. The main idea is that if a phase (of scheme 3) takes x number of edges to end with high probability, and in scheme 2 each rounds samples y edges without replacement, then in $\left\lceil \frac{x}{y} \right\rceil$ rounds, scheme 2 samples more than x edge, and thus we can simulate the phase in scheme 3 with $\left\lceil \frac{x}{y} \right\rceil$ rounds of scheme 2. The only difference is that scheme 2 groups the edges per round to make intermediate progress, whereas scheme 3 only forwards the information at the end of the phase, all at once. This is only beneficial to scheme 2, and any upper bound we get with this analysis will thus be an upper bound on the number of rounds scheme 2 needs to complete Broadcast.

We first start with the phases $i \leq \left\lceil \log \frac{n-f}{2} \right\rceil$ such that $\ln n \leq 2^{i-2}$ and the phases $i > \left\lceil \log \frac{n}{2} \right\rceil, i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ such that $\ln n \leq 2^{i-1}$. In that case, phase i needs at most $24 \cdot c \cdot n$ sampled edges to end with probability larger than $1 - n^{-c}$. We need at most $\left\lceil \frac{24 \cdot c \cdot n}{m} \right\rceil = \left\lceil \frac{24 \cdot c}{m/n} \right\rceil$ rounds to gather that many edges, and thus phase i ends in $\left\lceil \frac{24 \cdot c}{m/n} \right\rceil$ rounds with probability greater than $1 - n^{-c}$. There are at most $\lceil \log n \rceil$ such phases, and thus over all phases we require at most $O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$.

Let us now analyze the phases $i \leq \left\lceil \log \frac{n-f}{2} \right\rceil$ where $\ln n > 2^{i-2}$ (And symmetrically phases $i > \left\lceil \log \frac{n}{2} \right\rceil, i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ where $\ln n > 2^{j-2}$). In that case, phase i needs at most $24 \cdot c \cdot \ln n \cdot \frac{n}{2^{i-2}}$ sampled edges to end with probability larger than $1 - n^{-c}$. We need at most $\left\lceil \frac{24 \cdot c \cdot \ln n \cdot \frac{n}{2^{i-2}}}{m} \right\rceil = \left\lceil \frac{24 \cdot c \cdot \ln n}{m/n \cdot 2^{i-2}} \right\rceil$ rounds to gather that many edges, and thus phase i ends in $\left\lceil \frac{24 \cdot c \cdot \ln n}{m/n \cdot 2^{i-2}} \right\rceil$ rounds with probability greater than $1 - n^{-c}$. Summing the number of rounds over all such phases we get:

$$\sum_i \left\lceil \frac{24 \cdot c \cdot \ln n}{m/n \cdot 2^{i-2}} \right\rceil \leq \sum_i \frac{24 \cdot c \cdot \ln n}{m/n \cdot 2^{i-2}} + 1 \leq \frac{72 \cdot c \cdot \ln n}{m/n} + \log n = O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$$

The probability of success $p \geq 1 - n^{-c} \log n$ is simply a union bound on the number of phases. □

We can expand this result to all-to-all Broadcast, using a simple union-bound:

Corollary 4.9.52. *For any $c \geq 1$, in scheme 2, and therefore scheme 1, All-to-All Broadcast completes within $O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$ rounds with probability $p \geq 1 - n^{-c+1} \log n$.*

Theorem 4.9.53. *Let $\mathcal{A}$ be a distributed synchronous algorithm that runs on a static clique in T time, where $T \leq \alpha n^x$ for some constant $\alpha \in \mathcal{R}_+, x \in \mathbb{N}$, and has a probability of success p. Assume $\mathcal{A}$ is robust to f Byzantine nodes, and $f < \frac{2}{3}n$. Then, assuming cryptographic tools that allow nodes to sign and encrypt messages, there exists a distributed algorithm $\mathcal{A}'$ that runs on directed ErdsRényi graphs with m edges in $O\left(T \left\lceil \frac{c}{m/n} \right\rceil \log n\right)$ time, and has a probability of success $p' \geq p(1 - \alpha n^{1+x-c} \log n)$, for any $c \geq 1 + x$. Moreover, $\mathcal{A}'$ is robust to f Byzantine nodes.*

Proof. The algorithm $\mathcal{A}'$ works as follows: Each round of $\mathcal{A}$ will be simulated using $O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$ rounds of directed ErdsRényi graphs. Each round in $\mathcal{A}$ can be seen as (1) a *computation step*, where each node decides what message to send to every other node, and (2) a *communication step*, where each node sends the messages and receives the messages the other nodes have sent it. In $\mathcal{A}'$ the computation step is unchanged, except that each node uses the PKI to sign and encrypt the messages. The communication step on the other hand is extended over $O\left(\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$ rounds, in which every (honest) node simply forwards all received messages to all its out-neighbors. Theorem 4.9.52 ensures that with probability at most $n^{1-c} \log n$, at least one honest node fails to send a message to all other honest nodes. By a union bound over the T rounds, the probability that at least one message fails to be delivered is at most $\alpha n^{1+x-c} \log n$. By taking its complement, the probability that all messages are delivered in time before the next round's computation step is larger than $1 - \alpha n^{x+1-c} \log n$. The probability that $\mathcal{A}'$ succeeds is then $p' \geq p(1 - \alpha n^{x+1-c} \log n)$ □

We now give two applications of this theorem, namely Reliable Broadcast and Byzantine Consensus.

Theorem 4.9.54. *For any $c \geq 1$, and $f \leq \frac{2}{3}n - 1$, in the directed ErdsRényi graphs model, there exists an algorithm for Reliable Broadcast, that is robust to f Byzantine nodes, that runs in $O\left((f+1)\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$ rounds, and succeeds with probability $p \geq 1 - n^{2-c} \log n$.*

Proof. Dolev and Strong [50] have given an algorithm that solves reliable Broadcast, is robust to f Byzantine nodes, and runs in $T = f + 1$ rounds. Since $T \leq n$, we can apply Theorem 4.9.53 with $x = 1, \alpha = 1$, and we get the desired result. □

Theorem 4.9.55. *For any $c \geq 1$, in the directed ErdsRényi graphs model, there exists an algorithm for Byzantine Consensus, that is robust to f Byzantine nodes as long as $f < \frac{n}{3}$, that runs in $O\left((f+1)\left\lceil \frac{c}{m/n} \right\rceil \log n\right)$ rounds, and succeeds with probability $p \geq 1 - 2n^{2-c} \log n$.*

Proof. Berman, Garay and Perry [22] have given an algorithm (known as the King's algorithm) that solves reliable Broadcast, is robust to f Byzantine nodes, and runs in $T = 3(f+1)$ rounds. Since $T \leq 2n$, we can apply Theorem 4.9.53 with $x = 1, \alpha = 2$, and we get the desired result. □

4.9.9 Adversarial Edges in directed ErdsRényi graphs

In this section, we will study the case where in each round, an adversary chooses k edges in each round, then $m-k$ edges are chosen among the remaining edges. We restrict k to be smaller than $\frac{3}{4}n^2$, so that the adversary is not forced to choose an edge from an informed node to an uninformed one. In fact, in that case, the edges chosen by the adversary do not matter (as long as she doesn't choose an edge from an informed node to an uninformed one), as the edges chosen by the adversary cannot "protect" uninformed nodes as in the case of the trees. We will, thus, in the rest of this section, simply assume that k edges have been removed from the pool of possible edges, none of them being an edge from an informed node to an uninformed node.

As before, we will analyze three different schemes.

In *scheme 1*, in each round, $m-k$ edges are chosen uniformly at random *without* replacement among the n^2-k possible edges.

In *scheme 2*, in each round, $m-k$ edges are chosen uniformly at random *with* replacement among the n^2-k possible edges. This can only result in more rounds than scheme 1 as fewer disjoint edges are chosen in each round compared to scheme 1.

In *scheme 3*, we start with a unique informed node, let say node 1. There are two types of phases for a total of $2\left\lceil \log \frac{n}{2} \right\rceil$ phases. In each phase i with $1 \leq i \leq \left\lceil \log \frac{n}{2} \right\rceil$, we have $N_i = 2^{i-1}$ informed nodes in I_i at the beginning of the phase and the goal is to double that number within the phase. We set $E = \varnothing$ and add to E one edge at a time, sampled with replacement, until $|I_i \cup \mathcal{N}_E(I_i)| = \min\{2^i, \left\lceil \frac{n}{2} \right\rceil\}$. We then set $I_{i+1} = I_i \cup \mathcal{N}_E(I_i)$. Note that at the end of phase $i = \left\lceil \log \frac{n}{2} \right\rceil$, $N_{i+1} = \left\lceil \frac{n}{2} \right\rceil$ and this scheme is independent of m.

Then in each phase $i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ with $0 \leq j < \left\lceil \log \frac{n}{2} \right\rceil$, we initially have N_i informed nodes in I_i, and $\min\{2^j, \left\lfloor \frac{n}{2} \right\rfloor\}$ uninformed nodes in S_i and the goal is to half the number of uninformed nodes in each phase. We set $E = \varnothing$ and add to E one edge at a time sampled with replacement, until $|S_i \setminus \mathcal{N}_E(I_i)| = 2^{j-1}$. We then set $I_{i+1} = I_i \cup \mathcal{N}_E(I_i)$.

We start by analyzing scheme 3:

Lemma 4.9.56. *For any $c \geq 1$, any phase $i \leq \left\lceil \log \frac{n}{2} \right\rceil$ needs at most $8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}\frac{(n^2-k)}{2^{i-1}n}$ sampled edges to complete with probability $p \geq 1 - n^{-c}$.*

Proof. We first note that in phase i with $i \leq \left\lceil \log \frac{n}{2} \right\rceil$, the probability that an edge that is sampled increases $I_i \cup \mathcal{N}_E(I_i)$ is at least $\frac{2^{i-2}n}{(n^2-k)}$. Indeed, $|I_i \cup \mathcal{N}_E(I_i)| \leq \frac{n}{2}$ (otherwise the phase would have ended) and there are $|I_i| \cdot |[n] \setminus (I_i \cup \mathcal{N}_E(I_i))| \geq 2^{i-2}n$ edges that can increase $I_i \cup \mathcal{N}_E(I_i)$. Thus, picking any of those edges, out of (n^2-k) possible ones, has probability at least $\frac{2^{i-2}n}{(n^2-k)}$.

Next, we remark that if we sample $\frac{(n^2-k)}{2^{i-2}n}$ edges, then the probability that at least one of those edges increases $I_i \cup \mathcal{N}_E(I_i)$ is at least $\frac{1}{2}$. Indeed, the probability that all of those edges do not increase the size of $I_i \cup \mathcal{N}_E(I_i)$ is $(1 - \frac{2^{i-2}n}{(n^2-k)})^{\frac{(n^2-k)}{2^{i-2}n}} \leq e^{-1} \leq \frac{1}{2}$. We will, thus, group the sampled edges into disjoint "buckets" of $\frac{(n^2-k)}{2^{i-2}n}$ consecutively sampled edges.

We then use the fact that we only need 2^{i-1} edges that increase $I_i \cup \mathcal{N}_E(I_i)$ to end phase i. If we take $8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}$ buckets of $\frac{(n^2-k)}{2^{i-2}n}$ edges each, then applying Hoeffding's inequality, we have:

$$\mathbb{P}(|I_i \cup \mathcal{N}_E(I_i)| \leq 2^i - 1) \leq \exp\left(-2 \cdot 8c \cdot \max\{\ln n, 2^{i-2}\}\left(\frac{1}{2} - \frac{2^{i-1}}{8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}}\right)^2\right)$$
$$\leq \exp\left(-16 \cdot c \cdot \ln n \left(\frac{1}{2} - \frac{1}{4}\right)^2\right) = n^{-c}$$

which proves that with probability $p \geq 1 - n^{-c}$, phase i ends after sampling at most $8 \cdot c \cdot \max\{\ln n, 2^{i-2}\}\frac{(n^2-k)}{2^{i-2}n}$ edges. □

We have a symmetric result for the second phase:

Lemma 4.9.57. *For any $c \geq 1$, any phase i with $i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ for $0 \leq j < \left\lceil \log \frac{n}{2} \right\rceil$ needs at most $8 \cdot c \cdot \max\{\ln n, 2^{j-2}\}\frac{(n^2-k)}{2^{j-1}n}$ sampled edges to complete with probability $p \geq 1 - n^{-c}$.*

Proof. We first note that, by the stopping condition for phase $\left\lceil \frac{n}{2} \right\rceil$, for any phase i with $i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ for $0 \leq j < \left\lceil \log \frac{n}{2} \right\rceil$, it holds that $N_i \geq \left\lceil \frac{n}{2} \right\rceil$. We first show that in phase i, the probability that a sampled edge decreases $S_i \setminus \mathcal{N}_E(I_i)$ is at least $\frac{2^{j-2}n}{(n^2-k)}$. Indeed, $|[n] \setminus (I_i \cup \mathcal{N}_E(I_i))| = |S_i \setminus \mathcal{N}_E(I_i)| > 2^{j-1}$ (otherwise the phase would have ended) and, thus, there are $N_i \cdot |[n] \setminus (I_i \cup \mathcal{N}_E(I_i))| \geq 2^{j-2}n$ edges that would decrease $S_i \setminus \mathcal{N}_E(I_i)$. Thus, picking any of those edges, out of $(n^2 - k)$ possible ones, has probability at least $\frac{2^{j-2}n}{(n^2-k)}$.

The rest of the proof is symmetrical to the previous one. □

This allows us to prove the following result on scheme 2:

Theorem 4.9.58. *For any $c \geq 1$, in scheme 2, and therefore scheme 1, Broadcast completes within $O\left(\left\lceil \frac{c \cdot (n^2-k)}{(m-k)n} \right\rceil \log n\right)$ rounds with probability $p \geq 1 - n^{-c} \log n$.*

Proof. To see this, we are going to simulate scheme 3 with scheme 2. The main idea is a follows: If a phase (of scheme 3) requires x edges, sampled with replacement, in order to end, and in scheme 2 each round samples y edges with replacement, then in $\left\lceil \frac{x}{y} \right\rceil$ rounds, scheme 2 samples at least x edges, and, thus, we can simulate the phase in scheme 3 with $\left\lceil \frac{x}{y} \right\rceil$ rounds of scheme 2. The only difference is that scheme 2 groups the edges into rounds to make intermediate progress, whereas scheme 3 only forwards the information at the end of the phase, all at once. This implies that in scheme 2 each phase is faster than the corresponding one in scheme 3, and any upper bound we get with this analysis will thus be an upper bound on the number of rounds scheme 2 needs to complete Broadcast.

We first start with the phases $i \leq \left\lceil \log \frac{n}{2} \right\rceil$ such that $\ln n \leq 2^{i-2}$ and the phases $i > \left\lceil \log \frac{n}{2} \right\rceil$ with $i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ for $j \geq 1$ such that $\ln n \leq 2^{j-2}$. In that case, phase i needs at most $8 \cdot \frac{c}{n} \cdot (n^2 - k)$ sampled edges to end with probability larger than $1 - n^{-c}$. We need at most $\left\lceil \frac{8 \cdot c \cdot (n^2-k)}{n(m-k)} \right\rceil$ rounds to gather that many edges, and thus phase i ends in $\left\lceil \frac{8 \cdot c \cdot (n^2-k)}{n(m-k)} \right\rceil$ rounds with probability greater than $1 - n^{-c}$. There are at most $\lceil \log n \rceil$ such phases, and thus over all phases we require at most $O\left(\left\lceil \frac{c \cdot (n^2-k)}{(m-k)n} \right\rceil \log n\right)$.

Let us now analyze the phases $i \leq \left\lceil \log \frac{n}{2} \right\rceil$ where $\ln n > 2^{i-2}$ and symmetrically phases $i > \left\lceil \log \frac{n}{2} \right\rceil, i = 2\left\lceil \log \frac{n}{2} \right\rceil - j$ for $j \geq 1$ such that $\ln n > 2^{j-2}$. In that case, phase i needs at most $8 \cdot c \cdot \ln n \cdot \frac{n^2-k}{2^{i-2}n}$ sampled edges to end with probability larger than $1 - n^{-c}$. We need at most $\left\lceil \frac{8 \cdot c \cdot \ln n \cdot \frac{(n^2-k)}{2^{i-2}n}}{m-k} \right\rceil = \left\lceil \frac{8 \cdot c \cdot \ln n \cdot (n^2-k)}{(m-k)2^{i-2}n} \right\rceil$ rounds to gather that many edges, and thus phase i ends in $\left\lceil \frac{8 \cdot c \cdot \ln n \cdot (n^2-k)}{(m-k)2^{i-2}n} \right\rceil$ rounds with probability greater than $1 - n^{-c}$. Summing the number of rounds over all such phases we get:

$$\begin{aligned}\sum_i \left\lceil \frac{8 \cdot c \cdot \ln n \cdot (n^2-k)}{(m-k)2^{i-2}n} \right\rceil &\leq \sum_i \left(\frac{8 \cdot c \cdot \ln n \cdot (n^2-k)}{(m-k)2^{i-2}n} + 1 \right) \\ &\leq \frac{32 \cdot c \cdot \ln n \cdot (n^2-k)}{(m-k)n} + \log n = O\left(\left\lceil \frac{c \cdot (n^2-k)}{(m-k)n} \right\rceil \log n \right)\end{aligned}$$

The probability of success $p \geq 1 - n^{-c} \log n$ is simply a union bound on the number of phases. □

References

[3] Mohamad Ahmadi, Fabian Kuhn, Shay Kutten, Anisur Rahaman Molla, and Gopal Pandurangan. "The Communication Cost of Information Spreading in Dynamic Networks". In: *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*. IEEE, 2019, pp. 368–378. DOI: 10.1109/ICDCS.2019.00044. URL: https://doi.org/10.1109/ICDCS.2019.00044.

[14] John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. "Towards robust and efficient computation in dynamic peer-to-peer networks". In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. Ed. by Yuval Rabani. SIAM, 2012, pp. 551–569. DOI: 10.1137/1.9781611973099.47. URL: https://doi.org/10.1137/1.9781611973099.47.

[22] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. "Towards Optimal Distributed Consensus (Extended Abstract)". In: *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. IEEE Computer Society, 1989, pp. 410–415. DOI: 10.1109/SFCS.1989.63511. URL: https://doi.org/10.1109/SFCS.1989.63511.

[31] Arthur Cayley. "A theorem on trees". In: *Quart. J. Math.* 23 (1889), pp. 376–378.

[33] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. "Approximate Consensus in Highly Dynamic Networks: The Role of Averaging Algorithms". In: *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*. Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Vol. 9135. Lecture Notes in Computer Science. Springer, 2015, pp. 528–539. DOI: 10.1007/978-3-662-47666-6_42. URL: https://doi.org/10.1007/978-3-662-47666-6%5C_42.

[34] Bernadette Charron-Bost and André Schiper. "The Heard-Of model: computing in distributed systems with benign faults". In: *Distributed Comput.* 22.1 (2009), pp. 49–71. DOI: 10.1007/s00446-009-0084-6. URL: https://doi.org/10.1007/s00446-009-0084-6.

[41] Andrea E. F. Clementi, Pierluigi Crescenzi, Carola Doerr, Pierre Fraigniaud, Francesco Pasquale, and Riccardo Silvestri. "Rumor spreading in random evolving graphs". In: *Random Struct. Algorithms* 48.2 (2016), pp. 290–312. DOI: 10.1002/rsa.20586. URL: https://doi.org/10.1002/rsa.20586.

[42] Étienne Coulouma, Emmanuel Godard, and Joseph G. Peters. "A characterization of oblivious message adversaries for which Consensus is solvable". In: *Theor. Comput. Sci.* 584 (2015), pp. 80–90. DOI: 10.1016/j.tcs.2015.01.024. URL: https://doi.org/10.1016/j.tcs.2015.01.024.

[43] Frank Den Hollander. "Probability theory: The coupling method". In: *Lecture notes available online (https://pub.math.leidenuniv.nl/probability/lecturenotes/CouplingLectures.pdf)* (2012).

[45] Michael Dinitz, Jeremy T. Fineman, Seth Gilbert, and Calvin Newport. "Smoothed Analysis of Information Spreading in Dynamic Networks". In: *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*. Ed. by Christian Scheideler. Vol. 246. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 18:1–18:22. DOI: 10.4230/LIPICS.DISC.2022.18. URL: https://doi.org/10.4230/LIPIcs.DISC.2022.18.

[46] Stefan Dobrev and Imrich Vrto. "Optimal Broadcasting in Hypercubes with Dynamic Faults". In: *Inf. Process. Lett.* 71.2 (1999), pp. 81–85. DOI: 10.1016/S0020-0190(99)00093-9. URL: https://doi.org/10.1016/S0020-0190(99)00093-9.

[47] Stefan Dobrev and Imrich Vrto. "Optimal Broadcasting in Tori with Dynamic Faults". In: *Parallel Process. Lett.* 12.1 (2002), pp. 17–22. DOI: 10.1142/S0129626402000781. URL: https://doi.org/10.1142/S0129626402000781.

[48] Benjamin Doerr. "Probabilistic Tools for the Analysis of Randomized Optimization Heuristics". In: *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*. Ed. by Benjamin Doerr and Frank Neumann. Natural Computing Series. Springer, 2020, pp. 1–87. DOI: 10.1007/978-3-030-29414-4_1. URL: https://doi.org/10.1007/978-3-030-29414-4%5C_1.

[49] Benjamin Doerr and Mahmoud Fouz. "Asymptotically optimal randomized rumor spreading". In: *International Colloquium on Automata, Languages, and Programming (ICALP)*. Springer. 2011, pp. 502–513.

[50] Danny Dolev and H. Raymond Strong. "Authenticated Algorithms for Byzantine Agreement". In: *SIAM J. Comput.* 12.4 (1983), pp. 656–666. DOI: 10.1137/0212045. URL: https://doi.org/10.1137/0212045.

[54] Rick Durrett and Dong Yao. "Susceptible–infected epidemics on evolving graphs". In: *Electronic Journal of Probability* 27 (2022), pp. 1–66.

[55] Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, Zhifeng Sun, and Emanuele Viola. "On the Complexity of Information Spreading in Dynamic Networks". In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*. Ed. by Sanjeev Khanna. SIAM, 2013, pp. 717–736. DOI: 10.1137/1.9781611973105.52. URL: https://doi.org/10.1137/1.9781611973105.52.

[60] Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Asymptotically Tight Bounds on the Time Complexity of Broadcast and Its Variants in Dynamic Networks". In: *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*. Ed. by Yael Tauman Kalai. Vol. 251. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 47:1–47:21. DOI: 10.4230/LIPICS.ITCS.2023.47. URL: https://doi.org/10.4230/LIPIcs.ITCS.2023.47.

[61] Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Brief Announcement: Broadcasting Time in Dynamic Rooted Trees is Linear". In: *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*. Ed. by Alessia Milani and Philipp Woelfel. ACM, 2022, pp. 54–56. DOI: 10.1145/3519270.3538460. URL: https://doi.org/10.1145/3519270.3538460.

[62] Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Broadcast and Consensus in Stochastic Dynamic Networks with Byzantine Nodes and Adversarial Edges". In: *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*. Ed. by Dan Alistarh. Vol. 319. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 21:1–21:15. DOI: 10.4230/LIPICS.DISC.2024.21. URL: https://doi.org/10.4230/LIPIcs.DISC.2024.21.

[63] Faith Ellen, Barun Gorain, Avery Miller, and Andrzej Pelc. "Constant-Length Labeling Schemes for Deterministic Radio Broadcast". In: *ACM Trans. Parallel Comput.* 8.3 (2021), 14:1–14:17. DOI: 10.1145/3470633. URL: https://doi.org/10.1145/3470633.

[65] Patrick T Eugster, Rachid Guerraoui, A-M Kermarrec, and Laurent Massoulié. "Epidemic information dissemination in distributed systems". In: *Computer* 37.5 (2004), pp. 60–67.

[67] Pierre Fraigniaud and Emmanuel Lazard. "Methods and problems of communication in usual networks". In: *Discret. Appl. Math.* 53.1-3 (1994), pp. 79–133. DOI: 10.1016/0166-218X(94)90180-5. URL: https://doi.org/10.1016/0166-218X(94)90180-5.

[68] Matthias Függer, Thomas Nowak, and Kyrill Winkler. "On the radius of nonsplit graphs and information dissemination in dynamic networks". In: *Discret. Appl. Math.* 282 (2020), pp. 257–264. DOI: 10.1016/j.dam.2020.02.013. URL: https://doi.org/10.1016/j.dam.2020.02.013.

[69] Hugo Rincon Galeana, Ami Paz, Stefan Schmid, Ulrich Schmid, and Kyrill Winkler. "The Time Complexity of Consensus Under Oblivious Message Adversaries". In: *14th Innovations in Theoretical Computer Science (ITCS)*. 2023.

[70] Hugo Rincon Galeana, Ulrich Schmid, Kyrill Winkler, Ami Paz, and Stefan Schmid. "Topological Characterization of Consensus Solvability in Directed Dynamic Networks". In: *arXiv preprint arXiv:2304.02316* (2023).

[73] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. "Distributed MST and Routing in Almost Mixing Time". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*. Ed. by Elad Michael Schiller and Alexander A. Schwarzmann. ACM, 2017, pp. 131–140. DOI: 10.1145/3087801.3087827. URL: https://doi.org/10.1145/3087801.3087827.

[78] Spencer Greenberg and Mehryar Mohri. "Tight lower bound on the probability of a binomial exceeding its expectation". In: *Statistics & Probability Letters* 86 (2014), pp. 91–98.

[85] Wassily Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30.

[87] Juraj Hromkovi, Ralf Klasing, Burkhard Monien, and Regine Peine. "Dissemination of information in interconnection networks (broadcasting & gossiping)". In: *Combinatorial network theory*. Springer, 1996, pp. 125–212.

[98] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. "Distributed computation in dynamic networks". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. Ed. by Leonard J. Schulman. ACM, 2010, pp. 513–522. DOI: 10.1145/1806689.1806760. URL: https://doi.org/10.1145/1806689.1806760.

[104] Linyuan Lu, Austin Mohr, and László Székely. "Quest for negative dependency graphs". In: *Recent Advances in Harmonic Analysis and Applications*. Springer, 2012, pp. 243–258.

[106] Uri Meir, Ami Paz, and Gregory Schwartzman. "Models of Smoothing in Dynamic Networks". In: *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*. Ed. by Hagit Attiya. Vol. 179. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 36:1–36:16. DOI: 10.4230/LIPICS.DISC.2020.36. URL: https://doi.org/10.4230/LIPIcs.DISC.2020.36.

[110] James D Murray et al. *Mathematical biology I: an introduction*. 2002.

[116] Jim Pitman. "Coalescent random forests". In: *Journal of Combinatorial Theory, Series A* 85.2 (1999), pp. 165–193.

[119] Jonathan DH Smith. *Introduction to abstract algebra*. Vol. 31. CRC Press, 2015.

[129] Martin Zeiner, Manfred Schwarz, and Ulrich Schmid. "On linear-time data dissemination in dynamic rooted trees". In: *Discret. Appl. Math.* 255 (2019), pp. 307–319. DOI: 10.1016/j.dam.2018.08.015. URL: https://doi.org/10.1016/j.dam.2018.08.015.

CHAPTER 5

An Almost Tight Lower Bound for Plurality Consensus with Undecided State Dynamics in the Population Protocol Model

5.1 Abstract

We revisit the majority problem in the population protocol communication model, as first studied by Angluin et al. (Distributed Computing 2008). We consider a more general version of this problem known as plurality consensus, which has already been studied intensively in the literature. In this problem, each node in a system of n nodes, has initially one of k different opinions, and they need to agree on the (relative) majority opinion. In particular, we consider the important and intensively studied model of Undecided State Dynamics.

Our main contribution is an almost tight lower bound on the stabilization time: we prove that there exists an initial configuration, even with bias $\Delta = \omega(\sqrt{n \log n})$, where stabilization requires $\Omega(kn \log \frac{\sqrt{n}}{k \log n})$ interactions, or equivalently, $\Omega(k \log \frac{\sqrt{n}}{k \log n})$ parallel time for any $k = o\left(\frac{\sqrt{n}}{\log n}\right)$. This bound is tight for any $k \leq n^{\frac{1}{2}-\epsilon}$, where $\epsilon > 0$ can be any small constant, as Amir et al. (PODC'23) gave a $O(k \log n)$ parallel time upper bound for $k = O\left(\frac{\sqrt{n}}{\log^2 n}\right)$.

5.2 Introduction

Population protocols are a simple and natural computational framework, in which n anonymous nodes (also called agents) communicate and interact with each other to solve a predefined problem in a distributed manner. In the underlying communication model, a scheduler selects in each discrete time step two nodes for interaction. The selected nodes exchange their current states and each of them changes its own state according to the transition function defined by the population protocol. The set of nodes is expected to eventually stabilize in a final configuration that is given by the problem definition. In the plurality consensus problem, as

considered in this paper, every node has at the beginning an opinion assigned from a set of k different opinions, and in the final configuration all nodes agree on one of the initial opinions.

In the model introduced by Angluin et al. [10] the population is represented by the nodes of a graph and a scheduler can only choose nodes connected by an edge for interaction. The usual complexity measures in which one is interested are the cardinality of the state space of the transition function and the time needed for the population to stabilize. The time is defined as the number of interactions until a stable configuration is reached. As in previous work, we are also interested in the so-called parallel time, which corresponds to the number of interactions divided by the population size n.

Population protocols have various applications. In their original paper, Angulin et al. [10] motivated population protocols in the context of sensor networks where nodes perform simple computations. Other motivating examples are processes in chemical reaction networks [121]. Population protocols can also be implemented at the level of DNA molecules, as shown in [37]. Furthermore, Cardelli and Csiksz-Nagy [30] considered similarities between biochemical regulatory processes in living cells and population protocols. Population protocols highly influenced the way certain aspects of distributed computing evolved in recent years, for which in 2020 the original paper by Angluin et al. [10] has been awarded the Edsger W. Dijkstra Prize in Distributed Computing.

In the vast majority of the related papers, as in our paper as well, it is assumed that the graph modeling the population is a clique and a random scheduler is in place. That is, in each discrete time step two nodes are selected uniformly at random for interaction. One of the most prominent protocols for the plurality consensus problem is the Undecided State Dynamics, on which we focus in this paper. In its original, unconditional version, this protocol is simple and only uses $k+1$ different states (see the problem definition below for details). This protocol and variants of it have extensively been analyzed in different communication models. Although very recently an upper bound of $O(k \log n)$ on the parallel time has been derived [9], the question of whether this bound is tight is still open.

5.2.1 Model and problem

A population protocol and its time performance can be formalized as follows (cf. [20]). Let V denote the set of agents in the population, and let $n = |V|$. Let Σ be the set of states of the protocol, whose cardinality may grow with n. Two interacting nodes change their states according to a deterministic function $f : \Sigma^2 \mapsto \Sigma^2$. That is, $f(q', q'') = (r', r'')$ describes the following transition: if a node in state q' interacts with a node in state q'', then the first node changes its state to r' and the second node to r''. A population protocol is also assigned an *output function* $\gamma : \Sigma \to \Gamma$, which maps every state to an output value. The set Γ may be the same as Σ.

In this paper we analyze plurality consensus (aka the k-majority problem): Each agent i has as an input an initial opinion $s_i \in [k]$, and the goal of the agents is to decide which opinion was the (relative) majority at the start of the computation. The transition function we consider is given by the (unconditional) Undecided State Dynamics: Σ consists of $k + 1$ states, the original k opinions, and an extra one, $\perp$, that represents the state of being undecided. Then $f(s_1, s_2) = (\perp, \perp)$ if $s_1 \neq s_2$, and $s_1, s_2 \in [k]$, $f(s, \perp) = (s, s)$ for any $s \in [k]$. Otherwise,

f is just the identity function. In other words, when two agents with different opinions meet, they both become undecided, but when a decided agent meets an undecided one, the latter takes on the opinion of the former. Note that in the case of the Undecided State Dynamics the set $\Gamma = \Sigma$ and the output function γ is the identity. We assume that the interaction graph is a clique, and at each time step, two nodes are selected for interaction, which are chosen uniformly at random (without replacement), independently of the other time steps. We ask how long it takes for the system to stabilize, in the particular case where the majority opinion starts with an initial additional bias of $\Omega(\sqrt{n \log n})$. Note that such a bias is probably needed in order to guarantee that w.h.p. the opinion with the largest (relative) initial support wins (cf. [9, 15]).

5.2.2 Related work

Computations over dynamic networks (such as sensor networks) have already been studied intensively in the literature for various models [1, 99, 108]. Probably the most studied problems in the framework of population protocols are leader election and (exact) majority. For two recent surveys, which focus on these problems, we refer the reader to [7] and [64]. As in this paper we analyze plurality consensus, we only present the most relevant results on the related majority problem and adapt the description of [64] to outline the related work on majority in population protocols. In this problem, at the beginning every node is in one of two states (called e.g. A and B). In exact majority, the final opinion has to be the one with the largest initial support, even if at the beginning the difference between the support of the two was just 1. In approximate majority this requirement is weaker: the initial majority should only win with high probability if there is a sufficiently large initial bias between the two opinions (usually this bias is of order $\Omega(\sqrt{n \log n})$).

For the majority problem (as well as leader election), there are a number of results, which present lower bounds on the number of states under certain time requirements, or bound the (stabilization) time under specific assumptions w.r.t. the number of states. Furthermore, several majority and leader election algorithms have been derived, which upper bound the number of states as well as the stabilization time.

Two early papers by Draief and Vojnović [53] and Mertzios et al. [107] consider population protocols for exact majority. They studied (almost) the same four-state protocol, which has a polynomial stabilization time (with high probability[1] and in expectation) on any graph.

In an early paper, Angluin et al. [11] presented population protocols with a constant number of states for several different functions. The protocols they propose are only correct with high probability and they assume that a designated leader is available from the start of the computation, which synchronizes the nodes. Their exact majority protocol has a w.h.p. stabilization time of $O(\log^2 n)$, and it alternates between so-called cancellation and duplication phases, an idea used in many subsequent papers.

Note that any protocol for exact majority which uses a constant number of states (as the four-state protocol described above) is in general slow. However, if the initial imbalance between the support of A and B is large, then the four-state protocol stabilizes fast. In order to

[1] With high probability or w.h.p. means with probability at least $1 - n^{-\Omega(1)}$, where n is the number of agents.

increase the initial imbalance, Alistarh et al. [8] multiplied the opinion on each node by some (large) integer. Based on this idea, Alistarh et al. [4] achieved a stabilization time of $O(\log^3 n)$, w.h.p. and in expectation, by utilizing $O(\log^2 n)$ states.

Bilke et al. [24] extended the cancellation-duplication framework from [11] to the leaderless case, provided that the agents have enough states to store the number of interactions they performed so far. The stabilization time of their majority protocol is $O(\log^2 n)$ w.h.p. and in expectation, and it utilizes $O(\log^2 n)$ states.

On the lower bound side, Alistarh et al. [5] proved that any exact majority protocol with expected stabilization time $O(n^{1-\epsilon})$ (ϵ can be any positive constant), which satisfies two natural properties called *monotonicity* and *output dominance*, requires $\Omega(\log n)$ states. They also presented an algorithm with $\Theta(\log n)$ states and $O(\log^2 n)$ stabilization time (w.h.p. and in expectation). Monotonic protocols have the property that their running time does not increase if they are run on a smaller number of agents. Output dominance means that "if the positive counts of states in a stable configuration are changed, then the protocol will stabilize to the same output" (cf.[64]). The (w.h.p. and expected) stabilization time has subsequently been improved to $O(\log^{5/3} n)$ in [20], to $O(\log^{3/2} n)$ in [19], and finally to $O(\log n)$ in [51], by keeping the number of states at $O(\log n)$.

All the results above were derived for *stabilization* time, and the lower bounds do not hold for the so-called *convergence* time. The convergence time is the time required by the protocol to reach a configuration with the correct output property; however, the system may leave this configuration with a small probability. In contrast, if the system stabilizes, then the output of the system does not change anymore.[2] Kosowski and Uznański [97] and Berenbrink et al. [21] derived algorithms with polylogarithmic *convergence* time, which use $o(\log n)$ states. As outlined in [20], in [97] the authors presented a programming framework that leads to protocols which require only $O(1)$ states and converge in polylogarithmic time (in expectation), but they are only correct w.h.p. These protocols can be changed so that they are always correct by either allowing $O(\log \log n)$ states, while the convergence time still remains polylogarithmic, or by allowing $O(n^\epsilon)$ convergence time, while keeping the number of states constant. In [21] the authors presented an always correct protocol with a w.h.p. convergence time of $O(\log^2 n / \log s)$ and $\Theta(s + \log \log n)$ states, and an always correct protocol with w.h.p. stabilization time of $O(\log^2 n / \log s)$ and $O(s \cdot \log n / \log s)$ states, where $s \in [2, n]$.

One research direction in plurality consensus focuses on the state complexity (regardless of the time complexity) of protocols, which are required to *always* determine the plurality opinion. While clearly at least k states are required to encode k opinions, [114] shows that always correct plurality consensus needs even $\Omega(k^2)$ states. The protocol of [72] utilizes $O(k^{11})$ states, which can be improved to $O(k^6)$ provided that a total ordering among the opinions exists. Clearly, the lower bound of $\Omega(k^2)$ only holds, if it is required that the correct plurality opinion is determined with probability 1. If such strong guarantees are not required, then the number of states can be much smaller. In [15] a synchronized variant of the Undecided State Dynamics has been presented that reaches consensus w.h.p. in $O(\log^2 n)$ parallel time using $O(k \log n)$ states. However, this protocol solves *approximate* plurality consensus, i.e., if the initial bias is $\Omega(\sqrt{n \log n})$, then w.h.p. the opinion with the initially largest support wins, otherwise a so-called significant opinion wins w.h.p. In the case $k = 2$, the unconditional Undecided

[2] See e.g. [21] for details. In the Undecided State Dynamics, convergence and stabilization are equivalent.

State Dynamics has a w.h.p. and expected stabilization time of $O(\log n)$ [40]. Recently, Amir et al. [9] analyzed the unconditional Undecided State Dynamics for the plurality consensus problem and showed that their protocol stabilizes w.h.p. within $O(k \log n)$ parallel time for any initial configuration as long as $k = O(\sqrt{n}/\log^2 n)$. The question of whether this bound on the stabilization time is tight is still open.

We should note that the unconditional Undecided State Dynamics has extensively been analyzed in the Gossip communication model. In this model (which can be seen as a synchronous variant of the population protocol model) in each discrete time step, every node randomly chooses another node for interaction to perform a state transition. Becchetti et al. defined the concept of *monochromatic distance* $\mathrm{md}(c)$ of a configuration c and showed that in this model the time needed to reach a final configuration is $O(\mathrm{md}(c) \log n)$ w.h.p., where c is the initial configuration in the population [18]. They also derived a lower bound that is asymptotically tight up to a $\log n$ factor.

As described by Amir et al. [9], the differences in how nodes are scheduled for interaction in the population protocol model and the Gossip model, respectively, cause the Undecided State Dynamics to "exhibit significant qualitative differences when run in either setting, even in the case when $k = 2$". One of the reasons for these differences is the fact that while in the Gossip model in each step a node may change its opinion only once, and each node is selected for interaction, in the population protocol model a node may change its opinion up to $\Omega(\log n)$ many times in n consecutive interactions (which corresponds to one parallel round) while a constant fraction of nodes is not even selected for interaction. Thus, there are so far no general analysis techniques that allow us to transfer results from one model to the other one (cf. [40]).

5.2.3 Contribution

Our main contribution is an almost tight lower bound on the stabilization time of Undecided State Dynamics for plurality consensus in population protocols. Specifically, we show that the time needed to stabilize is $\Omega\left(kn \log \frac{\sqrt{n}}{k \log n}\right)$ interactions, or in $\Omega\left(k \log \frac{\sqrt{n}}{k \log n}\right)$ parallel time[3], in the case where $k = o(\frac{\sqrt{n}}{\log n})$. In our proofs we assume that our initial configuration has bias at most $O(\sqrt{n}/(k \log n))^{1/4} \sqrt{n \log n}$. Interestingly, this includes even an initial bias of $\omega(\sqrt{n \log n})$. This bound is tight for any $k \leq n^{\frac{1}{2}-\epsilon}$, where $\epsilon > 0$ can be any small constant, as Amir, Aspnes, Berenbrink, Biermeier, Hahn, Kaaser, Lazarsfeld [9] gave a $O(k \log n)$ parallel time upper bound. For larger values of k, as any initial configuration that is valid for k_0 is also valid for $k \geq k_0$, we can simply plug in $k_0 = \frac{\sqrt{n}}{\log n \log\log n}$ to get a $\Omega\left(\frac{\sqrt{n} \log\log\log n}{\log n \log\log n}\right)$ parallel time lower bound.

Our analysis is based on a precise characterization of the number of undecided nodes over time, using drift analysis and random walks. Our technical approach and its novelty will be discussed in more details in the next section.

[3] Recall that the parallel time is equal to the number of interactions divided by n.

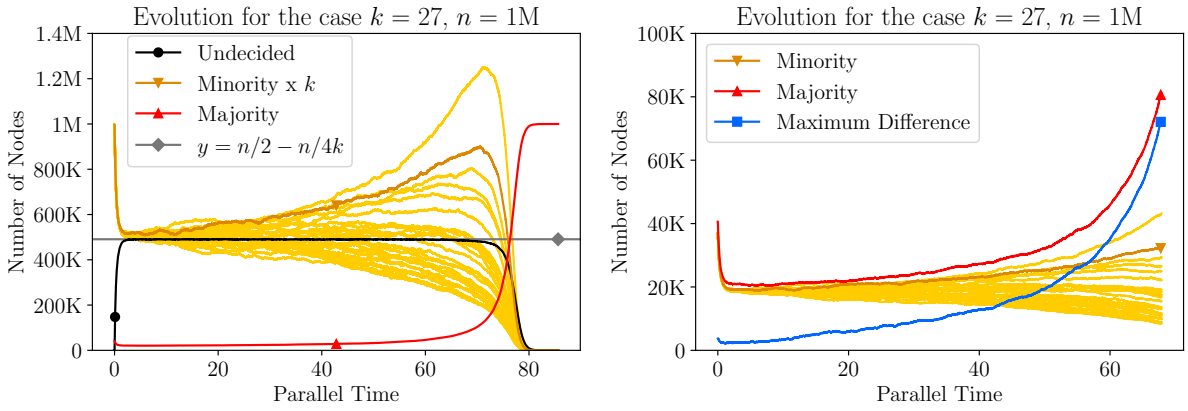


Figure 5.1: Intuition for stabilization of plurality consensus in undecided state dynamics. On the left figure, we scale up the minority opinions by a k factor for visibility. In this experiment, $n = 1,000,000$ agents and $k = \frac{\sqrt{n}}{\log n \log\log n}$. In the initial configuration $k-1$ opinions had the same support, while opinion 1 had an additive advantage of $\sqrt{n \log n}$. Out of all the minority opinions, we plot one with a darker color for visibility.

5.2.4 Notation and organization

Throughout the paper, $\mathbf{x}$ represents a configuration the system can be in: $\mathbf{x} = (x_1, \ldots, x_k, u)$, where x_i is the number of agents with opinion i for all i, and u is the number of undecided agents. We denote by $\mathbf{x}(t)$ the configuration of the system after the t-th interaction, and $\mathbf{x}(0)$ is the initial configuration. $x_i(t)$ is the number of agents with opinion i after interaction t. Similarly, $u(t)$ is the number of undecided nodes after t interactions. Accordingly, $x_i(0)$ is the initial number of nodes with opinion i (clearly, $u(0) = 0$). We assume that the opinions are initially ordered from most common opinion to least common, i.e, $x_1(0) \geq x_2(0) \geq \cdots \geq x_k(0)$.

The remainder of this paper is organized as follows. We first provide a technical overview and an empirical motivation in Section 5.3. Our lower bound analysis is presented in Section 5.4. We conclude and discuss open research questions in Section 5.5. Some technical details are deferred to the Appendix.

5.3 Motivation and Technical Overview

Before we delve into the technical details of our lower bound proof, let us provide some intuition about how agent opinions propagate across the distributed system, and how majority and minority opinions evolve accordingly. Based on this intuition and the observed challenges, we will then give an overview of our analytical approach and its key ideas.

To illustrate the evolution of opinions and the stabilization behavior, Figure 5.1 (left) plots a typical simulation run over parallel time. In this example, all minority opinions are initially equally frequent, while the majority one has an initial additional bias of $\sqrt{n \log n}$. The majority opinion is plotted in red. The minority opinions are plotted in yellow, except for a random one which we select as an example and which we highlight in orange; for better visibility we multiplied the number of each minority opinion by k.

We make several observations which highlight parts of the complexity of the problem. First, note that different minority opinions evolve differently. In particular, not all minority opinions are strictly decreasing over time, but many are actually increasing over a long time period. In the example in Figure 5.1, one opinion even surpasses its initial count. On the other hand, the majority opinion in this example (which is typical for many runs) remains low for a long time during stabilization, but then increases quickly toward the end. We can also see that the number of undecided opinions stays close to $\frac{n}{2} - \frac{n}{4k}$ throughout the execution.

In Figure 5.1 (right), we zoom in on the time period for which it takes x_1 to double from its initial number of nodes. We can see that it takes most of the stabilization time to reach this configuration, as the system needs around 70 parallel time to do so, after which only 20 more rounds are required to fully stabilize.

These simulations also provide some intuitive motivation for our choices for analyzing the problem theoretically in the next section. We will prove that the number of undecided nodes does not substantially exceed $\frac{n}{2} - \frac{n}{4k}$ with high probability, and we only consider the interactions before x_1 reaches $2x_1(0)$, which we will approximate as $2\frac{n}{k}$. One particular ingredient we consider in our analysis is the *maximum* difference between the majority opinion and minority ones (also plotted in Figure 5.1 (right)): $\max_{j \geq 2}\{x_1 - x_j\}$. The idea is that as long as this difference stays small, the system is very slow to change.

More concretely, for our analysis, we proceed as follows.

We first observe that a precise characterization of the number of undecided nodes $u(t)$ is key to understand the state dynamics as, intuitively, for any opinion i, the larger the number of undecided nodes, the more new nodes can the i-opinionated nodes "convert" to the opinion i, hoping to compensate for the i-opinionated nodes that meet nodes with other opinions and thus become themselves undecided. In fact, in their analysis of the stabilization time, Amir, Aspnes, Berenbrink, Biermeier, Hahn, Kaaser and Lazarsfeld [9] derived an upper bound and a lower bound on the number of undecided nodes throughout the interactions: $\frac{n}{2} - \frac{x_1}{2} \leq u(t) \leq \frac{n}{2}$ for any t after the first $n \log n$ interactions. More precisely, the evolution of the number of nodes in opinion i is driven by the number of undecided nodes. For each opinion, there exists a value u_i of the number of undecided nodes that acts as a threshold. That is, if the number of undecided nodes u is above u_i, then the corresponding opinion x_i increases, whereas if u is below the threshold, then x_i decreases. The threshold is a decreasing function in the number of nodes in opinion i: the larger x_i is, the smaller u_i is. One can see this in action in Figure 5.1 on the left. When $u(t)$ is still very small in the beginning, all of the opinions decrease quickly. Then, once $u(t)$ settles around $\frac{n}{2} - \frac{n}{4k}$, some opinions start to steadily increase (even minority ones) while others decrease. One can understand this the following way: in the initial steps, where the number of undecided nodes increase quickly, some opinions, due to randomness, end up having an advantage on the others. Towards the end, the number of undecided nodes starts dropping, it thus goes below all thresholds but the one of the majority opinion, which means all opinions drop quickly apart from the majority one.

In our case, for the study of the lower bound, we fully control the initial configuration of the opinions, and we are able to give a better upper bound on $u(t)$ over time, which happens to be very close to the threshold of any opinion i. To do so, we use drift analysis (for a nice introduction, we refer to Lengler [100]). More specifically, we use a result by Oliveto and Witt [115]. Here is a quick intuition on drift analysis: assuming we know the current

configuration of the system, we compute the expectation of the change in the number of agents in each state after the next interaction. The idea is, if a number changes in expectation by a value α at each interaction, and our goal is to show that the actual number does not wander off by more than β from its original value, this takes at least $\Omega(\beta/\alpha)$ many interactions. A few more hypotheses are needed to ensure that this result holds with high probability, and to ensure that the probabilities are well-behaved, but this is the main idea.

It turns out that $u(t)$ settles around $\frac{n}{2} - \frac{n}{4k}$, and to show that $u(t)$ never substantially exceeds this value, we prove that if it slightly exceeds this value, then at each interaction, in expectation, $u(t)$ decreases by at least $\sqrt{\log n/n}$. Drift analysis tools then allow us to ensure that $u(t)$ will drift no more than (roughly) $\theta(\sqrt{n \log n})$ away from that position.

The second step we make after giving an upper bound on $u(t)$, is to show that in $\theta(kn)$ interactions, no opinion can go from $\frac{3n}{2k}$ nodes to $\frac{2n}{k}$ agents. Here, classic drift analysis fails to help, and that for one main reason: drift analysis essentially looks at how much the expectation increases or decreases over time, but sometimes, the expectation gets beaten by the variance of the process, and drift analysis fails to capture the lack of concentration of the process. To understand that, consider a random walk starting at 0 and which at each step either increases or decreases by 1, each with probability $1/2$. The expectation of its increase is 0 so in expectation, after m steps, it is still at 0. However, we know that with high probability it will have reached $\theta(\sqrt{m})$ at some point during those m steps, as one can see the position after m steps as the result of a binomial distribution with parameters $(1/2, m)$, whose standard deviation is of the order of $\sqrt{m}$.

In our case, we avoid this problem by looking more carefully at the probabilities involved. By considering the evolution of x_i, while its expectation increases slowly over time, it is not overtaken by the variance. To see that, think again of our example of our ± 1 random walk: Imagine that now, the random walk increases with probability $p/2$, decreases with probability $p/2$, and stays put with probability $1 - p$. Look now at what happens after m steps. The walk actually moved for pm out of those steps, and thus the standard deviation is now around $\sqrt{pm}$. If $p = o(1)$, this has a significant impact.

This is exactly what happens in our case, where for x_i to change, in an interaction, a node with opinion i must have been chosen. If at most $2\frac{n}{k}$ nodes with this opinion exist, then the probability of this event happening is of the order of $1/k = o(1)$.

While we could have chosen to stop here for our analysis, and thus give a $\Omega(k)$ lower bound, we go one step further: there is a weakness in this analysis, i.e., we overestimate the number of nodes in opinion i, by stating that the initial count is at most $\frac{3n}{2k}$, while in practice, most opinions stay close to $\frac{n}{2k}$ for most of the process. This (high) estimate of $2\frac{n}{k}$ then gives a too low threshold on u for x_i to increase, and thus the upper bound on the rate at which x_i increases is not tight enough. To improve the result, we would need to give a better estimate on the initial x_i, and show that it is close to $\frac{n}{2k}$. This is not straightforward, as the initial count is close to $\frac{n}{k}$, and we would thus need to analyze the initial phase where u increases sharply while all the x_i decrease.

We find a workaround for this difficulty: while it might take effort to accurately approximate x_i after the first quick-changing phase, a value that is easier to estimate is $\Delta_{ij} = x_i - x_j$. Here, instead of requiring a good estimate on x_i and x_j to analyze the evolution of Δ_{ij}, it turns out

that we only need their order of magnitude, as well as a good estimate of Δ_{ij}, which we have. Therefore, knowing that $x_i \leq 2\frac{n}{k}$ for $\theta(kn)$ iterations allows us to show that the maximum Δ_{ij} needs at least $\theta(kn)$ iterations to double. However, if all Δ_{ij} are small enough (if they are all $o(\frac{n}{k})$), then it means that no opinion drifts far from the others, and thus $x_i \leq \frac{3n}{2k}$ after those $\theta(kn)$ interactions.

Hence, our high level argument works as follows: First give a good estimate on u, then on the number of nodes in each opinion, and on the maximum difference between opinions: As long as $x_i \leq \frac{3n}{2k}$ at a time t and $\Delta_{ij} = o(\frac{n}{k})$ for all i, j, then the order of x_i does not change in the next $\theta(kn)$ interactions, which in turn is used to show that Δ_{ij} does not double during those same interactions. This in turn shows that $x_i \leq \frac{3n}{2k}$, which allows us to restart another iteration of the induction. The induction holds for $\log\left(\frac{\sqrt{n}}{k\log n}\right)$ iterations, which then gives the lower bound.

5.4 Lower bound

In this section, we will give a lower bound on the number of rounds needed for the protocol to stabilize. For that, we analyze the situation where all minority opinions start with the same number of nodes, that is, $x_i(0) = x_j(0)$ for any $i, j \in [2, k]$, while the majority opinion starts with an initial gap of $x_1(0) - x_2(0) = O\left((\sqrt{n}/(k\log n))^{1/4}\sqrt{n\log n}\right)$. We also require that $k = o\left(\frac{\sqrt{n}}{\log n}\right)$. Since a $\Omega(\log n)$ lower bound for the problem is trivial (in $o(\log n)$ parallel time, w.h.p. there are nodes that have not interacted at all) which implies a $\Omega(k\log n)$ lower bound for $k = O(1)$, we focus on the case $k = \omega(1)$.

We first begin by giving an upper bound on the number of undecided nodes.

Lemma 5.4.1. *For any $\tau \leq n^4$, and any initial configuration, it holds with probability at least $1 - n^{-4}$ that $u(\tau) \leq \frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2} + (20\cdot 132 + 1)\sqrt{n\log n}$.*

Proof. To prove the lemma, we will use Theorem 5.7.1. We model $u(t)$ to be a random walk over integers, and first compute the expectation of $u(t+1)$ conditioned on the configuration of the system at time t, which is denoted by $\mathbf{x} = (x_1, \ldots, x_k, u)$ (for ease of notation, we write x_i instead of $x_i(t)$ and u instead of $u(t)$). Clearly, the number of undecided nodes can either decrease by 1, if a decided node interacts with an undecided node, or it can increase by two, if two nodes of different opinions interact. The probability to decrease by one is $2\frac{u}{n}\cdot\frac{n-u}{n-1}$, while the probability to increase by two is $\frac{\sum_{i\in[k]} x_i}{n}\cdot\frac{\sum_{j\neq i} x_j}{n-1}$. Note that $\frac{1}{n-1} = \frac{1}{n} + O(\frac{1}{n^2})$. We therefore have:

$$\begin{aligned}
&\mathbb{E}[u(t+1)\big|\mathbf{x}(t) = \mathbf{x}] \\
&\quad = u - 2\frac{u}{n}\cdot\frac{n-u}{n} + 2\frac{\sum_{i\in[k]} x_i}{n}\frac{\sum_{j\neq i} x_j}{n} + O\left(\frac{1}{n}\right) \\
&\quad = u - 2\frac{u}{n} + 2\frac{u^2}{n^2} + 2\frac{\sum_{i\in[k]} x_i(n-u-x_i)}{n^2} + O\left(\frac{1}{n}\right)
\end{aligned}$$

For the second equality we used $\sum_{j\neq i} x_j = n-u-x_i$. Splitting $x_i(n-u-x_i)$ into $x_i(n-u)$ and $-x_i^2$, and writing $\sum_{i\in[k]} x_i = n-u$ we obtain

$$\begin{aligned}
&= u - 2\frac{u}{n} + 2\frac{u^2}{n^2} + 2\frac{n^2-2un+u^2}{n^2} - 2\frac{\sum_{i\in[k]} x_i^2}{n^2} + O\left(\frac{1}{n}\right)\\
&= u - 6\frac{u}{n} + 4\frac{u^2}{n^2} + 2 - 2\frac{\sum_{i\in[k]} x_i^2}{n^2} + O\left(\frac{1}{n}\right)
\end{aligned}$$

Clearly, if we fix $n-u$, the sum $\sum_{i\in[k]} x_i^2$ is minimized if all x_i are equal. Then we have

$$\begin{aligned}
&\mathbb{E}[u(t+1)\Big|\mathbf{x}(t)=\mathbf{x}]\\
&\quad\leq u - 6\frac{u}{n} + 4\frac{u^2}{n^2} + 2 - 2\frac{(n-u)^2}{kn^2} + O\left(\frac{1}{n}\right)\\
&\quad= u - 6\frac{u}{n} + 4\frac{u^2}{n^2} + 2 - \frac{2}{k} + \frac{4u}{nk} - \frac{2u^2}{n^2k} + O\left(\frac{1}{n}\right)
\end{aligned}$$

Let us assume that $u = \frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2} + c\sqrt{n\log n}$ for some constant c. Then,

$$\begin{aligned}
&\mathbb{E}[u(t+1)\Big|\mathbf{x}(t)=\mathbf{x}]\\
&\qquad\leq u - 6\frac{\frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2} + c\sqrt{n\log n}}{n}\\
&\qquad\quad + 4\frac{(\frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2} + c\sqrt{n\log n})^2}{n^2} + 2 - \frac{2}{k}\\
&\qquad\quad + \frac{4(\frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2} + c\sqrt{n\log n})}{nk}\\
&\qquad\quad - \frac{2(\frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2} + c\sqrt{n\log n})^2}{n^2k} + O\left(\frac{1}{n}\right)\\
&\qquad\leq u - 3 + \frac{3}{2k} - \frac{60}{(k-1)^2} - 6c\sqrt{\frac{\log n}{n}}\\
&\qquad\quad + 1 + \frac{1}{4k^2} - \frac{1}{k} + \frac{40}{(k-1)^2}\\
&\qquad\quad + 4c\sqrt{\frac{\log n}{n}} + 2 - \frac{2}{k} + \frac{2}{k} - \frac{1}{k^2} - \frac{1}{2k} + \frac{1}{2k^2}\\
&\qquad\quad + O\left(\frac{1}{k^3}\right) + o\left(\sqrt{\frac{\log n}{n}}\right)\\
&\qquad\leq u - c\sqrt{\frac{\log n}{n}}
\end{aligned}$$

for any $c \geq 1$ if k is large enough. Define $\tilde{u} = \frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2}$.

We can now apply Theorem 5.7.1 with $X_t = -u(t), X_0 = 0, a = -\tilde{u} - \sqrt{n\log n} - 20\cdot 132\sqrt{n\log n}, b = -\tilde{u} - \sqrt{n\log n}, \ell = 20\cdot 132\sqrt{n\log n}, \epsilon = \sqrt{\frac{\log n}{n}}$ and $r = \sqrt{5}$. Then, we obtain that T^*, the first time so that $u(T^*) \geq \frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2} + \sqrt{n\log n} + 20\cdot 132\sqrt{n\log n}$, satisfies:

$$\mathbb{P}[T^* \leq \exp(4\log n)] \leq O(\exp(-4\log n))$$

This implies that $\mathbb{P}[T^* \leq n^4] \leq O(n^{-4})$. Therefore, with high probability $u(t)$ is less than $\frac{n}{2} - \frac{n}{4k} + \frac{10}{(k-1)^2} + \sqrt{n\log n} + 20 \cdot 132\sqrt{n\log n}$ for all $0 \leq t \leq n^4$. □

We will need the following lemma to continue our analysis. It is a simplified version of Theorem 20 in [100], and provides the tool we need for Lemmas 5.4.3 and 5.4.4.

Lemma 5.4.2. *Let $Y(t)$, $t \geq 0$, be a random walk defined over the set of integers as follows: $Y(0) = 0$, $Y(t+1) = Y(t)$ with probability $1 - p(t)$, $Y(t+1) = Y(t) + 1$ with probability $\frac{p(t)+q(t)}{2}$, and $Y(t+1) = Y(t) - 1$ with probability $\frac{p(t)-q(t)}{2}$. Assume there are values $p > 0$ and $q > 0$ such that $0 \leq p(t) \leq p$ and $-p(t) \leq q(t) \leq q$. For any T such that $T \geq 32\left(\frac{p-q^2}{2q} + \frac{2}{3}\right)\log n$, with probability at least $1-n^{-2}$, $Y(t) < T$ for every $t < \min\{\frac{T}{2q}, n^2\}$ steps.*

Proof. To prove this lemma, we will need Bernstein's inequality (cf. Theorem 5.7.2). We first introduce the random variables $\tilde{Y}(t)$, coupled to $Y(t)$ as follows: $\tilde{Y}(t)$ is a random walk such that with probability $1 - p(t)$, $\tilde{Y}(t+1) = \tilde{Y}(t)$, with probability $\frac{p(t)+q}{2}$, $\tilde{Y}(t+1) = \tilde{Y}(t) + 1$, and with probability $\frac{p(t)-q}{2}$, $\tilde{Y}(t+1) = \tilde{Y}(t) - 1$. Also, with probability 1, $Y(t+1) = Y(t)$ iif $\tilde{Y}(t+1) = \tilde{Y}(t)$ and if $Y(t+1) \geq Y(t)$, then $\tilde{Y}(t+1) \geq \tilde{Y}(t)$.

This can easily be done by sampling a random number $r(t) \in (0,1)$ uniformly at random for every t. If $r(t) \leq 1 - p(t)$, we set $Y(t+1) = Y(t)$ and $\tilde{Y}(t+1) = \tilde{Y}(t)$. If $1 - p(t) \leq r(t) \leq 1 - p(t) + \frac{p(t)+q(t)}{2}$, we set $Y(t+1) = Y(t) + 1$ and $\tilde{Y}(t+1) = \tilde{Y}(t) + 1$. If $1-p(t)+\frac{p(t)+q(t)}{2} \leq r(t) \leq 1-p(t)+\frac{p(t)+q}{2}$, we set $Y(t+1) = Y(t)-1$ and $\tilde{Y}(t+1) = \tilde{Y}(t)+1$. Else, we set $Y(t+1) = Y(t) - 1$ and $\tilde{Y}(t+1) = \tilde{Y}(t) - 1$.

Essentially, if $Y(t)$ increases, so does $\tilde{Y}(t)$. If $Y(t)$ stays put, so does $\tilde{Y}(t)$. If $Y(t)$ decreases, $\tilde{Y}(t)$ might either decrease or increase. This coupling ensures that $\tilde{Y}(t) \geq Y(t)$ for all t. To prove the theorem, it thus suffices to prove the theorem for $\tilde{Y}(t)$.

We will now use Bernstein's inequality with $X_i = \tilde{Y}(i+1) - \tilde{Y}(i) - q$. We use $M = 2$ and $\mathbb{E}[X_i^2] = (1-p(t))q^2 + \left(\frac{p(t)+q}{2}\right)(1-q)^2 + \left(\frac{p(t)-q}{2}\right)(-1-q)^2 = p(t) - q^2 \leq p - q^2$. Hence for any N, $\sum_{i\in[N]} \mathbb{E}[X_i^2] \leq N(p-q^2)$. With $N \leq \frac{T}{2q}$:

$$\begin{aligned}
\mathbb{P}\left(\tilde{Y}(N) \geq T\right) &= \mathbb{P}\left(\sum_{t=1}^{N} X_i \geq T - qN\right) \leq \mathbb{P}\left(\sum_{t=1}^{N} X_i \geq \frac{T}{2}\right) \\
&\leq \exp\left(-\frac{\frac{1}{8}T^2}{\sum_{i\in[N]}\mathbb{E}[X_i^2] + \frac{2T}{3}}\right) \\
&\leq \exp\left(-\frac{\frac{1}{8}T^2}{N(p-q^2) + \frac{2T}{3}}\right) \leq \exp\left(-\frac{\frac{1}{8}T}{\frac{p-q^2}{2q} + \frac{2}{3}}\right) \\
&\leq n^{-4}
\end{aligned}$$

Using a union bound over the first $\min\{\frac{T}{2q}, n^2\}$ steps, we get that the probability that T is reached within the first $\min\{\frac{T}{2q}, n^2\}$ steps is at most n^{-2}. □

We now prove that w.h.p. any given opinion can not increase too much in $\theta(kn)$ interactions, if it starts with fewer than $\frac{3n}{2k}$ nodes.

Lemma 5.4.3. *Let $i \in \{1, \dots, k\}$ be an arbitrary but fixed opinion. Let $\tau_{3n/2k} \leq n^2$ be a time where $x_i(\tau_{3n/2k}) \leq \frac{3n}{2k}$ and $\tau_{2n/k}$ the random variable denoting the time at which $x_i(\tau_{2n/k}) = \frac{2n}{k}$. Then, with probability at least $1 - O(n^{-2})$ we have $\tau = \tau_{2n/k} - \tau_{3n/2k} \geq \frac{kn}{25}$.*

Proof. Let us consider the evolution of $x_i(t)$ for some $t \in \{t_0, \dots, t_0 + kn/25\}$, where $t_0 = \tau_{3n/2k}$. For the analysis, we create a new random process $\mathbf{y}$ similar to $\mathbf{x}$ as follows: $\mathbf{y}(0) = \mathbf{x}(0)$, and for every t, $\mathbf{y}(t+1) = \mathbf{x}(t+1)$ if $\mathbf{y}(t) = \mathbf{x}(t)$ and $u(t+1) \leq \tilde{u} + (20 \cdot 132 + 1)\sqrt{n \log n}$. If $u(t+1)$ does not satisfy the condition, we halt $\mathbf{y}$ (we do not define $\mathbf{y}(t+1)$) and say that $\mathbf{y}$ fails at time $t+1$. By Lemma 5.4.1, $\mathbf{y}$ fails with probability $O(n^{-4})$ in the first n^4 rounds. The modification described above enforces that in $\mathbf{y}$ the number of undecided nodes never surpasses $\frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2} + (20 \cdot 132 + 1)\sqrt{n \log n}$. Clearly, if during the execution of $\mathbf{x}$ the number of undecided nodes does not reach $\frac{n}{2} - \frac{n}{4k} + \frac{10n}{(k-1)^2} + \sqrt{n \log n} + 20 \cdot 132\sqrt{n \log n}$, which happens with probability at least $1 - n^{-4}$ in the first n^4 interactions, then $\mathbf{x}$ and $\mathbf{y}$ behave identically.

Let us now consider the evolution of $x_i(t)$ in $\mathbf{y}$. $x_i(t)$ increases by one if a node of opinion i meets with an undecided node, which happens with probability

$$\mathbb{P}(+1) := \mathbb{P}(x_i(t+1) - x_i(t) = 1 \Big| \mathbf{y}(t) = \mathbf{y}) = 2\frac{x_i}{n}\frac{u}{n-1}$$

Similarly, the probability that it decreases by one is

$$\mathbb{P}(-1) := \mathbb{P}(x_i(t+1) - x_i(t) = -1 \Big| \mathbf{y}(t) = \mathbf{y}) = 2\frac{x_i}{n}\frac{n-u-x_i}{n-1}$$

The goal is to use Lemma 5.4.2. We thus need to compute $p(t) = \mathbb{P}(+1) + \mathbb{P}(-1)$ and $q(t) = \mathbb{P}(+1) - \mathbb{P}(-1)$.

As long as $x_i \leq 2\frac{n}{k}$, we have that $\mathbb{P}(+1) + \mathbb{P}(-1) = 2\frac{x_i}{n}\frac{n-x_i}{n-1} = 2\frac{x_i}{n}(1 + o(1)) \leq \frac{5}{k}$. The difference between the two is

$$\begin{aligned}
&\mathbb{P}(+1) - \mathbb{P}(-1) \\
&\quad = 2\frac{x_i}{n-1}\left(\frac{u}{n} - \frac{n-u-x_i}{n}\right) = 2\frac{x_i}{n(n-1)}(2u - n + x_i) \\
&\quad \leq 2\frac{x_i}{n^2}\left(n - \frac{n}{2k} + O\left(\frac{n}{k \log n}\right) - n + \frac{2n}{k}\right) \\
&\quad = \frac{x_i}{n^2}(3 + O(\frac{1}{\log n}))\frac{n}{k} = 3\frac{x_i}{n}\frac{1 + o(1)}{k} \leq \frac{6}{k^2}(1 + o(1))
\end{aligned}$$

We apply Lemma 5.4.2, with $p = \frac{5}{k}, q = \frac{6.25}{k^2}$, $T = \frac{n}{2k}$. It now suffices to check the conditions of Lemma 5.4.2.

$$\frac{p - q^2}{2q} \leq \frac{\frac{5}{k}}{2\frac{6.25}{k^2}} = O(k) = o(k \log n)$$

and

$$T = \frac{n}{2k} \geq \frac{k^2 \log^2 n}{2k} = \omega(k \log^2 n)$$

Therefore, we have that $T \geq 8(\frac{p-q^2}{2q} + \frac{2}{3}) \log n$. Thus, $x_i(t_0 + \tau) < \frac{2n}{k}$ for any $\tau \leq \frac{kn}{25}$ with probability at least $1 - n^{-2}$ in $\mathbf{y}$. As $\mathbf{x}$ and $\mathbf{y}$ behave identically with probability $1 - n^{-4}$ in the first n^4 interactions, the lemma follows. □

Now that we have a good upper bound on u and a good estimate on the order of magnitude of all of the x_i, we can show that it takes $\theta(kn)$ interactions for the difference between two opinions to double.

Lemma 5.4.4. *Assume that at some time step $t_0 \leq nk \log^2 n$ the difference between any two opinions is at most $\frac{\alpha}{2} = \omega(\sqrt{n \log n})$, where $\alpha = o\left(\frac{n}{k}\right)$. Furthermore, assume that $x_i(t_0)) \leq \frac{3n}{2k}$ for every $i \in [k]$. Then, with probability at least $1 - O(k^2/n^2)$, the difference between any two opinions does not exceed α before interaction $t_0 + \tau$, where $\tau = \frac{1}{24}kn$.*

Proof. Consider two arbitrary but fixed opinions i and j. Similarly to the proof of Lemma 5.4.3 we analyze the evolution of $\Delta_{ij}(t) = x_i(t) - x_j(t)$ for a time step $t \in \{t_0, \ldots, t_0 + \tau\}$. For the analysis, we create a new random process $\mathbf{y}$ similar to $\mathbf{x}$ as follows: $\mathbf{y}(0) = \mathbf{x}(0)$, and for every t, $\mathbf{y}(t+1) = \mathbf{x}(t+1)$ if $\mathbf{y}(t) = \mathbf{x}(t)$ and $u(t+1) \leq \tilde{u} + (20 \cdot 132 + 1)\sqrt{n \log n}$ and $x_i(t+1) \leq \frac{2n}{k}$ for every i. If either $u(t+1)$ or one of the x_i does not satisfy the condition, we halt $\mathbf{y}$ (we do not define $\mathbf{y}(t+1)$) and say that $\mathbf{y}$ fails at time $t+1$.

Clearly, $\mathbf{x}$ and $\mathbf{y}$ behave identically with probability $1 - O(kn^{-2})$ according to Lemmas 5.4.1 and 5.4.3 (by union bound over the k different opinions).

We consider now the evolution of $\Delta_{ij}(t)$ in $\mathbf{y}$. For ease of notation, we set $x_i = x_i(t)$, $x_i = x_i(t)$, and $u = u(t)$. $\Delta_{ij}(t)$ increases by one if x_i increases by one, which happens when a node with opinion i interacts with an undecided node, hence with probability $2\frac{x_i}{n}\frac{u}{n-1}$. It also increases by 1 if x_j decreases by one, but not x_i, which happens when a node with opinion j interacts with a node of opinion in $[k] \setminus \{i, j\}$, hence with probability $2\frac{x_i}{n}\frac{n-u-x_i-x_j}{n-1}$. Therefore, the probability that $\Delta_{ij}(t+1) > \Delta_{ij}(t)$ is:

$$\begin{aligned}\mathbb{P}(+1) := \mathbb{P}(\Delta_{ij}(t+1) - \Delta_{ij}(t) = 1 \Big| \mathbf{y}(t) = \mathbf{y}) \\ = 2\frac{x_i}{n}\frac{u}{n-1} + 2\frac{x_j}{n}\frac{n-u-x_i-x_j}{n-1}\end{aligned}$$

Similarly, the probability that it decreases by one is

$$\begin{aligned}\mathbb{P}(-1) := \mathbb{P}(\Delta_{ij}(t+1) - \Delta_{ij}(t) = -1 \Big| \mathbf{y}(t) = \mathbf{y}) \\ = 2\frac{x_j}{n}\frac{u}{n-1} + 2\frac{x_i}{n}\frac{n-u-x_i-x_j}{n-1}\end{aligned}$$

The goal is to use Lemma 5.4.2. We thus need to compute $p(t) = \mathbb{P}(+1) + \mathbb{P}(-1)$ and $q(t) = \mathbb{P}(+1) - \mathbb{P}(-1)$. Expanding the terms, we get that the sum of $\mathbb{P}(+1)$ and $\mathbb{P}(-1)$ is:

$$\begin{aligned}\mathbb{P}(+1)+\mathbb{P}(-1) &= 2\frac{(x_i+x_j)u}{n(n-1)}+2\frac{(x_i+x_j)}{n(n-1)}(n-u-x_i-x_j)\\ &\leq 2\frac{4\frac{n}{k}}{n(n-1)}(n-x_i-x_j)=8\frac{1}{k}(1+o(1))\end{aligned}$$

The difference between the two is:

$$\begin{aligned}&\mathbb{P}(+1)-\mathbb{P}(-1)\\ &\quad=\frac{2}{n(n-1)}((ux_i+nx_j-ux_j-x_ix_j-x_j^2)\\ &\qquad+(-ux_j-nx_i+ux_i+x_i^2+x_ix_j))\\ &\quad=\frac{2}{n(n-1)}(2u(x_i-x_j)-n(x_i-x_j)\\ &\qquad+(x_i-x_j)+(x_i-x_j)(x_i+x_j))\\ &\quad=\frac{2}{n(n-1)}(x_i-x_j)(2u-n+x_i+x_j)\\ &\quad\leq\frac{2}{n(n-1)}(x_i-x_j)(n-\frac{n}{2k}+o(\frac{n}{k})-n+2\frac{n}{k})\\ &\quad=\frac{3}{n-1}\Delta_{ij}(\frac{1}{k}+o(\frac{1}{k}))\end{aligned}$$

If the difference at a time t is $x_i(t)-x_j(t)\leq\alpha$, then $\mathbb{P}(+1)-\mathbb{P}(-1)\leq\frac{3\alpha}{nk}(1+o(1))$.

We apply Lemma 5.4.2, with $p=\frac{9}{k}$, $q=\frac{6\alpha}{nk}$, and $T=\frac{\alpha}{2}$, $Y(t)=\Delta_{ij}(t)-\frac{\alpha}{2}$. Indeed, we can see the problem as a random walk that starts at $Y(0)=0$ (which corresponds to $\Delta_{ij}=\frac{\alpha}{2}$) and whose target is $Y(\tau)=\frac{\alpha}{2}$ (which corresponds to $\Delta_{ij}=\alpha$). As long as the target is not reached, we have that $\Delta_{ij}\leq\alpha$, and thus, $q(t)\leq\frac{6\alpha}{nk}$. It now suffices to check the conditions of Lemma 5.4.2.

We have $T=\frac{\alpha}{2}=\omega(\sqrt{n\log n})$, $\frac{2}{3}\log n=o(\sqrt{n\log n})$ and $\frac{p-q^2}{q}\log n\leq\frac{p}{q}\log n=O(\frac{n}{\alpha}\log n)=o(\frac{n}{\sqrt{n\log n}}\log n)=o(\sqrt{n\log n})$, and thus $32\log n\left(\frac{p-q^2}{2q}+\frac{2}{3}\right)=o(\sqrt{n\log n})\leq T=\omega(\sqrt{n\log n})$. We thus know that with probability at least $1-n^{-2}$ the difference will not exceed α before $\frac{T}{2q}=\frac{1}{24}kn$.

Taking a union bound over all $k(k-1)/2$ pairs of opinions, we know that the probability that one difference exceeds α at time $t_0+\tau$ is less than k^2/n^{-2}. Since $\mathbf{x}$ and $\mathbf{y}$ as described at the beginning of this proof behave identically with probability $1-O(kn^{-2})$, we obtain the lemma. The lemma also implies that with probability $1-O(k^2/n^2)$ the support of each opinion is less than $3n/2k$ at time $t_0+\tau$. □

We are now ready to prove the main theorem.

Theorem 5.4.5. *For any $k=\omega(1), k=o(\frac{\sqrt{n}}{\log n})$, any initial configurations where the maximum difference between two opinions is $\max_{i,j\in[k]}\{x_i(0)-x_j(0)\}=$*

$O\left((\sqrt{n}/(k\log n))^{1/4}\sqrt{n\log n}\right)$ *under the Undecided State Dynamics for Plurality Consensus does not stabilize in* $\frac{k}{25}\log\frac{\sqrt{n}}{k\log n}$ *parallel time, with high probability.*

Proof. Define f: $f(n)=\left(\frac{\sqrt{n}}{k\log n}\right)^{\frac{1}{4}}$.

We bunch together the interactions by groups of $\tau=\frac{kn}{25}$ interactions, and use induction on ℓ: For $\ell\leq\log\left(\frac{\frac{n^{\frac{3}{4}}}{k^{\frac{1}{2}}}}{\sqrt{n\log n}f(n)}\right)$, after the ℓ-th group of interactions, we have that $x_i(\ell\tau)\leq\frac{3n}{2k}$ for every i, and that $\Delta(\ell\tau):=\max_{ij}\Delta_{ij}(\ell\tau)\leq 2^\ell\beta$ with probability $1-O(\ell k^2/n^2)$, where the initial bias starts at $\Delta(0)=\beta=O\left(f(n)\sqrt{n\log n}\right)$.

The base step is trivial. To show the induction step, we first note that for $\ell\leq\log\left(\frac{\frac{n^{\frac{3}{4}}}{k^{\frac{1}{2}}}}{\sqrt{n\log n}f(n)}\right)$, we have $2^\ell\beta\leq\frac{n^{\frac{3}{4}}}{k^{\frac{1}{2}}}$, which in turn means that, using $k\leq\frac{n^{\frac{1}{2}}}{\log n}$, $2^\ell\beta\leq\frac{n^{\frac{3}{4}}}{k^{\frac{1}{2}}}\cdot\left(\frac{n^{\frac{1}{2}}}{k\log n}\right)^{\frac{1}{2}}=o(\frac{n}{k})$. We then apply Lemma 5.4.3 and Lemma 5.4.4: First, since for any i, $x_i(\ell\tau)\leq\frac{3n}{2k}$, and by our result on u (Lemma 5.4.1), we apply Lemma 5.4.3 to have that during the whole interval, with probability at least $1-O(n^{-2})$, x_i does not exceed $2\frac{n}{k}$. By union bound over all values of i, it holds that for all i, no x_i exceeds $2\frac{n}{k}$ during the $\frac{kn}{25}$ interactions with probability at least $1-O(kn^{-2})$. Then, since $\Delta(\ell\tau)\leq 2^\ell\beta$, $x_i(t)\leq 2\frac{n}{k}$ for any $t\leq(\ell+1)\tau$, and by our result on u, we can apply Lemma 5.4.4. This ensures that $\Delta((\ell+1)\tau)\leq 2^{\ell+1}\beta$ with probability at least $1-O((\ell+1)k^2/n^2)$. For $\ell+1\leq\log\left(\frac{\frac{n^{\frac{3}{4}}}{k^{\frac{1}{2}}}}{\sqrt{n\log n}f(n)}\right)$, this ensures $\Delta((\ell+1)\tau)\leq\frac{n^{\frac{3}{4}}}{k^{\frac{1}{2}}}$, which in turn means that, using $k\leq\frac{n^{\frac{1}{2}}}{\log n}$, $\Delta((\ell+1)\tau)\leq\frac{n^{\frac{3}{4}}}{k^{\frac{1}{2}}}\cdot\left(\frac{n^{\frac{1}{2}}}{k\log n}\right)^{\frac{1}{2}}=o(\frac{n}{k})$. Thus $x_i((\ell+1)\tau)\leq\frac{3n}{2k}$ for every $i\in[k]$, as otherwise $\sum_{j\in[k]}x_j((\ell+1)\tau)\geq\frac{3}{2}\frac{n}{k}+\sum_{j\neq i}x_j((\ell+1)\tau)\geq\frac{3}{2}\frac{n}{k}+(k-1)\frac{n}{k}>n$. This proves the double induction.

Hence, with high probability, the system does not stabilize before $\frac{kn}{25}\log\left(\frac{\frac{n^{\frac{3}{4}}}{k^{\frac{1}{2}}}}{\sqrt{n\log n}f(n)}\right)=\frac{kn}{25}\left(\frac{1}{2}\log\left(\frac{\sqrt{n}}{k\log n}\right)-\log f(n)\right)=\theta\left(kn\log\frac{\sqrt{n}}{k\log n}\right)$ interactions, concluding the proof. □

5.5 Conclusion

We presented an almost tight lower bound on the stabilization time of the Undecided State Dynamics for plurality consensus in the population protocol model. While our result settles the question about the stabilization time, there are several interesting avenues for future research. In particular, it would be interesting to explore scenarios where (slightly) more memory is available at the nodes and where synchronization is possible to some extent: at which point can we break the lower bound barrier? Another open question concerns the required initial bias. While it is known from previous work that with an initial bias in the order of $O(\sqrt{n})$, the system can stabilize to a minority opinion with non-negligible probability [40], we assumed a

slightly higher initial bias of $\Omega(\sqrt{n \log n})$ which ensures stabilization to the majority opinion (cf. [9]); it remains to close this small gap.

5.6 Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (MoDynStruct, No. 101019564) and the Austrian Science Fund (FWF) grant DOI 10.55776/I5862, grant DOI 10.55776/I5982, and grant DOI 10.55776/P33775 with additional funding from the netidee SCIENCE Stiftung, 20202024 and the German Research Foundation (DFG), grant 470029389 (FlexNets).

5.7 Useful theorems

Theorem 5.7.1 (Theorem 2 of [115]). *Let X_t, $t \geq 0$,be real-valued random variables describing a stochastic process over some state space. Suppose there exist an interval $[a, b] \subseteq \mathcal{R}$ and, possibly depending on $\ell := b - a$, a drift bound $\epsilon := \epsilon(\ell) > 0$ as well as a scaling factor $r := r(\ell)$ such that for all $t \geq 0$ the following three conditions hold:*

- $\mathbb{E}[X_{t+1} - X_t | X_0, \ldots, X_t; a < X_t < b] \geq \epsilon$.
- $\mathbb{P}[|X_{t+1} - X_t| \geq jr | X_0, \ldots, X_t] \leq e^{-j}$ *for* $j \in \mathbb{N}_0$.
- $1 \leq r^2 \leq \frac{\epsilon \ell}{132 \log \frac{r}{\epsilon}}$.

Then for the first hitting time $T^\star := \min\{t \geq 0 : X_t \leq a | X_0, \ldots, X_t; X_0 \geq b\}$ it holds that $\mathbb{P}\left[T^* \leq \exp\left(\frac{\epsilon \ell}{132 r^2}\right)\right] = O\left(\exp\left(-\frac{\epsilon \ell}{132 r^2}\right)\right)$

Theorem 5.7.2 (Bernstein's Inequality). *Let $X_1, \ldots, X_n$ be independent zero-mean random variables. Suppose that $|X_i| \leq M$ almost surely, for all i. Then, for all positive t,*

$$\mathbb{P}\left(\sum_{i=1}^{n} X_i \geq t\right) \leq \exp\left(-\frac{\frac{1}{2}t^2}{\sum_{i=1}^{n} \mathbb{E}\left[X_i^2\right] + \frac{1}{3}Mt}\right)$$

References

[1] Yehuda Afek and Eli Gafni. "Asynchrony from synchrony". In: *Distributed Computing and Networking: 14th International Conference, ICDCN 2013, Mumbai, India, January 3-6, 2013. Proceedings 14*. Springer. 2013, pp. 225–239.

[4] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. "Time-Space Trade-offs in Population Protocols". In: *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'17*. 2017, pp. 2560–2579.

[5] Dan Alistarh, James Aspnes, and Rati Gelashvili. "Space-Optimal Majority in Population Protocols". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'18*. 2018, pp. 2221–2239.

[7] Dan Alistarh and Rati Gelashvili. "Recent Algorithmic Advances in Population Protocols". In: *SIGACT News* 49.3 (2018), pp. 63–73. DOI: 10.1145/3289137.3289150. URL: https://doi.org/10.1145/3289137.3289150.

[8] Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. "Fast and Exact Majority in Population Protocols". In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC'15*. Ed. by Chryssis Georgiou and Paul G. Spirakis. ACM, 2015, pp. 47–56. ISBN: 978-1-4503-3617-8. DOI: 10.1145/2767386.2767429. URL: http://doi.acm.org/10.1145/2767386.2767429.

[9] Talley Amir, James Aspnes, Petra Berenbrink, Felix Biermeier, Christopher Hahn, Dominik Kaaser, and John Lazarsfeld. "Fast Convergence of k-Opinion Undecided State Dynamics in the Population Protocol Model". In: *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*. Ed. by Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu. ACM, 2023, pp. 13–23. DOI: 10.1145/3583668.3594589. URL: https://doi.org/10.1145/3583668.3594589.

[10] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. "Computation in networks of passively mobile finite-state sensors". In: *Distributed Computing* 18.4 (2006), pp. 235–253.

[11] Dana Angluin, James Aspnes, and David Eisenstat. "Fast computation by population protocols with a leader". In: *Distributed Computing* 21.3 (Sept. 2008), pp. 183–199.

[15] Gregor Bankhamer, Petra Berenbrink, Felix Biermeier, Robert Elsässer, Hamed Hosseinpour, Dominik Kaaser, and Peter Kling. "Fast Consensus via the Unconstrained Undecided State Dynamics". In: *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*. Ed. by Joseph (Seffi) Naor and Niv Buchbinder. SIAM, 2022, pp. 3417–3429. DOI: 10.1137/1.9781611977073.135. URL: https://doi.org/10.1137/1.9781611977073.135.

[18] Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Riccardo Silvestri. "Plurality Consensus in the Gossip Model". In: *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'15*. 2015, pp. 371–390.

[19] Stav Ben-Nun, Tsvi Kopelowitz, Matan Kraus, and Ely Porat. "An $O(\log^{3/2} n)$ Parallel Time Population Protocol for Majority with $O(\log n)$ States". In: *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*. Ed. by Yuval Emek and Christian Cachin. ACM, 2020, pp. 191–199. DOI: 10.1145/3382734.3405747. URL: https://doi.org/10.1145/3382734.3405747.

[20] Petra Berenbrink, Robert Elässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. "A population protocol for exact majority with $O(\log^{5/3} n)$ stabilization time and $O(\log n)$ states". In: *Proceedings of the 32nd International Symposium on Distributed Computing, DISC'18*. 2018.

[21] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Domink Kaaser Peter Kling, and Tomasz Radzik. "Time-space trade-offs in population protocols for the majority problem". In: *Distributed Computing* 34.2 (2021). Full version available at arXiv:1805.04586, pp. 91–111.

[24] Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. "Brief Announcement: Population Protocols for Leader Election and Exact Majority with $O(\log^2 n)$ States and $O(\log^2 n)$ Convergence Time". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC'17*. Full version available at arXiv:1705.01146. 2017, pp. 451–453.

[30] Luca Cardelli and Attila Csiksz-Nagy. "The cell cycle switch computes approximate majority". In: *Nature Scientific Reports* 2 (2012), p. 656.

[37] Yuen-Jyue Chen, Neil Dalchau, Niranjan Srnivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. "Programmable chemical controlers made from DNA". In: *Nature Nanotechnology* 8.10 (2013), pp. 755–762.

[40] Andrea Clementi, Mohesen Ghaffari, Luciano Gualà, Emanuele Natale, Francesco Pasquale, and Giacomo Scornavacca. "A Tight Analysis of the Parallel Undecided-State Dynamics with Two Colors". In: *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science, MFCS'18*. 2018, 28:1–28:15.

[51] David Doty, Mahsa Eftekhari, Leszek Gasieniec, Eric E. Severson, Przemyslaw Uznanski, and Grzegorz Stachowiak. "A time and space optimal stable population protocol solving exact majority". In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 1044–1055. DOI: 10.1109/FOCS52979.2021.00104. URL: https://doi.org/10.1109/FOCS52979.2021.00104.

[53] Moez Draief and Milan Vojnovic. "Convergence Speed of Binary Interval Consensus". In: *Proceedings of the 29th IEEE International Conference on Computer Communications, INFOCOM'10*. 2010, pp. 1792–1800. DOI: 10.1109/INFCOM.2010.5461999. URL: http://dx.doi.org/10.1109/INFCOM.2010.5461999.

[64] Robert Elsässer and Tomasz Radzik. "Recent Results in Population Protocols for Exact Majority and Leader Election". In: *Bull. EATCS* 126 (2018). URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/549/546.

[72] Leszek Gasieniec, David D. Hamilton, Russell Martin, Paul G. Spirakis, and Grzegorz Stachowiak. "Deterministic Population Protocols for Exact Majority and Plurality". In: *20th International Conference on Principles of Distributed Systems, OPODIS 2016*. Ed. by Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone. Vol. 70. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 14:1–14:14. DOI: 10.4230/LIPIcs.OPODIS.2016.14. URL: https://doi.org/10.4230/LIPIcs.OPODIS.2016.14.

[97] Adrian Kosowski and Przemyslaw Uznanski. "Brief Announcement: Population Protocols Are Fast". In: *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing, PODC'18*. Full version available at arXiv:1802.06872. 2018, pp. 475–477. DOI: 10.1145/3212734.3212788. URL: http://doi.acm.org/10.1145/3212734.3212788.

[99] Fabian Kuhn and Rotem Oshman. "Dynamic networks: models and algorithms". In: *ACM SIGACT News* 42.1 (2011), pp. 82–96.

[100] Johannes Lengler. "Drift Analysis". In: *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*. Ed. by Benjamin Doerr and Frank Neumann. Natural Computing Series. Springer, 2020, pp. 89–131. DOI: 10.1007/978-3-030-29414-4_2. URL: https://doi.org/10.1007/978-3-030-29414-4%5C_2.

[107] George B. Mertzios, Sotiris E. Nikoletseas, Christoforos L. Raptopoulos, and Paul G. Spirakis. "Determining Majority in Networks with Local Interactions and Very Small Local Memory". English. In: *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, ICALP'14*. Springer Berlin Heidelberg, 2014, pp. 871–882. ISBN: 978-3-662-43947-0. DOI: 10.1007/978-3-662-43948-7_72. URL: http://dx.doi.org/10.1007/978-3-662-43948-7_72.

[108] Othon Michail, George Skretas, and Paul G Spirakis. "Distributed computation and reconfiguration in actively dynamic networks". In: *Proceedings of the 39th Symposium on Principles of Distributed Computing*. 2020, pp. 448–457.

[114] Emanuele Natale and Iliad Ramezani. "On the Necessary Memory to Compute the Plurality in Multi-agent Systems". In: *Algorithms and Complexity - 11th International Conference, CIAC 2019*. Ed. by Pinar Heggernes. Vol. 11485. Lecture Notes in Computer Science. Springer, 2019, pp. 323–338. DOI: 10.1007/978-3-030-17402-6_27. URL: https://doi.org/10.1007/978-3-030-17402-6_27.

[115] Pietro S. Oliveto and Carsten Witt. "Improved time complexity analysis of the Simple Genetic Algorithm". In: *Theor. Comput. Sci.* 605 (2015), pp. 21–41. DOI: 10.1016/J.TCS.2015.01.002. URL: https://doi.org/10.1016/j.tcs.2015.01.002.

[121] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Brook. "Computation with finite stochastic chemical reaction networks". In: *Natural Computing* 4.7 (2008), pp. 615–633.

CHAPTER 6

Deterministic and Exact Fully-dynamic Minimum Cut of Superpolylogarithmic Size in Subpolynomial Time

This chapter appeared in parts or fully in

Antoine El-Hayek, Monika Henzinger, and Jason Li. "Fully Dynamic Approximate Minimum Cut in Subpolynomial Time per Operation". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*. Ed. by Yossi Azar and Debmalya Panigrahi. SIAM, 2025, pp. 750–784. DOI: 10.1137/1.9781611978322.22. URL: https://doi.org/10.1137/1.9781611978322.22

and

Antoine El-Hayek, Monika Henzinger, and Jason Li. "Deterministic and Exact Fully-dynamic Minimum Cut of Superpolylogarithmic Size in Subpolynomial Time". In: *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*. Ed. by Kasper Green Larsen and Barna Saha. SIAM, 2026, pp. 613–663. DOI: 10.1137/1.9781611978971.25. URL: https://doi.org/10.1137/1.9781611978971.25

6.1 Abstract

We present an exact fully-dynamic minimum cut algorithm that runs in $n^{o(1)}$ deterministic update time when the minimum cut size is at most $2^{\Theta(\log^{3/4-c} n)}$ for any $c > 0$, improving on the previous algorithm of Jin, Sun, and Thorup (SODA 2024) whose minimum cut size limit is $(\log n)^{o(1)}$. Combined with graph sparsification, we obtain the first $(1+\epsilon)$-approximate fully-dynamic minimum cut algorithm on *weighted* graphs, for any $\epsilon \geq 2^{-\Theta(\log^{3/4-c} n)}$, in $n^{o(1)}$ randomized update time.

Our main technical contribution is a deterministic local minimum cut algorithm, which replaces the randomized LocalKCut procedure from El-Hayek, Henzinger, and Li (SODA 2025).

6.2 Introduction

We consider the minimum cut problem in an undirected, unweighted graph: find a set of edges of minimum size whose removal disconnects the graph. This is a basic problem in combinatorial optimization with a rich body of work, and its study has led to the discovery of many fundamental concepts in graph algorithms, such as randomized contractions [92] and graph sparsification [93]. The randomized running time was famously settled by Karger [91], who also posed as an open problem whether a fast deterministic algorithm exists. That problem proved notoriously difficult and resisted progress for 20 years, until a recent sequence of works [95, 102, 101, 82] finally obtained a deterministic algorithm that is optimal up to polylogarithmic factors.

However, on dynamic graphs undergoing edge insertions and deletions, this problem is much less understood. For a long time, the state of the art algorithm for $(1 + o(1))$-approximate minimum cut was due to Thorup [123], which is Monte-Carlo randomized and runs in $\tilde{O}(\sqrt{n})$ worst-case update time. Recently, El-Hayek, Henzinger, and Li [82] improved the running time to $n^{o(1)}$ amortized update time [59], also by a Monte-Carlo randomized algorithm. In comparison, the fastest deterministic approximation algorithm is the $\sqrt{2 + o(1)}$-approximate algorithm of Thorup and Karger [125], which runs in $O(\lambda^2 \text{polylog}(n))$ time where λ is the minimum cut of the graph, but is only capable of outputting the (approximate) cut-size of the minimum cut. For larger approximations, Goranci, Räcke, Saranurak and Tan [77], then van den Brand, Chen, Kyng, Liu, Meierhans, Gutenberg, and Sachdeva [26] gave an $n^{o(1)}$-approximate algorithm that runs in $n^{o(1)}$ worst-case deterministic update time, in unweighted and weighted graphs respectively. For exact minimum cut, Jin, Sun, and Thorup [89] obtained $n^{o(1)}$ worst-case deterministic update time when the minimum cut-size is $(\log n)^{o(1)}$, while Goranci, Henzinger, Nanongkai, Saranurak, Thorup, and Wulff-Nilsen [75] obtained the first sublinear-time algorithm for all cut-sizes, which takes $\tilde{O}(n)$ worst-case randomized time or $\tilde{O}(m^{30/31})$ amortized deterministic time, which was further improved by de Vos and Christiansen [126] to $O(m^{11/12})$ update time with a deterministic algorithm. Overall, the randomized running times are notably better than the deterministic ones, which suggests that deterministic minimum cut is notoriously difficult even in the dynamic setting.

In this paper, we significantly advance the state of the art for exact algorithms, obtaining an algorithm in $n^{o(1)}$ amortized deterministic update time for minimum cut-sizes up to $2^{\Theta(\log^{3/4-c} n)}$ for any $c > 0$. This is a significant leap over the $(\log n)^{o(1)}$ cut-size limit of Jin, Sun, and Thorup [89], at a cost of amortized instead of worst-case update time. Most notably, due to the increased minimum cut-size that our algorithm can handle, we can use the standard approach of sparsifying the graph to reduce the minimum cut-size to $O(\frac{\log n}{\epsilon^2})$ and then applying our algorithm on the resulting graph. This leads to the first $(1 + o(1))$-approximate dynamic minimum cut algorithm even on *weighted* graphs in $n^{o(1)}$ update time.

To construct our deterministic algorithm, we replace the randomized LocalKCut procedure of [59] with an entirely new algorithm based on tree packing and color coding, based on the techniques of Lokshtanov, Saurabh, and Surianarayanan [103]. This new deterministic LocalKCut procedure is one of our main technical contributions. Furthermore, to increase the minimum cut-size bound to $2^{\Theta(\log^{3/4-c} n)}$ for any $c > 0$, we adapt an idea from the static setting [82] to the dynamic setting.

Our main result and its applications. We next describe our main result as well as its applications. Since our main result is about *proper* cuts, we need to generalize the minimum cut problem to the minimum *proper* cut problem.

Definition 6.2.1 (Minimum proper cut)**.** *Let $G = (V, E)$ be a graph, and $F \subset E$ be a subset of edges. F is said to be a* proper *cut of G if the removal of F strictly increases the number of connected components of G, and is minimal for this property. Equivalently, a set $U \subset V$ of nodes is said to be a proper cut if it is connected and has at least one outgoing edge.*

Thus, a minimum proper cut is the smallest non-zero cut in a potentially disconnected graph. Our data structure will work recursively, where in each level of the recursion only a subset of the cuts is analyzed. To do so, certain edges have been removed from the graph, which might lead to a disconnected subgraph on this level. Instead of starting a new data structure for each connected component, which is what is usually done in this case, we maintain *one* data structure per level, representing the subgraph on this level. This is crucial for achieving our improvements, but in turn we need for each level an algorithm that is able to maintain a minimum *proper* cut.

More specifically, our main technical contribution is a deterministic algorithm that solves the dynamic minimum proper cut problem for specific parameter ranges: [Bounded Dynamic Minimum Proper Cut] Let G be an unweighted fully dynamic graph with n nodes under edge updates, and let $\lambda_{\min} < \lambda_{\max} \in \mathbb{N}$ be two values with $\lambda_{\max} \leq 1.2\lambda_{\min} = 2^{O(\log^{3/4-c} n)}$ for some $c > 0$. Assume that the minimum proper cut of G is at least $\lambda_{\min}$ at all times. The *Bounded Dynamic Minimum Proper Cut* problem is to maintain a data structure that, depending on the value of the minimum proper cut λ, gives the following output:

noitemsep If $\lambda_{\min} \leq \lambda \leq \lambda_{\max}$, outputs the (exact) value of the minimum proper cut. Moreover, a partition achieving the minimum proper cut should be stored implicitly.

noiitemsep If $\lambda > \lambda_{\max}$, outputs "$\lambda > \lambda_{\max}$".

Theorem 6.2.2. *There exists a deterministic algorithm that solves Section 6.2 in $n^{o(1)}$ update time.*

Before we delve into the main techniques we developed to prove this theorem, we first discuss its implications. The first one is an exact algorithm for dynamic minimum cut for $\lambda < 2^{\Theta(\log^{3/4-c} n)}$ for any $c > 0$:

Theorem 6.2.3 (Fully Dynamic Exact Mincut)**.** *There exists a deterministic algorithm that solves the minimum cut problem* exactly *on a fully dynamic unweighted graph in $n^{o(1)}$ update time, as long as the value of the minimum cut is $\lambda < 2^{\Theta(\log^{3/4-c} n)}$ for some $c > 0$. It outputs the value of the minimum cut and stores a partition achieving the minimum cut implicitly.*

To prove Theorem 6.2.3 from Theorem 6.2.2, simply run a connectivity algorithm and additionally $O(\log n)$ many instances of the algorithm from Theorem 6.2.2, one per range where $\lambda_{\min} = 1.2^i$ and $\lambda_{\max} = 1.2^{i+1}$, for all ranges, i.e., all values of $i \in [0, \lceil 2 \log n \rceil]$. Note that each instance is built for the whole graph, whether it is connected or disconnected, there is *not* one instance per connected component and range – in fact, we ensure, as we discuss

below, that the input for every instance is connected at all times, by delaying updates to the different instances to ensure connectivity guarantees. Whenever the connectivity algorithm returns that the graph is disconnected, simply output that the minimum cut is 0. Otherwise, we need to find the minimum proper cut, which is equal to the minimum cut. To ensure that the minimum proper cut is at least $\lambda_{\min}$ for every instance, we give first the initial graph to the instance with smallest index $i = 0$. If that instance returns "$\lambda > \lambda_{\max}$", this ensures that $\lambda \geq 1.2^{i+1}$ which is the $\lambda_{\min}$ of the next instance. We thus give the graph to the next instance, and do that iteratively until one instance returns a value for the minimum proper cut that is in its range. We then store for each instance an integer "Last update time", that is either 0 if it has been instantiated, or -1 otherwise. To handle an update, we proceed similarly, updating the connectivity algorithm first, and if it returns that the graph is connected, updating or instantiating all the instances in increasing order of indices, and stopping when one instance returns "$\lambda > \lambda_{\max}$". During that phase, when considering an instance, we know that before updating it, either the instance has not been instantiated, or the graph it is considering – which might be out of date – satisfies $\lambda \geq \lambda_{\min}$. In the first case, since we know from the instance with lower index that $\lambda \geq \lambda_{\min}$ in the current graph, we can instantiate the instance with the current graph. In the latter case, we know, since the graph is connected (from the connectivity algorithm), that handling all edge insertions then all edge deletions since the last update to the instance, will bring the instance up to date, and ensure that $\lambda \geq \lambda_{\min}$ throughout. We then update the variable "Last update time" accordingly. To answer the query, we report the value of the one algorithm that outputs a value inside its range and is up to date. We update the instances in this way as we have to guarantee that the minimum proper cut in any instance is at least $\lambda_{\min}$ at all time.

A note on the maximum value of λ, $\lambda_{\min}$ and $\lambda_{\max}$: It is worth discussing briefly why we require $\lambda_{\min} = 2^{O(\log^{3/4-c} n)}$ for any $c > 0$, as opposed to having an algorithm valid for all values of λ that runs in time $\mathrm{poly}(\lambda_{\min}) \cdot n^{o(1)}$. As we discuss below, our algorithm is in fact a $(1+\delta)$-approximate algorithm, for $\delta = 2^{-O(\log^{3/4-c} n)}$. Said differently, our algorithm can only be exact for values of $\lambda_{\max}$ that are less than $1/\delta$. We cannot decrease our approximation ratio, as our algorithm is recursive, and the graph size (number of edges in the graph) and recourse for each recursive call depends on δ and $\lambda_{\max}$: more precisely, the recourse per level is $\tilde{O}(\frac{\rho\lambda_{\max}}{\delta^3})$ with $\rho = 2^{O(\sqrt{\log n})}$ for any $c > 0$, and after a recursive depth of $r = \sqrt[4]{\log n}$, the goal is to have a recourse of $2^{O(\log^{1-c} n)} = n^{o(1)}$. Thus, making δ too small or $\lambda_{\max}$ too high would make the overall recourse too high.

Note also that our algorithm relies on the dynamic expander decomposition of Goranci, Räcke, Saranurak and Tan [77]. This algorithm maintains an expander hierarchy with expander parameter $\phi = 2^{-\Theta(\log^{3/4} n)}$ and recourse $\rho = 2^{\Theta(\log^{1/2} n)}$. Our cluster decomposition achieves then the same depth as their expander hierarchy, a depth of $O(\sqrt[4]{\log n})$, which is a direct consequence of the choice of the value of ϕ, as the number of edges is multiplied by ϕ from a level to the next, and $m\phi^{a\sqrt[4]{\log n}} = 1$ for a constant a that is large enough. Importantly, their recourse per level is truly smaller than $\frac{1}{\phi}$, and hence, after $O(\sqrt[4]{\log n})$ many levels, the recourse is $\rho^{O(\sqrt[4]{\log n})} = n^{o(1)}$. One could hope for other dynamic expander decompositions with different tradeoffs for ϕ and ρ. In that case, our algorithm would adapt very effectively: For example, let $\gamma > 0$ be a real value and assume that we have an algorithm that maintains an expander decomposition for $\phi = n^{-O(\gamma)}$. The depth of recursion would then be $O(\frac{1}{\gamma})$, and assume the

recourse per level is ρ for a new value of ρ. Then our recourse per level would be $\tilde{O}(\frac{\rho\lambda_{\max}}{\delta^3})$, and over $O(\frac{1}{\gamma})$ levels, this would add up to $\left(\frac{\rho\lambda_{\max}}{\delta}\right)^{O(\frac{1}{\gamma})}$. Each update requiring $\left(\frac{\rho\lambda_{\max}}{\phi\delta}\right)^{O(1)}$ time to be processed, this would yield a total update time of $\left(\frac{\rho\lambda_{\max}}{\delta}\right)^{O(\frac{1}{\gamma})}\frac{1}{\phi} = \left(\frac{\rho\lambda_{\max}}{\delta}\right)^{O(\frac{1}{\gamma})} n^{\gamma}$. This could be leveraged to get different tradeoffs between running time and largest acceptable $\lambda_{\max}$.

The second implication of our algorithm is an approximation algorithm for the dynamic minimum cut problem, where there are no restrictions on the size of λ nor on the edges being unweighted. Note that this is the first algorithm that maintains a $(1+\epsilon)$-approximate minimum cut in *weighted* graphs:

Theorem 6.2.4 (Fully Dynamic Approximate Mincut)**.** *Let G_w be a weighted fully dynamic graph with n nodes under edge updates, and let $\epsilon \geq 2^{-\Theta(\log^{3/4-c} n)}$ for any $c > 0$. There exists a dynamic algorithm that maintains a $(1+\epsilon)$-approximation of the value of the minimum cut in $n^{o(1)}$ update time. Moreover, a partition achieving the minimum cut is stored implicitly.*

The algorithm is randomized and works against an oblivious adversary, and is correct with high probability.

Let us quickly discuss how we obtain Theorem 6.2.4 from Theorem 6.2.2. Our first step is to sparsify the graph as to ensure that the minimum cut value is reduced to at most $O(\frac{\log n}{\epsilon^2}) < 2^{\Theta(\log^{3/4-c} n)}$ for some $c > 0$ using the edge-sampling approach of Karger [93]. It samples each edge with a probability p aptly chosen as described in the following lemma:

Theorem 6.2.5 (Theorem 2.1 and Corollary 2.1 of [93])**.** *Let G be any graph with minimum cut λ, let $\epsilon > 0$, and let $p \geq \frac{54 \ln n}{\epsilon^2 \lambda}$. Let $G(p)$ be the graph obtained from G by sampling each edge independently with probability p. Then with high probability, the minimum cut S in $G(p)$ will correspond to a $(1+\epsilon)$-minimum cut of G. Moreover, also with high probability, the value of* any *cut S in $G(p)$ has value no less than $(1-\epsilon)$ and no more than $(1+\epsilon)$ times its expected value.*

We adapt this sparsifying technique to weighted graphs, and sparsify the graph for different values of estimates of λ $\lambda_0, \ldots, \lambda_{\log_{1.1}(n^2 \cdot \frac{U}{L})}$ where $\lambda_i = L \cdot 1.1^i$ for every $i \in [\log_{1.1}(n^2 \cdot \frac{U}{L})]$. We then run the algorithm from Theorem 6.2.2 on each sparsified graph, and return the value of the algorithm with the correct estimate.

Organization. The paper is organized as follows: in Section 6.3, we formally define the problem and introduce definitions and notations. In Section 6.4, we give a technical overview of our work. In Section 6.5, we present our deterministic LocalKCut. In Section 6.6, we present our fragmenting algorithm. In Section 6.7, we present the Cluster decomposition and Hierarchy. In Section 6.8, we discuss how to maintain mirror cuts, a building block of our data structure. In Section 6.9, we present a static algorithm for the minimum proper cut problem for a specific range, and make this algorithm dynamic in Section 6.10. In Section 6.11, we present the last details needed for our data structure. And finally, in Section 6.12, we discuss how to deal with weighted graphs.

6.3 Preliminaries

Problem Definition. We are given an undirected unweighted graph $G = (V, E)$, where V is the set of vertices, and E is the set of edges. Every vertex subset $\emptyset \subsetneq S \subsetneq V$ is also called a *cut* of G. We denote by $\mathrm{Vol}(S)$ the sum of the degrees of the nodes in S. We define, for every $T \subseteq V$, a cost associated with T, that we call the *boundary-size* of T, or equivalently, the *(cut-)size* of T:

$$\partial T = |E(T, V \setminus T)|$$

Typically, we will use the term "cut-size" when T is a candidate to be the minimum cut, while we will prefer the term "boundary-size" when T is a cluster in our cluster decomposition (which is the internal data structure that we introduce later on), and thus ∂T is a value that we do not wish to compare (yet) to the cut-sizes of other cuts.

A *minimum proper cut* of G is a set $MC(G) \subsetneq V$ that minimizes the cut-size, while still being non-zero: $MC(G) \in \arg\min_{\emptyset \subsetneq T \subsetneq V, \partial T \neq 0} (\partial T)$. Note that there are dynamic algorithms with polylogarithmic time per operation that maintain the connected components of a graph (See e.g. Holm, de Lichtenberg, and Thorup [86]). However, we do not rely on them. Our algorithm detects connected components itself.

We study the problem in the *dynamic setting under edge updates*, that is, at every time step, an edge can be either inserted or deleted.

Remark 6.3.1. *We allow parallel edges. All our running times that include the number of edges count their multiplicities. For example, in a graph with two nodes and 3 parallel edges between them, we have that* $m = 3$.

Definitions and Notations. For any sets $A, B \subseteq V$, with $A \cap B = \varnothing$, we use $E(A, B)$ to denote the set of edges going from A to B and $w(A, B)$ to denote $|E(A, B)|$.

For any subset $C \subseteq V$, G/C is the graph where the set C is contracted to a single node, adding parallel edges if necessary. No self-loops are allowed.

Let $H = (V_H, E_H)$ with $V_H \subseteq V$ and $E_H \subseteq E$, be a subgraph of G. We define the boundary $\partial_H T$ of a set $T \subseteq V_H$ to be the boundary of T in H. If E_H is not defined explicitly, by convention we mean H is the subgraph of G induced by V_H. We also identify V_H and H.

A *cluster* is a set of vertices that are "highly connected". It is defined in more detail below.

Definition 6.3.1 (Crossing and uncrossing)**.** *We say that a cut S* crosses *cluster C if both $S \cap C$ and $C \setminus S$ are nonempty. A cut S'* uncrosses a cut S in a cluster C *if S crosses C and S' either equals $S \cup C$ or $S \setminus C$.*

Our algorithm only uncrosses a cut S in a cluster C when $S \cap C$ is not $(1-\delta)$-boundary sparse (see Definition 6.4.1) in C, which will imply that S' is a $(1 + O(\delta))$-approximation of S.

Definition 6.3.2 (Induced subgraph with self-loops)**.** *For any subset $U \subseteq V$, $G[U]$ is the subgraph induced by U, and $G[U]^r$ for $r \in \mathcal{R}$ is the subgraph induced by U where for every boundary edge $\{v, x\}, v \in U, x \notin U$ we add $\lceil r \rceil$ self-loops to v.*

Definition 6.3.3 (Conductance of a cut)**.** *The conductance of a cut $U \subsetneq V$ in the graph G is*

$$\phi_G(U) = \frac{w(U, V \setminus U)}{\min\{\mathrm{Vol}_G(U), \mathrm{Vol}_G(V \setminus U)\}}.$$

A graph G is a ϕ-expander if every cut in G has conductance at least ϕ.

Definition 6.3.4 ((α, ϕ)-expander)**.** *(Definition 4.1 of [77]) For a graph $G = (V, E)$ and parameters $\alpha, \phi \in (0, 1)$ and $H \in \mathcal{R}$, a subgraph $U \subset V$ is (α, ϕ)-boundary-linked expander with slack H in G if the graph $G[U]^{\frac{\alpha}{\phi}}$ is a ϕ/H-expander.*

We will maintain a partition of the graph into vertex-disjoint expanders, called *expander decomposition*. An *inter-expander* edge is an edge whose endpoints belong two different expanders. We will further maintain a *pre-cluster decomposition* and a *cluster decomposition* that is a refinement of the expander decomposition. In the pre-cluster decomposition, each *cluster* is a set of vertices and any edge that lies between two different clusters is an *inter-cluster* edge. A cluster decomposition is a refinement of the pre-cluster decomposition, and any edge that lies between two clusters of the cluster decomposition but inside a cluster of the pre-cluster decomposition is a "fragmented" edge.

To compute and maintain the expander decomposition under edge insertions and deletions we use the following result from Goranci, Räcke, Saranurak and Tan [77]. In this theorem, the term *contracted graph* refers to the graph obtained from the original graph by merging all nodes in an expander into one (for each expander). *Recourse* is the number of changes made to the contracted graph, which correspond to the number of edges that switch from being intra-cluster edges to inter-cluster edges or vice-versa.

Theorem 6.3.5 (Dynamic Expander Decomposition, Lemma 7.3 of [77])**.** *Let $\phi' = 2^{-\Theta(\log^{3/4} n)}$. Suppose a graph G initially contains m edges and undergoes a sequence of at most $O(\frac{m\phi}{\rho})$ adaptive updates such that $V(G) \leq n$ and $E(G) \leq m$ always hold. Then there exists an algorithm that maintains an (α, ϕ')-expander decomposition with slack $38^h = 2^{-O(\log^{1/2} n)}$ and its contracted graph with the following properties:*

1. *update time: $\tilde{O}(\frac{\psi 38^{2h}}{\phi^2})$*
2. *preprocessing time: $\tilde{O}(\frac{m}{\phi})$*
3. *initial volume of the contracted graph (after preprocessing): $\tilde{O}(\phi m)$*
4. *amortized recourse (number of updates to the contracted graph): $O(\rho)$*

Since the amortized recourse is $O(\rho)$, and the algorithm can handle $O(\frac{m\phi}{\rho})$ updates, this shows that overall the number of edges that are initially or became inter-cluster edges during the given sequence of updates is $O(m\phi)$.

Choice of parameter values. Throughout the paper we assume $c > 0$, $\delta = 2^{O(\log^{3/4-c} n)} \leq 0.04$. We set $\lambda_{\min} = 2^{\Theta(\log^{3/4-c} n)}$ and $\lambda_{\max} \leq 1.2\lambda_{\min}$, $H = 38^h = 2^{-O(\log^{1/2} n)}$, $\phi = \frac{\phi'}{38^h} = 2^{-\Theta(\log^{3/4} n)}$, $\rho = 2^{\Theta(\log^{1/2} n)}$ and $\alpha = \frac{1}{\mathrm{poly}\log n}$.

6.4 Technical Overview

6.4.1 Overview of [59]

As our result builds up on the work of El-Hayek, Henzinger and Li [59], we start by giving an overview of their techniques first. The core of their algorithm is a fully-dynamic subroutine that takes as input two parameters, $\lambda_{\min}$ and $\lambda_{\max}$, and is able to maintain the value of the minimum cut if that value falls in the range $[\lambda_{\min}, \lambda_{\max}]$. If it falls outside of this range, then it simply outputs "$\lambda < \lambda_{\min}$" or "$\lambda > \lambda_{\max}$" accordingly.

Let G be a graph. The starting point of the algorithm is the expander decomposition of Goranci, Räcke, Saranurak and Tan [77]. Then, the key point is to realize that any cut S of cut-size at most $\lambda_{\max}$ in G leads to a cut of cut-size at most $\lambda_{\max}$ in the graph induced by any of the expanders in the expander decomposition of G. Recall that for any ϕ-expander C, we have $\frac{w(C\cap S, C\setminus S)}{\min\{\mathrm{Vol}(C\cap S), \mathrm{Vol}(C\setminus S)\}} \geq \phi$, which implies an upper bound of $\lambda_{\max}/\phi$ on the volume of the smaller side of the cut in the expander. Finding these cuts of small volume can be efficiently done using the algorithm from Nalam and Saranurak [86], the so-called *LocalKCut algorithm*. Thus, refining the expander decomposition by repeatedly cutting expanders along cuts of cut-size at most $\lambda_{\max}$ and of volume at most $\frac{\lambda_{\max}}{\phi}$ ensures that if the minimum cut has value at most $\lambda_{\max}$, it does not cut any cluster obtained from that refined decomposition.

The problem with this approach, however, is that cutting along all such cuts can lead to a decomposition that is "too fine-grained": In the extreme case, each vertex forms its own cluster, meaning that the contracted graph is identical to the original graph. This is problematic, as the algorithm then contracts each cluster in the decomposition and calls itself recursively on the contracted graph, that, for efficiency reasons, needs to be "sufficiently smaller" than the original graph. To guarantee that the contracted graph is indeed "small enough", the idea of [59] is to filter the cuts returned by the LocalKCut algorithm and to only cut along cuts that are $(1-\delta)$-boundary sparse, a concept introduced by Henzinger, Li, Rao, and Wang [82]:

Definition 6.4.1 (Boundary-sparse)**.** *(Def. 2.3 of [82]) For a set $C \subsetneq V$ and parameter $\delta \leq 1$, a cut $U \subsetneq C$ is $(1-\delta)$-boundary sparse in C iff*

$$w(U, C \setminus U) < (1-\delta)\min\{w(U, V\setminus C), w(C\setminus U, V\setminus C)\}$$

Intuitively, the cut $U \subsetneq C$ is boundary sparse[1] in C if the number of edges connecting it to the rest of C is a $(1-\delta)$ factor smaller than the number of edges that connect it to $V \setminus C$ and, for symmetry, $C \setminus U$ also fulfills this property. Thus, if the minimum cut S is such that $S \cap C = U$, one can "uncross" S into $S \setminus C$ or $S \cup C$, whichever results in a cut of smaller cut-size, and the resulting near-minimum cut will only gain a $(1 + O(\delta))$ factor in cut-size. Therefore, from the expander decomposition, if one decomposes the expanders further along cuts that have volume at most $\frac{\lambda_{\max}}{\phi}$, cut-size at most $\lambda_{\max}$ and are $(1-\delta)$-boundary sparse, then in most cases the minimum cut can be approximated by a cut that does not cross any cluster. The only case where this does not happen is when the minimum cut is contained in at most two expanders. In that case, one can show that this cut can be retrieved by looking at the minimum cut of small volume in the so-called *mirror clusters*, where a mirror cluster of a cluster C is obtained from G by contracting all nodes outside of C into one node.

[1] Boundary sparseness indicates *sparseness in comparison to the boundary*.

Therefore, the algorithm maintains the following invariant for its cluster decomposition: at no point in time does there exist a cut in a cluster that (1) has volume at most $\frac{\lambda_{\max}}{\phi}$ in the cluster and (2) cut-size at most $\lambda_{\max}$ and (3) is $(1-\delta)$-boundary sparse. It also maintains the mirror clusters and the cuts of small volume in it.

6.4.2 Our techniques

To prove Theorem 6.2.2 we (1) derandomize the LocalKCut algorithm and (2) improve the approximation ratio of the algorithm in [59] which we described above.

Throughout this paper, $\lambda_{\max}$ is set to be at most $2^{O(\log^{3/4-c} n)}$ with $c > 0$. As discussed below, this is because the recourse of our algorithm is dependent on $\lambda_{\max}$. Over $r = \sqrt[4]{\log n}$ many levels of recursion, this is at most $\lambda_{\max}^{O(\sqrt[4]{\log n})}$. However to achieve an $n^{o(1)}$ update time, this value has to be $n^{o(1)}$, which in turn requires that $\lambda_{\max}$ is at most $2^{O(\log^{3/4-c} n)}$ with $c > 0$.

A deterministic LocalKCut algorithm

The input of the LocalKCut algorithm is a vertex v, a volume ν and a cut-size $\lambda_{\max}$, and a cluster C and its goal is to return all cuts in C that contain v, are of volume (in $G[C]$) at most ν, are of cut-size at most $\lambda_{\max}$, and are connected. The LocalKCut algorithm by Nalam and Saranurak [112] heavily relies on randomization, as at its core, the algorithm starts at v, and randomly explores the graph checking at each step if the set of all vertices visited until that point form a cut that satisfies the desired conditions or not. They show that with a "high enough" probability, any cut that satisfies the desired conditions is found in this way, and thus, running this algorithm enough times guarantees that all such cuts are found whp.

We, thus, have to develop a whole new technique to derandomize it. For that, we rely on the *greedy tree packing technique* developed by Karger [91]. The tree packing technique works as follows: start with the graph itself, assign to each edge a weight of 0, and create an empty set of trees, called the *tree packing*. Then iteratively find a minimum spanning tree of the graph, add it to the tree packing, and increase the weight of every edge in that tree by 1.

Karger showed that if we choose enough trees in the tree-packing, then for every minimum cut S, there must be a tree that 2-*respects* S, that is, the edges of the minimum cut intersect the edges of the tree at most twice. We show an immediate generalization of this result: for any β-approximation of the minimum cut, there exists a tree that 2β-respects the cut for a suitable integer constant β. Assume for the moment that any cut in G of cut-size at most $\lambda_{\max}$ is a β-approximation of the minimum cut.

We discuss below how to guarantee this condition for the graphs on which LocalKCut is executed.

Now we implement a deterministic version of LocalKCut inspired by techniques of Lokshtanov, Saurabh, and Surianarayanan [103] for approximate minimum k-cut as follows: A *red-blue edge coloring* is a coloring of the edges where every edge is either red or blue, but not both. We assume for the moment that we are given a family $\mathcal{F}_1$ of red-blue edge colorings and also a family $\mathcal{F}_2$ of green-yellow edge colorings and discuss later how we find and maintain such colorings efficiently. The first family $\mathcal{F}_1$ consists of a collection of red-blue edge colorings such that for any two disjoint edge sets F_r, F_b with $|F_r| \leq 2\beta$ and $|F_b| \leq \nu$, there exists a coloring

in $\mathcal{F}_1$ with all edges in F_r red and all edges in F_b blue. The second family $\mathcal{F}_2$ consists of a collection of green-yellow edge colorings such that for any two disjoint edge sets F_g, F_y with $|F_g| \leq 2\beta - 1$ and $|F_y| \leq \lambda_{\max}$, there is a coloring in $\mathcal{F}_2$ with all edges in F_g green and all edges in F_y yellow. We use the term *pair of colorings* to denote a pair of colorings consisting of one coloring of $\mathcal{F}_1$ and one of $\mathcal{F}_2$.

The algorithm maintains during the sequence of edge updates a greedy tree packing of G, and the two edge coloring families $\mathcal{F}_1$ and $\mathcal{F}_2$.

Then, whenever LocalKCut is started on a vertex v it performs, for every pair of colorings and every tree T, a breadth-first search *using only the blue edges that belong to T and the green edges that do not belong to T*. Red edges and yellow edges are ignored. If the BFS visits nodes of total volume more than ν, LocalKCut stops the current BFS, and moves to the next pair of colorings. If the BFS cannot visit any new vertices and is still below the volume threshold of ν, LocalKCut checks whether the resulting cut induced by the visited vertices fulfills the desired conditions on cut-size and volume (i.e. has edge capacity at most $\lambda_{\max}$ and volume at most ν) and if so, it adds the cut to the collection of cuts to be returned. It then moves on to the next pairs of colorings.

We claim that this deterministic version of LocalKCut finds every cut S that contains v, is connected, has volume at most ν, and cut-size at most $\lambda_{\max}$ using the following argument and setting $\beta \geq 8$, as long as S is a β-approximation of the minimum cut in C.

As, by assumption, S is β-approximation of the minimum cut in C, we are guaranteed that there exists a tree T in our tree packing that 2β-respects S, i.e., at most 2β of the edges of T cross the cut induced by S. Choose F_r to be these cut edges and choose F_b to be all the edges of T with both endpoints in S. By the conditions that S fulfills, it follows that $|F_r| \leq 2\beta$ and $|F_b| \leq \nu$. Hence there exists a blue-red coloring in $\mathcal{F}_1$ where all edges in F_r are red, and all edges in F_b are blue. As the edges in F_b are edges of T (that spans the cluster C) and they are connected by at most 2β (red) tree edges to the rest of T, they can form at most 2β distinct connected components. As S is connected, these components can be connected by at most $2\beta - 1$ edges of G. Let F_g be a minimum-cardinality set of edges that connect these components. Then $|F_g| \leq 2\beta - 1$ and all these edges have both endpoints in S. Furthermore, let $F_y = E(S, V \setminus S)$. Note that $|F_y| \leq \lambda_{\max}$. Therefore, there exists a green-yellow coloring in $\mathcal{F}_2$ where all the edges in F_y are yellow and all edges in F_g are green. As a result it holds that $F_b \cup F_g$ spans S and all tree edges leaving S are red and and all non-tree edges leaving S are yellow. Recall that the algorithm explores the graph in BFS fashion using only the blue tree edges and the green non-tree edges for all trees in our tree packing and all pairs of edge colorings in $\mathcal{F}_1 \times \mathcal{F}_2$. Thus, when our algorithm described above runs with that particular tree and these particular colorings it is guaranteed to find S as $F_b \cup F_g$ spans S and none of the tree edges leaving S are blue and none of the non-tree edges leaving S are green.

In the above argument we used the fact that every cut in G of cut-size at most $\lambda_{\max}$ is a β-approximation of the minimum cut. Note that our algorithm in [59] only runs LocalKCut on either clusters or mirror clusters, and not on arbitrary graphs G. Therefore, if we can ensure that the minimum cut-size is at least $\frac{\lambda_{\max}}{\beta}$ in every cluster and mirror cluster, and then run our deterministic LocalKCut algorithm on every cluster and mirror cluster, we are guaranteed to find all cuts that we need.

To ensure this property we note that if a cut in a cluster has cut-size smaller than $\frac{\lambda_{\max}}{\beta}$ with $\beta > 3$, and if the graph has minimum cut at least $\lambda_{\min}$, then the cut must be $(1-\delta)$-boundary sparse in the cluster. Note that the correctness proof for the cluster decomposition [59] (and we will we will also base our correctness proof on it) does not place any conditions on cutting clusters other than requiring that the cuts are $(1-\delta)$-boundary-sparse. Thus, our algorithms is allowed to cut the cluster along any cut of cut-size smaller than $\frac{\lambda_{\max}}{\beta}$, and do so repeatedly until no such cut remains. After recursively cutting all these small cuts the cluster has the desired property and we can run our deterministic LocalKCut routine on it.

The question remains on how to efficiently (1) find such cuts and (2) how many colorings are needed and how to find and maintain them dynamically.

(1) We need determine new cuts that fulfill the requirements of our LocalKCut algorithm after each edge update and if an update splits a custer, we need to run our LocalKCut algorithm recursively on the new clusters. To do so we maintain a LocalKCut data structure that allows to quickly output all desired cuts after an edge update. However, after an update a cluster C might be split into two clusters that are not connected with each other in C, rendering the LocalKCut data structure for this cluster useless since it only works for β-approximations of the minimum cut in that cluster, which is now 0. We could create a separate LocalKCut data structure for each cluster, creating new LocalKCut data structures whenever a cluster is split into two, but this proves to be too slow.

Instead, our solution is to not run and maintain one LocalKCut data structure per cluster, but instead build one LocalKCut data structure *for all clusters*, i.e., for the whole graph. Thus, to split a cluster it suffices to simply delete from the data structure the edges that connect the two new clusters. This also works fine with our constructions in general, although one needs to be careful. For example, the greedy tree packing for our LocalKCut data structure now needs to be replaced by a greedy *forest* packing, which is a greedy tree packing on each of the connected components. The resulting LocalKCut data structure maintains a subgraph G' of G and implements the following operations:

Definition 6.4.2 (LocalKCut)**.** *Let G be a graph that is updated by a sequence of edge insertions and deletions. A LocalKCut data structure parametrized by $\nu \in \mathbb{N}, s \in \mathbb{N}, \beta \in \mathbb{N}$ maintains a subgraph G' of G and implements the following operations:*

(i) LocalKCut query(v): Return all connected cuts in G' containing v of volume (in G') at most ν and cut-size in G' at most s, that are a β-approximation of the minimum proper cut of G'.

(ii) DeleteEdge(e): Remove the edge e from G'.

(iii) InsertEdge(e): Add the edge e to G'.

For our LocalKCut data structure the time for each LocalKCut query is $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$. The amortized time for every DeleteEdge and InsertEdge operations is $\tilde{O}\left(\beta^4(\lambda_{\max}\nu)^{O(\beta)}\right)$. Using $\beta = O(1)$ and $\lambda, \nu = n^{o(1)}$ we get that every operation takes amortized time $n^{o(1)}$.

(2) Let us now discuss how to find the red-blue and yellow-green colorings. For that, we use the following lemma which was shown by Chitnis, Cygan, Hajiaghayi, Pilipczuk and Pilipczuk [39]

and that guarantees the existence of a family of colorings of a set of elements and gives a fast algorithm for finding them. In our case each element is an edge.

Lemma 6.4.3 (Lemma 1.1 of [39]). *Given a set U of cardinality n, and integers $0 \leq a,\ b \leq n$, one can in time $2^{O(\min(a,b)\log(a+b))}n\log n$ construct a family $\mathcal{F}$ of at most $2^{O(\min(a,b)\log(a+b))}\log n$ subsets of U, such that the following holds: for any sets $A, B \subseteq U$, $A \cap B = \emptyset$, $|A| \leq a$, $|B| \leq b$, there exists a set $S \in \mathcal{F}$ with $A \subseteq S$ and $B \cap S = \emptyset$.*

In our case, U is the set of edges, and a and b are respectively 2β and ν for the red-blue coloring, and $2\beta - 1$ and $\lambda_{\max}$ for the green-yellow coloring. Plugging in $\lambda_{\max}, \nu = n^{o(1)}$ and $\beta = O(1)$ we have that at most $n^{o(1)}$ many of these colorings are necessary to ensure the properties that we need.

Another feature of this construction is that it is remarkably robust against updates: indeed, if an edge is deleted, the colorings do not have to change, and the properties we require still hold. If another edge is then inserted, then it can take over the colors of the deleted edge, and the properties still hold, as it is a set-theoretic result, and not a graph-theoretic one. Therefore, when computing colorings, one can do so for $2m$ many edges, and assign only the first m colors of a coloring to the existing edges, sparing the m other ones for potential edge insertions. These colorings will thus be valid until the number of edges in the graph doubles. When this happens we can afford to compute new family of colorings. Overall, we spend $m^{1+o(1)}$ time to compute these colorings, that we can amortize over at least m updates, for an update time of at most $n^{o(1)}$.

Getting a better approximation ratio

In [59], the approximation ratio is $\left(1 + \Theta(\frac{1}{\sqrt{\log n}})\right)$. As a consequence, if we want to use this algorithm to compute an exact answer, the minimum cut can only be as large as $O(\sqrt{\log n})$, as otherwise the approximation ratio is not small enough to guarantee that the answer is exact.

The reason for such a weak approximation ratio is that cutting along $(1-\delta)$-boundary sparse cuts is done in an arbitrary order. As a result, we only achieve an upper bound on the total number of inter-cluster edges created between any two rebuilds of the expander decomposition of roughly $M2^{O(\frac{1}{\delta})}$, where M is the number of inter-expander edges in the expander decomposition, which is typically $m\phi = m2^{-\Theta(\log^{3/4} n)}$. We need ϕ to dominate $2^{O(\frac{1}{\delta})}$ to ensure that the total number of inter-cluster edges can still be bounded by $m2^{-\Theta(\log^{3/4} n)}$ and, thus, in [59] δ cannot be truly smaller than $\Omega(\frac{1}{\log^{3/4} n})$. In this work we aim for $\delta = 2^{-O(\log^{3/4-c} n)}$ with $c > 0$, and thus need a cluster decomposition with at most $O(\frac{M}{\text{poly}(\delta)})$ many inter-cluster edges. Since $M = O(m\phi)$, the ϕ factor dominates the $\frac{1}{\text{poly}(\delta)}$ factor, as $\phi = 2^{-\Theta(\log^{3/4} n)}$ and $\delta = 2^{-O(\log^{3/4-c} n)}$. We are, thus, able to show an approximation ratio of $1 + 2^{-O(\log^{3/4-c} n)}$, which implies that our new algorithm gives an exact solution for $\lambda_{\max} = 2^{O(\log^{3/4-c} n)}$ with $c > 0$, improving on the result of [59].

We now explain how to achieve a better approximation ratio. Henzinger, Li, Rao, and Wang, who presented the first near-linear time deterministic *static* exact minimum cut algorithm in weighted graphs [82], also cut clusters along $(1-\delta)$-boundary sparse cut. They are able to get the desired $O(\frac{M}{\text{poly}(\delta)})$ upper bound on the number of inter-cluster edges by performing the cuts

in a different manner than [59]: If the cluster has high boundary-size (say, at least $3\lambda_{\max}$), they cut the cluster along any boundary sparse cuts; if, however, the cluster has small boundary-size, then they call a subroutine, which we name *fragmenting*, which carefully selects the order in which the $(1-\delta)$-boundary sparse cuts are performed, and outputs which edges need to be cut to get the decomposition – that is, which edges need to become inter-cluster edges. Fragmenting runs in time polynomial in the volume of the cluster that is decomposed, but this is not a problem for them as the clusters they consider have only volume polylogarithmic in n.

We cannot run this algorithm in a black-box manner for the following reasons: (a) The cluster we want to fragment might have large volume as they might be the expanders of the expander decomposition, and (b) their algorithm is static and not dynamic. Thus, we instead adapt their fragmenting algorithm to our needs, solving both problems simultaneously: we give a static deterministic algorithm that runs in $n^{o(1)}$ time on clusters of boundary-size $O(\lambda_{\max}) = n^{o(1)}$ and has output of size $\tilde{O}(\frac{\lambda_{\max}}{\text{poly}(\delta)}) = n^{o(1)}$. This is fast enough so that we can simply rerun it after every update on each affected cluster. Running the algorithm from scratch after every update makes it dynamic: Since its output is of size $\tilde{O}(\frac{\lambda_{\max}}{\text{poly}(\delta)})$, which corresponds to the cut-size of the cuts along which the cluster has to be partitioned, the recourse is $\tilde{O}(\frac{\lambda_{\max}}{\text{poly}(\delta)})$. Its static running time of $n^{o(1)}$ translates then in a $n^{o(1)}$ time per update on clusters of boundary-size at most $O(\lambda_{\max})$.

To speed up the algorithm we rely on two observations: the first one is that, for clusters C of small boundary-size, that is, $\partial C = O(\lambda_{\max})$, the running time of the fragmenting algorithm is dominated by the time needed to find the set of all $(1-\delta)$-boundary sparse cuts in the cluster. We can replace this step by running our version of LocalKCut instead, only looking at cuts of small volume. Since any $(1-\delta)$-boundary sparse cut must have at least one boundary edge, it thus suffices to run LocalKCut from every node incident to a boundary edge. This takes $n^{o(1)}$ time per LocalKCut query. As the number of queries is $O(\lambda_{\max}) = n^{o(1)}$ exploring only volume $\nu = n^{o(1)}$ in C, this leads to a $\lambda_{\max} n^{o(1)} = n^{o(1)}$ overall running time for the fragmenting algorithm.

The second observation is that the number of edges that our fragmenting algorithm cuts, that is, the number of inter-cluster edges that it creates, is $\tilde{O}(\frac{\partial C}{\delta^2})$, as we can show that it creates at most $\tilde{O}(\frac{1}{\delta^2})$ many clusters of cut-size at most $\partial C = O(\lambda_{\max})$ each. This number increases the recourse of our cluster decomposition. As long as $\delta = 2^{-O(\log^{3/4-c} n)}$ and $\lambda_{\max} = 2^{O(\log^{3/4-c} n)}$, the additive increase in the recourse is still dominated by $\frac{1}{\phi}$, and thus we can afford such a recourse increase for the cluster decomposition. We can therefore run our fragmenting algorithm from scratch on every cluster that is affected by an update of the dynamic expander decomposition as both the running time and the recourse stay sufficiently bounded.

Overall, to maintain our cluster decomposition, we run three dynamic algorithms back-to-back: the dynamic expander decomposition, the pre-cluster decomposition that cuts along $(1-\delta)$-boundary-sparse cuts arbitrarily as long as the clusters have boundary-size at least $\Omega(\lambda_{\max})$, and the fragmenting algorithm that only runs on clusters of boundary-size $O(\lambda_{\max})$ and that chooses which $(1-\delta)$-boundary-sparse cuts to cut and in which order. The expander decomposition has recourse $\rho = 2^{O(\sqrt{\log n})}$, the pre-cluster decomposition has recourse $O(\frac{1}{\delta})$ for each update caused by the expander decomposition, while the fragmenting algorithm has recourse $\tilde{O}(\frac{\lambda_{\max}}{\delta^2})$ for each update caused by any of the prior two decomposition algorithms. Overall we thus get a recourse which is a product of these three factors, i.e., of $\tilde{O}(\frac{\rho\lambda_{\max}}{\delta^3}) =$

$2^{O(\log^{3/4-c} n)}$ per level of recursion. As we show that the recursive depth is at most $O(\sqrt[4]{\log n})$, this yields a total recourse of $2^{O(\log^{1-c} n)}$, which we can process in $n^{o(1)}$ time per change, thus $n^{o(1)}$ overall time.

Hence, we get an algorithm that in $n^{o(1)}$ time per update maintains a cluster hierarchy, that is, a collection of graphs associated to a cluster decomposition each, where each graph is obained from the previous graph by contracting each cluster. As in [59], we show that any minimum proper cut is well-approximated by a cut of small volume in a graph of this hierarchy. We thus maintain additionally, for each vertex v, the best proper cut of small volume in which v is contained if there exists one small enough, store them in a heap data structure, and simply return the cut of smallest cut-size among all these cuts to obtain our result and return an approximate minimum cut.

6.5 A deterministic LocalKCut data structure

In this section, we present a deterministic LocalKCut data structure that achieves the following bounds:

Theorem 6.5.1. *For any parameters $\lambda_{\max}, \nu, \beta \in \mathbb{N}_+$ with $\lambda_{\max} < \nu$, there is a deterministic, fully dynamic LocalKCut data structure that outputs, on each LocalKCut query specified by a vertex $v \in V$, a collection $\mathcal{S}$ of cuts $S \subseteq V$ containing v such that*

1. *Each cut $S \subseteq V$ satisfies $\partial S \leq \lambda_{\max}$, $\mathbf{vol}(S) \leq \nu$, and $G[S]$ is connected.*
2. *$\mathcal{S}$ contains all β-approximate mincuts $S \subseteq V$ satisfying condition (1).*

The data structure runs in $\tilde{O}(m\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ preprocessing time, $\tilde{O}\left(\beta^4(\lambda_{\max}\nu)^{O(\beta)}\right)$ amortized update time and $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ query time.

The goal is to set $\beta = O(1)$, $\lambda_{\max} = n^{o(1)}$ and $\nu = n^{o(1)}$ so that the preprocessing time is $m^{1+o(1)}$ and the update and query time are $n^{o(1)}$.

6.5.1 Description of the algorithm and proof of correctness

The full description of the algorithm can be found in Algorithm 6.5.1, but we present it step by step here. We begin with some preliminaries from [125].

A *tree packing* is an assignment of weights to spanning trees of G so that each edge e gets *load*

$$\ell(e) = \sum_{T \ni e} w(T) \leq 1.$$

The value of the tree packing is $\sum_T w(T)$. We let τ denote the value of the maximum tree packing of G. The classic Nash-Williams' theorem relates τ to the mincut λ of the graph.

Theorem 6.5.2 (Nash-Williams [113])**.** $\lambda/2 \leq \tau \leq \lambda$.

A tree packing is $(1+\epsilon)$-*approximate* if $\sum_T w(T) \geq \tau/(1+\epsilon)$. In order to use the same parameters as prior work, we slightly abuse the notation in this section: The ϵ in this section is independent from the ϵ from all other sections.

Theorem 6.5.3 (Generalisation of [91])**.** *For any β-approximate mincut and any $(1+\epsilon)$-approximate tree packing, there is a tree in the packing that $\lfloor 2(1+\epsilon)\beta \rfloor$-respects the β-approximate mincut.*

Proof. Since the tree packing $\mathcal{T}$ is $(1+\epsilon)$-approximate, we have $\sum_T w(T) \geq \tau/(1+\epsilon)$. If we sample a random tree $T \in \mathcal{T}$ with probability proportional to $w(T)$, then the probability that the sampled tree contains a given edge e is

$$\frac{\sum_{T \ni e} w(T)}{\sum_T w(T)} \leq \frac{1}{\sum_T w(T)} \leq \frac{1}{\tau/(1+\epsilon)}.$$

For a given β-approximate mincut, the expected number of edges in the sampled tree is at most

$$\beta\lambda \cdot \frac{1}{\tau/(1+\epsilon)} = (1+\epsilon)\beta\lambda/\tau,$$

which is at most $2(1+\epsilon)\beta$ by Nash-Williams' theorem. It follows that there exists a tree $T \in \mathcal{T}$ that $\lfloor 2(1+\epsilon)\beta \rfloor$-respects the β-approximate mincut. □

Our goal now is to have good approximate tree packings. An easy way to get an approximate tree packing is by looking at *Greedy Tree Packings*, popularized by Karger and Thorup [125, 123]. Let H be a connected unweighted graph. A greedy tree packing with k trees is a packing obtained using the following procedure: start with H with weights 0 on all edges. Take a minimum spanning tree on H, add it to the packing, then increase the weights of all edges included in that tree. Repeat the instructions in the former sentence until k trees are in the packing.

Theorem 6.5.4 (Lemma 5 of [123])**.** *A greedy tree packing with at least $6\lambda \log m/\epsilon^2$ trees is $(1+\epsilon)$-approximate.*

We modify this slightly and consider a *Greedy Forest Packing*, where we take the exact same procedure as for greedy tree packings, but the graph G does not have to be connected, and the minimum spanning tree is replaced by a minimum spanning forest, that is, a forest where each tree is a minimum spanning tree of its connected component in G.

The idea here is that a greedy forest packing is essentially a greedy tree packing on each of the connected components of G, which has the nice properties discussed above that we will harness.

We first discuss how to maintain a greedy forest packing dynamically:

Theorem 6.5.5 (Theorem 8 of [86])**.** *There is a fully dynamic deterministic Minimum Spanning Forest algorithm that for a graph with n vertices, starting with no edges, maintains a minimum spanning forest in $O(\log^4 n)$ amortized time per edge insertion or deletion.*

As discussed in Section 3.2 of [125], by adding priorities to the edges, one can ensure that there is a unique minimum spanning forest in each graph. This leads to the following lemma, which we prove similarly to [125]:

Lemma 6.5.6. *Each insertion or deletion of an edge changes at most i edges in the i-th forest.*

Proof. We will consider only edge deletions, as the edge insertions are treated symmetrically.

Let (u, v) be the deleted edge.

- If the edge deletion disconnects a component, then the edge load drops to 0. On each connected component of G, the tree packing does not change, as otherwise the forest packing before the deletion was not greedy. We have thus only 1 edge change on the i-th forest, for each i.
- If the edge deletion does not disconnect a component: For any edge e, we define $\ell_i(e)$ to be the number of forests with index smaller or equal to i that contain e. This is the *load* of e after forst i. We distinguish $\ell_i^{old}(e)$ from $\ell_i^{new}(e)$, the former being the load of e before the edge deletion, and the latter being the load of e after the edge deletion.

  Then we prove by induction that, for every $j \geq 1$:

  $$\ell_j^{old}(e) \leq \ell_j^{new(e)} \qquad \forall e \in E$$

  that is, the edge deletion only increases the loads for all j.

  This, coupled to the fact that for each j, the total load of the edges apart from (u, v) is equal to $j \cdot (n - \text{number of components of } G)$, proves that at most j edges e satisfy $\ell_j^{old}(e) < \ell_j^{new}(e)$.

  - Base case: for $j = 1$, if an edge is deleted within a component, this edge is replaced by another one in the acyclic matroid.
  - Induction step: Assume we have that $\ell_j^{old}(e) \leq \ell_j^{new(e)} \quad \forall e \in E$, and we want to show that $\ell_{j+1}^{old}(e) \leq \ell_{j+1}^{new(e)} \quad \forall e \in E$. There are three cases:
    * Either $f = (u, v)$ is in the $j + 1$-th forest after the edge deletion. Then $\ell_{j+1}^{new}(f) = \ell_j^{new}(f) + 1 \geq \ell_j^{old}(f) + 1 \geq \ell_{j+1}^{old}(f)$,
    * or $f = (u, v)$ was not in the $j + 1$-th forest before the edge deletion. Then $\ell_{j+1}^{old}(f) = \ell_j^{old}(f) \leq \ell_j^{new}(f) \leq \ell_{j+1}^{new}(f)$,
    * or $f = (u, v)$ was in the $j + 1$-th forest before the deletion and is not in the forest after the deletion. This means that before the edge deletions, f was in the minimum spanning forest in G with weights ℓ_j^{old}, but is not anymore after the deletions with weights ℓ_j^{new}. Since, by induction, none of the weights have decreased between ℓ_j^{old} and ℓ_j^{new}, this means that the load on f has strictly increased: $\ell_j^{new} \geq \ell_j^{old} + 1$. Hence: $\ell_{j+1}^{new}(f) = \ell_j^{new}(f) \geq \ell_j^{old}(f) + 1 = \ell_{j+1}^{old}(f)$.

□

This, in turn, ensures that we can maintain a dynamic greedy forest packing with k trees in $\tilde{O}(k^2)$ time per update operations

Theorem 6.5.7 (Direct Generalization of Theorem 4 of [125])**.** *For any parameter $\lambda_{\max}$, there is a fully dynamic algorithm that maintains a greedy forest packing of $\tilde{O}(\lambda_{\max}/\epsilon^2)$ trees in $\tilde{O}(\lambda_{\max}^2/\epsilon^4)$ amortized time per update. Whenever the graph has edge connectivity $\leq \lambda_{\max}$, the tree packing is $(1+\epsilon)$-approximate. Whenever the edge connectivity of the graph is $> \lambda_{\max}$, the tree packing has no guarantees.*

Our dynamic algorithm calls Theorem 6.5.7 with parameters $\lambda_{\max}$ and $\epsilon = \frac{1}{3\beta}$. Remember that we want to find all cuts containing v that have volume smaller than ν, cut-size smaller than $\lambda_{\max}$ and are β-approximations of the minimum cut. By Theorem 6.5.3, this cut 2β-respects a tree in the forest packing. Therefore, it suffices to implement the following dynamic data structure, and run the queries on each tree currently in the packing.

Lemma 6.5.8. *For any parameters $\lambda_{\max}, \nu, \beta \in \mathbb{N}_+$, there is a fully dynamic algorithm that outputs, on each query specified by a vertex $v \in V$ and a spanning tree T of the connected component of v, the collection of all cuts $S \subseteq V$ satisfying $\partial S \leq \lambda_{\max}$, $\mathbf{vol}(S) \leq \nu$, $G[S]$ is connected, and S 2β-respects the tree T. The algorithm runs in $\tilde{O}(\lambda_{\max}\nu)^{O(\beta)}$ time per update and query.*

More precisely, the proof of Theorem 6.5.1 proceeds as follows: Given a query vertex $v \in V$, our dynamic algorithm runs Lemma 6.5.8 on the queries (v, T) for each T currently in the packing, and takes the union of all sets $\mathcal{S}$ returned by the queries. The final running time is $\tilde{O}(\lambda_{\max})$ many calls, one per tree, each taking $\tilde{O}(\lambda_{\max}\nu)^{O(\beta)}$ time, which is $\tilde{O}(\lambda_{\max}\nu)^{O(\beta)}$ overall.

It remains to prove Lemma 6.5.8. The algorithm is inspired by the fixed-parameter tractable (FPT) algorithm for approximate minimum k-cut [103]. The algorithm maintains two families of 2-colorings of edges of the graph. The first family consists of red-blue edge colorings such that for any disjoint sets F_r, F_b of edges with $|F_r| \leq 2\beta$ and $|F_b| \leq \nu$, there is a coloring with all edges in F_r red and all edges in F_b blue. We discuss the size and maintenance of this family later on. The second family consists of green-yellow edge colorings such that for any disjoint sets F_g, F_y of edges with $|F_g| \leq 2\beta - 1$ and $|F_y| \leq \lambda_{\max}$, there is a coloring with all edges in F_g green and all edges in F_y yellow.

Algorithm for answering a LocalKCut Query. The algorithm iterates over all pairs of colorings, one red-blue coloring and one green-yellow coloring. For each pair of colorings, the algorithm runs a depth-first search in the subgraph H consisting of blue edges in the tree T and green edges from the graph that are not in T. If this depth-first search for this pair of colorings encounters more than ν edges, then it terminates early, spending time $O(\nu)$. Let $S \subseteq V$ be the set of vertices visited by the search. If S satisfies $\partial S \leq \lambda_{\max}$, $\mathbf{vol}(S) \leq \nu$, $G[S]$ is connected, and S 2β-respects the tree T, then add it to the final collection $\mathcal{S}$ then proceeds to the next pair of colorings.

Correctness. We claim that any set S satisfying the above requirements is added to $\mathcal{S}$ for some pair of colorings. To prove this, it suffices to show that S is precisely the connected component containing v in the subgraph H on some pair of colorings, so that the depth-first search visits

exactly the vertices in S. Since $|S| \leq \mathbf{vol}(S) \leq \nu$, there are at most $|S| - 1 \leq \nu - 1$ many tree edges in $G[S]$, and since S 2β-respects T, there are at most 2β tree edges in ∂S. So there is a red-blue coloring with all tree edges in $G[S]$ colored blue, and all tree edges in ∂S colored red. Moreover, since S 2β-respects T, the tree edges in $G[S]$ form at most 2β connected components. Let $1 \leq j < 2\beta$ be the number of connected components formed by the tree edges in $G[S]$. Then these connected components are connected in G by exactly $j-1$ non-tree edges $\mathcal{E}$ because $G[S]$ is connected. There is a green-yellow coloring such that the edges $\mathcal{E}$ (if $\mathcal{E} \neq \emptyset$) are green and all edges in ∂S are yellow (of which there are at most $\lambda_{\max}$). In the iteration where these two colorings are considered, by construction S is the connected component containing v in the graph H, and we are done.

It remains to bound the number of colorings. We use the lemma below.

Lemma 6.4.3 (Lemma 1.1 of [39]). *Given a set U of cardinality n, and integers $0 \leq a, b \leq n$, one can in time $2^{O(\min(a,b)\log(a+b))} n \log n$ construct a family $\mathcal{F}$ of at most $2^{O(\min(a,b)\log(a+b))} \log n$ subsets of U, such that the following holds: for any sets $A, B \subseteq U$, $A \cap B = \emptyset$, $|A| \leq a$, $|B| \leq b$, there exists a set $S \in \mathcal{F}$ with $A \subseteq S$ and $B \cap S = \emptyset$.*

At the beginning of the algorithm, and at every point where the number of edges double compared to the last recomputation, we set U to be a universe of elements of size at most $2m$ (where m is the number of edges at this point in time) and associate each current edge of the graph with a unique element in U. On each edge insertion, associate the new edge to an unassigned element in U. If there are no unassigned elements remaining, then recompute U to be double the size. This way, the time spent recomputing is amortized among the edge insertions.

For the family of red-blue colorings, we set $a = 2\beta$ and $b = \nu$ and map each set $S \in \mathcal{F}$ to the coloring that colors all edges with associated element in S red and all other edges blue. This family has size $(2\beta)^{O(\log \nu)} \log n = \nu^{O(\beta)} \log n$. For the family of green-yellow colorings, we set $a = 2\beta - 1$ and $b = \lambda_{\max}$ and map each set $S \in \mathcal{F}$ to the coloring that colors all edges with associated element in S green and all other edges yellow. This family has size $(2\beta)^{O(\log \lambda_{\max})} \log n = \lambda_{\max}^{O(\beta)} \log n$.

The pseudocode of the algorithm is given next.

Algorithm 6.5.1 (Deterministic LocalKCut). *Input:*

1. *A dynamic unweighted graph G*
2. *A maximum cut-size $\lambda_{\max} \in \mathbb{N}_+$*
3. *A maximum volume $\nu \in \mathbb{N}_+$, $\nu > \lambda_{\max}$*
4. *An approximation ratio $\beta \in \mathbb{N}_+$, $\beta < \lambda_{\max}$.*

Preprocessing and maintained variables:

1. *Set $\epsilon = \frac{1}{3\beta}$*

2. *Start an instance of Theorem 6.5.7 – a greedy forest packing with $6\lambda_{\max}\log m/\epsilon^2$ forests.*
3. *Create red-blue colorings of $2m$ edges with $a = 2\beta$ and $b = \nu$ using Lemma 6.4.3.*
4. *Create yellow-green colorings of $2m$ the edges with $a = 2\beta$, $b = \lambda_{\max}$ using Lemma 6.4.3.*

Processing an edge insertion:

1. *Feed it to Theorem 6.5.7, updating the greedy forest packing.*
2. *Give it an unused color for each of the colorings. If there are no unused colors, create new colorings of double the size.*

Processing an edge deletion:

1. *Give it to Theorem 6.5.7, updating the greedy forest packing.*
2. *Free its colors in each coloring to make it available for a future edge insertion.*

Processing a request on node v:

1. $\mathcal{S} \leftarrow \varnothing$
2. *For each forest, red-blue and yellow-green colorings:*
   a) *Do a DFS starting at v, exploring edges that are blue if they are in the forest, yellow if not in the tree, capped at ν volume visited.*
   b) *If the DFS terminates before the cap, let S be the set of explored vertices. If $\partial S \leq \lambda_{\max}$, add S to $\mathcal{S}$.*
3. *return $\mathcal{S}$.*

6.5.2 Running time analysis

Lemma 6.5.9. *The preprocessing time of Algorithm 6.5.1 is $\tilde{O}(m\beta^2(\lambda_{\max}\nu)^{O(\beta)})$.*

Proof. Step 1 of preprocessing takes constant time. Step 2 takes $O(m\cdot\log^4 m\cdot 6\lambda_{\max}\log m/\epsilon^2)$. Indeed, we initialize the forests by inserting each edge one by one in the first instance of Theorem 6.5.5, then on to the second one, and so on until the last one. Each edge inserted takes $O(\log^4 n)$ time, and there are $O(m)$ many edge, each inserted into $6\lambda_{\max}\log m/\epsilon^2$ many trees. Setting ϵ to $\frac{1}{3\beta}$ we get that Step 2 takes $\tilde{O}(m\lambda_{\max}\beta^2)$. Step 3 takes, by Lemma 6.4.3, $2^{O(\min(2\beta,\nu)\log(2\beta+\nu))}m\log m = \tilde{O}(m\nu^{O(\beta)})$, where we use $\beta < \nu$. Step 4 takes, by Lemma 6.4.3, $2^{O(\min(2\beta,\lambda_{\max})\log(2\beta+\lambda_{\max}))}m\log m = \tilde{O}(m\lambda_{\max}^{O(\beta)})$, where we use $\beta < \lambda_{\max}$. □

Lemma 6.5.10. *The amortized update time of Algorithm 6.5.1 on an update is $\tilde{O}\left(\beta^4(\lambda_{\max}\nu)^{O(\beta)}\right)$.*

Proof. Let us first consider an edge insertion. Step 1 takes $O(\lambda_{\max}^2\beta^4)$ by Theorem 6.5.7, where $\epsilon = \frac{1}{3\beta}$. Step 2 takes $O(1)$ time if we do not compute new colorings, but $\tilde{O}(m(\lambda_{\max}\nu)^{O(\beta)})$ if we need to compute new ones, as discussed in the previous proof. However, if it is the case, it means the number of edges have been doubled since the last time colorings were computed, and thus we can amortize this recomputation over the last $\frac{m}{2}$ edges. This yields an amortized update time of $\tilde{O}((\lambda_{\max}\nu)^{O(\beta)})$.

The case for edge deletions is similar (and easier as we do not need to recompute any colorings). □

Lemma 6.5.11. *The query time of Algorithm 6.5.1 is $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$.*

Proof. For every forest in the forest packing, red-blue coloring and yellow-green coloring, we run a DFS that can explore nodes of at most $O(\nu)$ volume. Hence, every DFS takes $O(\nu)$ time. There are $\tilde{O}(\beta^2\lambda_{\max})$ many forests, $2^{O(\min(2\beta,\nu)\log(2\beta+\nu))} = \tilde{O}(\nu^{O(\beta)})$ many red-blue colorings, and $2^{O(\min(2\beta,\lambda_{\max})\log(2\beta+\lambda_{\max}))} = \tilde{O}(\lambda_{\max}^{O(\beta)})$ many yellow-green colorings, which means we run overall $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ many DFS. □

6.6 Fragmenting a Cluster

In this section, we discuss how to fragment a cluster. Note that this algorithm is static. This algorithm and its analysis closely follow Lemma 6.4 of [82]. For this section, we assume that the data structure we present in Section 6.11 supports our cluster decomposition.

Theorem 6.6.1 (Fragmenting, inspired by Lemma 6.4 of [82])**.** *Let G be a graph, and C a cluster in that graph. Let $\lambda_{\max}, \lambda_{\min}, \nu, \delta, \beta$ be parameters and assume that:*

1. *The minimum cut of $G[C]$ is at least $\lambda_{\min} \geq \frac{\lambda_{\max}}{\beta}$*
2. *$\partial C \leq 6\lambda_{\max}$.*
3. *an instance of the LocalKCut data structure (Theorem 6.5.1) is running on the intra-cluster edges of G with parameters $\lambda_{\max}, \nu, \beta$.*

Then, there exists an algorithm that decomposes C into a disjoint union of clusters $C_1, \ldots, C_k$ for some $k \in \mathbb{N}$, such that:

1. *For any set $S \subseteq C$ with $\mathrm{Vol}(S) \leq \nu$ and $\partial_{G[C]}S \leq \lambda_{\max}$, there exists a partition $\mathcal{A}$ of $\{C_1, \ldots, C_k\}$ such that for each element $A \in \mathcal{A}$ (which is a subset of $\{C_1, \ldots, C_k\}$), the set $S \cap (\cup_{A' \in A} A')$ is not $(1-\delta)$-boundary sparse in $\cup_{A' \in A} A'$.*
2. *There are at most $O(\delta^{-2}\log^{O(1)}|C|)$ many clusters, and each cluster satisfies $\partial C_i \leq \partial C$.*

The algorithm runs in $\tilde{O}(\delta^{-2}\beta^2(\nu\lambda_{\max})^{O(\beta)})$ time.

Definition 6.6.2 (Intersection of a set and a set of sets)**.** *Let A be a set, $\mathcal{C}$ a set of subsets of A and $S \subseteq A$ a subset of A. Then the intersection $\mathcal{C} \cap S$ of $\mathcal{C}$ and S is the set of subsets of S: $\mathcal{C} \cap S = \{T \cap S : T \in \mathcal{C}\}$.*

Algorithm 6.6.1.

Input:

1. *a graph G with a partition $\mathcal{P}$*
2. *a cluster $C \in \mathcal{P}$*
3. *access to a LocalKCut instance on $G \setminus \{e \in E(G) : \text{label}(e) = \text{“intercluster”}\}$*
4. *A maximum cut value $\lambda_{\max}$, a maximum volume ν, an approximation ratio β, with $\lambda_{\max} \leq \nu$.*

Algorithm:

1. *Run a LocalKCut query on every vertex v in the set $C \cap \mathcal{N}(\mathcal{N}(C))$ of vertices in C incident to boundary edges of C, which outputs $\mathcal{C}_v = \{U_1, \ldots, U_\ell\}$, a collection of cuts returned for v.*
2. *Let $\mathcal{C}$ be the collection of cuts obtained by taking the union of the collection of cuts returned by LocalKCut: $\mathcal{C} = \cup_{v \in \mathcal{N}(C)} \mathcal{C}_v$.*
3. $\textsc{Trim}(C, \mathcal{C})$

$\textsc{Trim}(C', \mathcal{C}')$

1. *Remove from $\mathcal{C}'$ all cuts that are not $(1-\delta)$-boundary sparse in C' (we call this the* pruning *step).*
2. *If there is a $D \in \mathcal{C}'$ that crosses C' such that $\partial_{G[C']} D \leq 0.4\lambda_{\max}$, remove C' from $\mathcal{P}$, add $C' \setminus D$ and D to $\mathcal{P}$ and recursively call $\textsc{Trim}(C' \setminus D, \mathcal{C}' \cap (C' \setminus D))$ and $\textsc{Trim}(D, \mathcal{C}' \cap D)$.*
3. *Otherwise, if $\mathcal{C}' \neq \varnothing$, choose the set $D \in \mathcal{C}'$ with minimum $\min\{|D|, |C' \setminus D|\}$ (in other words, D is the most unbalanced – cardinality wise – cut in $\mathcal{C}'$). Remove C' from $\mathcal{P}$, add $C' \setminus D$ and D to $\mathcal{P}$ and recursively call $\textsc{Trim}(C' \setminus D, \mathcal{C}' \cap (C' \setminus D))$ and $\textsc{Trim}(D, \mathcal{C}' \cap D)$.*

6.6.1 Correctness

Proof of item 1. Consider a set $S \subseteq C$ with $\text{Vol}(S) \leq \nu$ and $\partial_{G[C]} S \leq \lambda_{\max}$.

Case 1: If $S = \varnothing$ or $S = C$, then item 1 is trivial because $\varnothing$ and C are always non-$(1-\delta)$-boundary-sparse in C, and we can set $\mathcal{A} = \{\{C_1, \ldots, C_k\}\}$.

Case 2: If $w(S, V \setminus C) = 0$, then similarly, S is trivially non-$(1-\delta)$-boundary-sparse in C, and we can set $\mathcal{A} = \{\{C_1, \ldots, C_k\}\}$.

Case 3: Thus we are left to consider the case $S \neq \varnothing$,$S \subsetneq C$, and $w(S, V \setminus C) > 0$. Since $\partial_{G[C]} S \leq \lambda_{\max}$ and $w(S, V \setminus C) > 0$, there exists an edge (u, v) with $u \in S \cap C, v \in V \setminus C$.

Querying LocalKCut on v, as Trim does in the first step, finds S, by Theorem 6.5.1. Therefore, we have that $S \in \mathcal{C}$, the collection of cuts found at the root instance $\textsc{Trim}(C, \mathcal{C})$. Moreover, the property $S \cap C' \in \mathcal{C}'$ holds recursively along the recursion tree as long as Step 1 of Trim does not apply.

We now prove item 1 for $S' = S \cap C' \in \mathcal{C}'$ by bottom-up induction on the recursion tree of the Trim subroutine, starting at the calls where $S' = S \cap C'$ is pruned from $\mathcal{C}'$ in Step 1. As base cases, we consider instances of Trim with first parameter C' where S is removed from $\mathcal{C}'$ in the initial pruning step of Trim. Since S' is removed from $\mathcal{C}'$, it is non-$(1-\delta)$-boundary sparse in C', that is, we can set $\mathcal{A} = \{\{C_i \subseteq C'\}, \{C_1, \ldots, C_k\} \setminus \{C_i \subseteq C'\}\}$.

For the induction step, we consider sets S', where S' survives the initial pruning step of Trim. Then $\mathcal{C}' \neq \varnothing$ after Step 1. This implies that C' is replaced by D and $C' \setminus D$ for some $D \subsetneq C'$. By induction on the recursive instances $\textsc{Trim}(C' \setminus D, \mathcal{C}' \cap (C' \setminus D))$ and $\textsc{Trim}(D, \mathcal{C}' \cap D)$, there exist two partitions $\mathcal{P}_D$ and $\mathcal{P}_{C' \setminus D}$ of $\{C_1, \ldots C_k\}$, the first of $\{C_1, \ldots C_k\} \cap D \subsetneq \{C_1, \ldots C_k\}$ and the latter of $\{C_1, \ldots C_k\} \cap C' \setminus D \subsetneq \{C_1, \ldots C_k\}$, where for each collection $A \in \mathcal{P}_D \cup \mathcal{P}_{C' \setminus D}$, such that $S \cap (\cup_{A' \in A} A')$ is not $(1-\delta)$-boundary sparse in $\cup_{A' \in A} A'$. Thus, we simply set $\mathcal{A} = \mathcal{P}_D \cup \mathcal{P}_{C' \setminus D}$. □

The rest of this subsection is devoted to prove item 2, focusing on the Trim subroutine. We start by showing that the boundary-size of the clusters only gets smaller as we run Trim.

Observation 6.6.3. *If* Trim *decomposes C' into D and $C' \setminus D$ in Step (2), then $\partial D \leq \partial C' - 0.04\lambda_{\min}$ and $\partial(C' \setminus D) \leq \partial C' - 0.04\lambda_{\min}$.*

Proof. Recall that $\lambda_{\max} = 1.2\lambda_{\min}$. In Step (2) we have that

$$w(D, V \setminus C') = \partial D - w(D, C' \setminus D) \geq \lambda - 0.4\lambda_{\max} \geq \lambda_{\min} - 0.4\lambda_{\max} = 0.52\lambda_{\min}$$
$$\Rightarrow \partial(C' \setminus D) = \partial C' - w(D, V \setminus C') + w(D, C' \setminus D) \leq \partial C' - 0.52\lambda_{\min} + 0.4\lambda_{\max} \leq \partial C' - 0.04\lambda_{\min}$$

and by symmetry, we also obtain $\partial D \leq \partial C' - 0.04\lambda_{\min}$. In other words, the boundary-size decreases by at least $0.04\lambda_{\min}$ on both recursive instances. □

Observation 6.6.4. *If* Trim *decomposes in Step (3) into D and $C' \setminus D$, then $\partial D \leq \partial C' - 0.4\delta\lambda_{\min}$ and $\partial(C' \setminus D) \leq \partial C' 0.4\delta\lambda_{\min}$.*

Proof. If the algorithm decomposes in Step (3), then since D is $(1-\delta)$-boundary sparse in C',

$$\partial(C' \setminus D) = \partial C' - w(D, V \setminus C') + w(D, C' \setminus D)$$
$$\leq \partial C' - \frac{1}{1-\delta} w(D, C' \setminus D) + w(D, C' \setminus D) \leq \partial C' - \delta w(D, C' \setminus D) \leq \partial C' - 0.4\delta\lambda_{\max}$$

where we used that $w(D, C' \setminus D) \geq 0.4\lambda_{\max}$ since we are not in Step (2). By symmetry, $\partial D \leq \partial C' - 0.4\delta\lambda_{\max} \leq \partial C' - 0.4\delta\lambda_{\min}$. □

The two observations together show that the boundary-size of the cluster C' Trim is called on decreases by at least $\min\{0.04\lambda_{\min}, 0.4\delta\lambda_{\min}\}$ on both recursive instances D and $C' \cap D$.

We now upper bound the number of clusters by tracking the following three parameters of a cluster C' as Trim is called on it:

1. The boundary-size $\partial C'$,
2. The maximum cut-size within C' of a cut in $\mathcal{C}'$ after the pruning step (Step 1 of $\textsc{Trim}$), defined by
$$c(C') = \max_{\substack{D \in \mathcal{C}' \\ \text{after} \\ \text{pruning}}} \partial_{G[C']} D$$
which we define as 0 if $\mathcal{C}' = \varnothing$,
3. The cluster cardinality $|C'|$.

We will give an upper bound on the number of clusters parametrized by those three parameters, by casing on the value of the maximum cut-size within C' and a double induction on the other two parameters.

We claim that the first two parameters are at most $6\lambda_{\max}$ and $\lambda_{\max}$ respectively. The first parameter is always at most $6\lambda_{\max}$ since originally $\partial C \leq 6\lambda_{\max}$ by assumption, and we have shown in the two observations above that boundary-size never increases upon recursion. The second parameter is at most $\lambda_{\max}$ originally (since it is the parameter given to LocalKCut) and it never increases upon recursion since the value $\partial_{G[X]} D$ decreases as X gets replaced by smaller (inclusion-wise) and smaller clusters upon recursion and $\mathcal{C}'$ can only lose elements as one walks down the recursion tree.

Let $f(b, c, d)$ be the maximum number of clusters in the final decomposition starting from a cluster C' with $\partial C' \leq b$, $c(C') \leq c$ and $|C'| \leq d$. Our goal is to show that

$$f(b, c, d) \leq O\left(\left(\frac{b}{\delta\lambda_{\min}}\right)^2 (\log d)^{O(b/\lambda_{\min})}\right).$$

We recursively bound $f(b, c, d)$ by stepping through each case of the $\textsc{Trim}$ algorithm. To do so, we first make two observations.

Observation 6.6.5. *If $\textsc{Trim}$ decomposes in Step (2) into D and $C' \setminus D$, then the maximum number of clusters in the final decomposition of each recursive instance is upper bounded by $f(b - 0.04\lambda_{\min}, c, d)$, i.e., we have $f(b, c, d) = 2f(b - 0.04\lambda_{\min}, c, d)$.*

Proof. This observation follows directly from Observation 6.6.3 as it shows that $\partial D, \partial(C' \setminus D) \leq \partial C' - 0.04\lambda_{\min} \leq b - 0.04\lambda_{\min}$. □

Observation 6.6.6. *If $\textsc{Trim}$ decomposes in Step (3), let $D' = D$ if $|D| = \min\{|D|, |C' \setminus D|$ and $D' = C' \setminus D$ otherwise. Then we must have $|D'| \leq |C'|/2$. Moreover, all sets $U \in \mathcal{C}'$ that cross D' must also cross $C' \setminus D'$.*

Proof. The first statement is trivial. For the second statement, recall that $D \in \mathcal{C}'$ when this step is executed. Suppose for contradiction that there exists $U \in \mathcal{C}'$ that crosses D' but not $C' \setminus D'$. Note that this implies that $U \neq D'$ and $U \neq C' \setminus D'$. Furthermore it either implies that $U \subsetneq D'$ or $C' \setminus D' \subsetneq U$. The latter is equivalent to $C' \setminus U \subsetneq D'$. This contradicts that D is chosen in Step (3) to minimize $\min\{|D|, |C' \setminus D|\}$, as U would have smaller cardinality. □

Observation 6.6.7. *If* Trim *decomposes in Step (3), we obtain the recursive bound*

$$f(b,c,d) \leq \max\{1, \quad 2f(b-0.04\lambda_{\min},c,d)+f(b,c,d/2), \quad f(b,0.6c,d)+f(b-0.4\delta\lambda_{\min},c,d)\}$$

Proof. Suppose that Trim decomposes in Step (3) into D' and $C' \setminus D'$, with D' defined as in Observation 6.6.6. By that observation, we have that $|D'| \leq |C'|/2$ and moreover, all sets $U \in \mathcal{C}'$ that cross D' must also cross $C' \setminus D'$. We case on whether $c(D') \geq 0.6c(C')$, i.e., whether there exists $U \in \mathcal{C}'$ that crosses D' with $\partial_{G[D']}U \geq 0.6c(C')$.

1. If such a set U exists, then from $\partial_{G[D']}U+\partial_{G[C'\setminus D']}U \leq \partial_{G[C']}U$, we obtain $\partial_{G[C'\setminus D']}U \leq \partial_{G[C']}U - \partial_{G[D']}U \leq c(C') - 0.6c(C') \leq 0.4c(C')$. Since U also crosses $C' \setminus D'$, it follows that the recursive instance $C' \setminus D'$ must go through Step (2). Let U_1, U_2 be the decomposition of $C' \setminus D'$. Then by Observation 6.6.3, $\partial U_1 \leq \partial(C' \setminus D') - 0.04\lambda_{\min} \leq \partial C' - 0.04\lambda_{\min}$ and $\partial U_2 \leq \partial(C' \setminus D') - 0.04\lambda_{\min} \leq \partial C' - 0.04\lambda_{\min}$. Also by Observation 6.6.6 $|D'| \leq |C'|/2 \leq d/2$. By induction on clusters D', U_1, U_2, the total number of clusters at the end is at most

$$2f(b-0.04\lambda_{\min},c,b) + f(b,c,d/2)$$

2. Otherwise, we have $c(D') < 0.6c(C') \leq c/2$. By Observation 6.6.4, we have that $\partial(C' \setminus D') \leq \partial C' - 0.4\delta\lambda_{\min}$. By induction on D' and $C' \setminus D'$, the total number of clusters at the end is at most

$$f(b,0.6c,d) + f(b-0.4\delta\lambda_{\min},c,d)$$

Overall, if the algorithm decomposes by Step (3), we obtain the recursive bound

$$f(b,c,d) \leq \max\{1, \quad 2f(b-0.04\lambda_{\min},c,d)+f(b,c,d/2), \quad f(b,0.6c,d)+f(b-0.4\delta\lambda_{\min},c,d)\}$$

□

Next, we state the base cases for our induction that bounds $f(b,c,d)$.

1. Since we always have $\partial C' \geq \lambda \geq \lambda_{\min}$, we have the vacuous bound $f(b,c,d) = 0$ for $b < \lambda_{\min}$.
2. If $\mathcal{C}' = \varnothing$, no further recursion is necessary. Thus, $f(b,0,d) = 1$.
3. If $|C'| = 1$, the cluster cannot be cut any further. Thus, $f(b,c,1) = 1$.

We now bound $f(b,c,d)$ using a case analysis depending on the value of the second parameter. As shown by Observation 6.6.5 and Observation 6.6.7, the value of the second parameter never increases. Furthermore, if it drops to at most $0.4\lambda_{\max}$ on a set C', then the algorithm always executes Step (2) on subsets of C'. Thus, we can bound this setting first.

Case 1. If $c(C') \leq 0.4\lambda_{\max}$, then TRIM either takes Step (2) or does nothing (in the case $\mathcal{C}' = \varnothing$), so

$$f(b, c, d) \leq \max\{1, 2f(b - 0.04\lambda_{\min}, c, d)\} \quad \text{for} \quad c \leq 0.4\lambda_{\max}$$

Every recursive call with first parameter D still has $c(D) \leq 0.4\lambda_{\min}$, so all recursive calls made in the subtree of the call with first parameter C' is made in Step (2). We bound $f(b, c, d)$ by bounding the number of recursive calls of TRIM as each call increases the number of clusters in $\mathcal{P}$ by 1. The number of recursive calls in Step (2) doubles for at most $\left\lceil \frac{b-\lambda_{\min}}{0.04\lambda_{\min}} \right\rceil \leq \frac{b}{0.04\lambda_{\min}} \leq 25\frac{b}{\lambda_{\min}}$ levels of recursion and $2^x \geq 1$ for all positive x, so it holds that $f(b, c, d) \leq 2^{25b/\lambda_{\min}}$.

Case 2. For $c \in (0.4\lambda_{\max}, 0.61\lambda_{\max}]$, TRIM either takes Step (2) or Step (3), or does nothing. Note that by Observation 6.6.5 and Observation 6.6.7, in both cases we have

$$f(b, c, d) \leq \max\{1, \quad 2f(b-0.04\lambda_{\min}, c, d)+f(b, c, d/2), \quad f(b, 0.6c, d)+f(b-0.4\delta\lambda_{\min}, c, d)\}$$

We show the following bound by double induction on b and d.

$$f(b, c, d) \leq \frac{b}{0.4\delta\lambda_{\min}}(2 + 2\log d)^{25b/\lambda_{\min}}. \tag{6.1}$$

The base case is trivially true if either $b < \lambda_{\min}$ or $d = 1$. For the induction step, note that $0.6c \leq 0.6 \cdot 0.61\lambda_{\max} \leq 0.4\lambda_{\max}$, so that a call on a set with second parameter $0.6c$ guarantees that Step (2) is executed on this call and we can apply the bound from Case 1. Now by Observation 6.6.7 it holds that:

$$\begin{aligned} f(b, c, d) \leq \max\{1 & \\ & 2f(b - 0.04\lambda_{\min}, c, d) + f(b, c, d/2), && (6.2)\\ & f(b, 0.6c, d) + f(b - 0.4\delta\lambda_{\min}, c, d)\} && (6.3) \end{aligned}$$

We apply the induction hypothesis (Equation (6.1)) to both terms of Equation (6.2) and the second term of Equation (6.3), while we apply the bound of Case 1 to the first term of Equation (6.3):

$$\begin{aligned} f(b, c, d) \leq \max\{1, & \\ & 2 \cdot \frac{b}{0.4\delta\lambda_{\min}}(2 + 2\log d)^{25(b-0.04\lambda_{\min})/\lambda_{\min}} + \frac{b}{0.4\delta\lambda_{\min}}(2 + 2\log\frac{d}{2})^{25b/\lambda_{\min}}, && (6.4)\\ & 2^{25b/\lambda_{\min}} + \frac{b - 0.4\delta\lambda_{\min}}{0.4\delta\lambda_{\min}}(2 + 2\log d)^{25b/\lambda_{\min}}\} && (6.5) \end{aligned}$$

We develop the exponent in the first term of Equation (6.4), and develop the logarithm in its second term. We upper bound 2 by $2 + 2\log d$ in Equation (6.5):

$$\begin{aligned}
&\leq \max\{1, \\
&\qquad 2 \cdot \frac{b}{0.4\delta\lambda_{\min}}(2+2\log d)^{25b/\lambda_{\min}-1} + \frac{b}{0.4\delta\lambda_{\min}}(2+2\log d)^{25b/\lambda_{\min}-1} \cdot (2+2\log d-2), \\
&\qquad (2+2\log d)2^{25b/\lambda_{\min}} + \frac{b-0.4\delta\lambda_{\min}}{0.4\delta\lambda_{\min}}(2+2\log d)^{25b/\lambda_{\min}}\} \\
&= \qquad \frac{b}{0.4\delta\lambda_{\min}}(2+2\log d)^{25b/\lambda_{\min}}.
\end{aligned}$$

where the last equality comes from the fact that $\frac{b}{0.4\delta\lambda_{\min}}(2+2\log d)^{25b/\lambda_{\min}} \geq \frac{b}{\delta\lambda_{\min}}2^{25b/\lambda_{\min}} \geq \frac{1}{\delta}2^{25} \geq 1$.

Case 3. For $c \in [0.61\lambda_{\max}, \lambda_{\max}]$, Trim either takes Step (2) or Step (3), or does nothing. Note that by Observation 6.6.5 and Observation 6.6.7, in both cases we have

$$f(b,c,d) \leq \max\{1, \quad 2f(b-0.04\lambda_{\min},c,d)+f(b,c,d/2), \quad f(b,0.6c,d)+f(b-0.4\delta\lambda_{\min},c,d)\}$$

We show the bound

$$f(b,c,d) \leq \left(\frac{b}{0.4\delta\lambda_{\min}}\right)^2 (2+2\log d)^{25b/\lambda_{\min}} \tag{6.6}$$

by induction on b and d.

The base case is trivially true if either $b < \lambda_{\min}$ or $d = 1$. For the induction step, note that $0.6c \leq 0.6\lambda_{\max}$, which implies that a call on a set with second parameter $0.6c$ guarantees that the bound from Case 2 can be applied to this call. Now by induction on $b \geq \lambda_{\min}$ and $d > 1$, it holds that

$$\begin{aligned}
f(b,c,d) \leq \max\{1& \\
&2f(b-0.04\lambda_{\min},c,d) + f(b,c,d/2), && (6.7)\\
&f(b,0.6c,d) + f(b-0.4\delta\lambda_{\min},c,d)\} && (6.8)
\end{aligned}$$

We apply the induction hypothesis (Equation (6.6)) to both terms of Equation (6.7) and the second term of Equation (6.8), while we apply the bound of Case 2 to the first term of Equation (6.8):

$$\begin{aligned}
&\leq \max\{1, \\
&\qquad 2 \cdot \left(\frac{b}{0.4\delta\lambda_{\min}}\right)^2 (2+2\log d)^{25(b-0.04\lambda_{\min})/\lambda_{\min}} + \left(\frac{b}{0.4\delta\lambda_{\min}}\right)^2 (2+2\log\frac{d}{2})^{25b/\lambda_{\min}}, && (6.9)\\
&\qquad \frac{b}{0.4\delta\lambda_{\min}}(2+2\log d)^{25b/\lambda_{\min}} + \left(\frac{b-0.4\delta\lambda_{\min}}{0.4\delta\lambda_{\min}}\right)^2 (2+2\log d)^{25b/\lambda_{\min}}\} && (6.10)
\end{aligned}$$

We develop the exponent in the first term of Equation (6.9), and develop the logarithm in its second term. We upper bound $b - 0.4\delta\lambda_{\min}$ by b in Equation (6.10):

$$\begin{aligned}
&\leq \max\{1, \\
&\qquad 2 \cdot \left(\frac{b}{0.4\delta\lambda_{\min}}\right)^2 (2 + 2\log d)^{25b/\lambda_{\min}-1} \\
&\qquad + \quad \left(\frac{b}{0.4\delta\lambda_{\min}}\right)^2 (2 + 2\log d)^{25b/\lambda_{\min}-1}(2 + 2\log d - 2), \\
&\qquad \frac{b}{0.4\delta\lambda_{\min}}(2 + 2\log d)^{25b/\lambda_{\min}} + \left(\frac{b - 0.4\delta\lambda_{\min}}{0.4\delta\lambda_{\min}}\right)\left(\frac{b\lambda_{\min}}{0.4\delta\lambda_{\min}}\right)(2 + 2\log d)^{25b/\lambda_{\min}}\} \\
&= \quad \left(\frac{b}{0.4\delta\lambda_{\min}}\right)^2 (2 + 2\log d)^{25b/\lambda_{\min}}.
\end{aligned}$$

Since for the set C on which the algorithm was run we know that $b \leq 6\lambda_{\max}$, $c \leq \lambda_{\max}$, $d \leq |C|$ and $\lambda_{\max} \leq 1.2\lambda_{\min}$, we obtain the desired bound $O(\delta^{-2}\log^{O(1)}|C|)$ on the number of clusters, which concludes the proof of item 2.

6.6.2 Running Time analysis

Lemma 6.6.8. *The running time of Algorithm 6.6.1 is* $\tilde{O}(\delta^{-2}\beta^2(\nu\lambda_{\max})^{O(\beta)})$.

Proof. Step (1) of Algorithm 6.6.1 makes $O(\lambda_{\max})$ many requests to LocalKCut. By Theorem 6.5.1, this takes $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ time. In particular, we know that we have at most $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ many cuts in $\mathcal{C}$ in Step (2) of Algorithm 6.6.1, each of volume ν.

It remains to bound the time for Step (3) of Algorithm 6.6.1. Since we proved in the correctness proof that the algorithm added at most $O(\delta^{-2}\log^{O(1)}|C|)$ many clusters to $\mathcal{P}$ at termination, the recursion tree is binary and each recursive call increases the number of clusters in $\mathcal{P}$ by 1, we know that Trim is run at most $O(\delta^{-2}\log^{O(1)}|C|)$ times. It thus suffices to show that a run of Trim takes $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ time.

In Step (1), for every cut in $\mathcal{C}'$, it takes $O(\nu)$ time to check whether a cut is boundary sparse by Theorem 6.11.1, and thus $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ to check all cuts. Similarly, in Step (2), it takes $O(\nu)$ time for each cut to decide if a cut satisfies the if condition, $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ overall. Removing C' and adding D and $C' \setminus D$ takes $O(\nu)$ time, as this is encoded by simply changing the labels of the boundary edges to inter-cluster. Creating $\mathcal{C}' \cap D$ and $\mathcal{C}' \cap (C' \setminus D)$ takes $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ time, as there are $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ cuts of volume ν each to intersect with another set.

Similarly, in Step (3), it takes $O(\nu)$ time for each cut to find its cardinality and the one of its complement, thus $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ overall to find the minimum. Removing C' and adding D and $C' \setminus D$ takes $O(\nu)$ time, as this is encoded by simply changing the labels of the boundary edges to inter-cluster. Creating $\mathcal{C}' \cap D$ and $\mathcal{C}' \cap (C' \setminus D)$ takes $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ time, as there are $\tilde{O}(\beta^2(\nu\lambda_{\max})^{O(\beta)})$ cuts of volume ν each to intersect with another set. □

6.7 The Cluster Decomposition and the Cluster Hierarchy

In this section, we lay out the main definitions and data structures that we maintain throughout our algorithm. The central object we maintain is the cluster hierarchy, a hierarchy of cluster decompositions.

Choice of parameter values. We remind the reader that we have $c > 0$, $\delta = 2^{O(\log^{3/4-c} n)} \leq 0.04$. We set $\lambda_{\min} = 2^{\Theta(\log^{3/4-c} n)}$ and $\lambda_{\max} \leq 1.2\lambda_{\min}$, $H = 38^h = 2^{-O(\log^{1/2} n)}$, $\phi = \frac{\phi'}{38^h} = 2^{-\Theta(\log^{3/4} n)}$, $\rho = 2^{\Theta(\log^{1/2} n)}$ and $\alpha = \frac{1}{\text{poly}\log n}$.

6.7.1 Cluster Decomposition

Definition 6.7.1 (Pre-Cluster decomposition)**.** *Let $\alpha, \phi, \delta, H \in \mathcal{R}$, $\lambda_{\max}, \in \mathbb{N}$. An* $(\alpha, \phi, \lambda_{\max}, H, \delta)$- pre-cluster decomposition *of a graph G is a partition $\{C_j\}_{j\in[\ell]}$ of the vertex set of G, for some $\ell \in \mathbb{N}_+$. Moreover, that partition satisfies, for each $j \in [\ell]$:*

i. C_j is connected.

ii. C_j is contained in an $(\alpha, \phi' = \phi H)$-boundary linked expander.

iii. C_j satisfies either:

- *a) it contains no $(1-\delta)$-boundary sparse cut S with $\partial_G(S) \leq \lambda_{\max}$ and $\text{Vol}(S) \leq \frac{\lambda_{\max}}{\phi}$, or*
- *b) it satisfies $\partial C_j \leq 6\lambda_{\max}$.*

Definition 6.7.2 (Cluster decomposition)**.** *Let $\alpha, \phi, \delta \in \mathcal{R}$, $\lambda_{\max}, \lambda_{\min} \in \mathbb{N}$. An* $(\alpha, \phi, \lambda_{\max}, H, \delta)$-cluster decomposition *of a graph G is a partition $\{C_j\}_{j\in[\ell]}$ of the vertex set of G, for some $\ell \in \mathbb{N}_+$, obtained from an $(\alpha, \phi, \lambda_{\max}, H, \delta)$-pre-cluster decomposition, where each cluster C_j that does not satisfy Condition ?a is fragmented using Algorithm 6.6.1 (see Theorem 6.6.1).*

As α, ϕ, $\lambda_{\max}$, H and δ do not change, we simply use *cluster decomposition* to denote an $(\alpha, \phi, \lambda_{\max}, H, \delta)$-cluster decomposition in the following.

The main point of the cluster decomposition is that it can approximate most minimum proper cuts, that is, in most cases, if S is a minimum proper cut in G of cut-size λ with $\lambda_{\min} \leq \lambda \leq \lambda_{\max}$, then we can uncross S by a cut S' that consists of the union of some clusters, and that is an approximate minimum proper cut.

Let us now talk about the corner cases. In fact, given a cluster decomposition we can always uncross a cut S into a set S'. However, S' is not guaranteed to be a proper cut, that is, we could have $\partial S' = 0$. In this case, we show S is a *local cut* in a *mirror cluster*, defined next.

Definition 6.7.3 (Local cut)**.** *A* local cut *is a cut of volume at most $4\frac{\lambda_{\max}}{\phi}$.*

Local cuts are central in our analysis, as they are easier to find and maintain than non-local cuts. To deal with the local cuts, we introduce *mirror clusters*, one for each cluster.

Definition 6.7.4 (Mirror clusters, graphs and cuts)**.** *The* mirror cluster C' *of a cluster* C *in graph* G *is the graph* $G/(V \setminus C)$*, that is, the graph where all nodes outside of* C *have been contracted into one node.*

A mirror graph $G_{\mathcal{P}_1}$ *of* G *according to partition* $\mathcal{P}_1$ *is the graph that is the collection of all mirror clusters of the clusters in* $\mathcal{P}_1$*.*

A mirror cut S_v *of a vertex* $v \in V$ *of a graph* G*, given a partition* $\mathcal{P}_1$*, a volume* ν *and a maximum cut value* $\lambda_{\max}$*, is defined as follows. Let* $C_v \in \mathcal{P}_1$ *be the cluster of* G *such that* $v \in C_v$*. Then* S_v *is a proper cut of smallest cut-value among proper local cuts* $S \subsetneq C_v$ *of cut-value at most* $\lambda_{\max}$ *and that contain* v*. Note that many cuts can be mirror cuts of* v*, and that such a cut may also not exist.*

Mirror clusters have the nice property that all local cuts can be found by running LocalKCut on them. We can thus state the main property of a cluster decomposition:

Proposition 6.7.5 (Cluster Decomposition Uncrossing)**.** *Let* $\{C_j\}_{j\in[\ell]}$ *be an* $(\alpha, \phi, \lambda_{\max}, H, \delta)$*-cluster decomposition of a graph* G*. If* $\lambda_{\min} \leq \lambda \leq \lambda_{\max}$*, then every minimum proper cut* S *of value* λ *can be* $(1+2\delta)$*-approximated by a cut* S'*, that can be one of two cases:*

1. S' *is a cut of* G *and* S' *crosses no cluster.*
2. *There exists a cluster* C_j *such that* S' *is a local cut in the mirror cluster of* C_j*.*

We prove this proposition in the rest of this section.

We let $\{C_j\}_{j\in[\ell]}$ be a cluster decomposition as defined in Proposition 6.7.5, and first show that any minimum cut can cross at most two clusters using the following lemmata.

Lemma 6.7.6. *Let* $\{C_j\}_{j\in[\ell]}$ *be an* $(\alpha, \phi, \lambda_{\max}, H, \delta)$*-cluster decomposition of a graph* G *with* $\alpha > \phi$*. Let* S *be a cut of* G*, and let* C_j *be a cluster that* S *crosses. Let* X *equal* $S \cap C_j$ *if* $\mathrm{Vol}(S \cap C_j) \leq \frac{\mathrm{Vol}(C_j)}{2}$*, and let* X *equal* $C_j \setminus S$ *otherwise. We have that* $\mathrm{Vol}(X) \leq \frac{\partial S}{\phi}$*.*

In particular, if S *is a minimum proper cut of cut-size at most* $\lambda_{\max}$*, we have that* $\mathrm{Vol}(X) \leq \frac{\lambda_{\max}}{\phi}$*.*

Proof. As C_j is a cluster in the expander decomposition, C_j is contained (or equal to) a cluster in the pre-cluster decomposition, which itself is contained in an (α, ϕ')-boundary linked expander as per Condition ? of Definition 6.7.1. Let C be this expander. All volumes in this proof are the volumes of the sets in $G[C]^{\frac{\alpha}{\phi'}}$, which are an upper bound of the volumes in G. We use $G[C]^{\frac{\alpha}{\phi'}}$ as this is the graph on which we have the guarantee of being a ϕ-expander as per Theorem 6.3.5.

Let $X' = S \cap C$ if $\mathrm{Vol}(S \cap C) \leq \frac{\mathrm{Vol}(C)}{2}$, and $X' = C \setminus S$ otherwise. Since $G[C]^{\frac{\alpha}{\phi'}}$ is a ϕ-expander, we have that $\mathrm{Vol}(X') \leq \frac{w(X, C\setminus X)}{\phi} \leq \frac{\partial S}{\phi}$. Therefore, $\mathrm{Vol}(X' \cap C_j) \leq \frac{\partial S}{\phi}$. Recall

that $\mathrm{Vol}(X) = \min\{\mathrm{Vol}(S \cap C_j), \mathrm{Vol}(C_j \setminus S)\}$. If $X' = S \cap C$ then $\mathrm{Vol}(X) \leq \mathrm{Vol}(S \cap C_j) \leq \mathrm{Vol}(S \cap C) = \mathrm{Vol}(X') \leq \frac{\partial S}{\phi}$. If $X' = C \setminus S$ then $\mathrm{Vol}(X) \leq \mathrm{Vol}(C_j \setminus S) \leq \mathrm{Vol}(C \setminus S) = \mathrm{Vol}(X') \leq \frac{\partial S}{\phi}$. Thus, the result follows. □

For the rest of this paper, we will thus simply call our (α, ϕ')-boundary linked expanders as ϕ-expanders.

Corollary 6.7.7. *Let $\mathcal{P}_1$ be a pre-cluster decomposition, and let C_j be a cluster in that decomposition. Let $S \subseteq C_j$ be a proper cut of G of cut-size Λ. Then there exists in the mirror cluster of C_j a cut of cut-size ∂S, of volume at most $3\frac{\Lambda}{\phi}$ (in G).*

Proof. By Lemma 6.7.6, we have that either S or $C_j \setminus S$ has volume at most $\frac{\Lambda}{\phi}$. In the case where $\mathrm{Vol}(S) \leq \frac{\Lambda}{\phi}$, the claim is trivial. In the case where $\mathrm{Vol}(C_j \setminus S) \leq \frac{\Lambda}{\phi}$, note that in the mirror cluster, we have that $X = (C_j \setminus S) \cup \{v\}$ has boundary-size equal to $\partial S = \Lambda$, where v is the vertex representing the contracted $G \setminus C_j$. It remains to show that $\mathrm{Vol}(X) \leq 3\frac{\Lambda}{\phi}$. Note that $\mathrm{Vol}(v) \leq \mathrm{Vol}(C_j \setminus S) + \partial S \leq 2\frac{\Lambda}{\phi}$, and thus $\mathrm{Vol}(X) \leq \mathrm{Vol}(v) + \mathrm{Vol}(C_j \setminus S) \leq 3\frac{\Lambda}{\phi}$. □

Lemma 6.7.8. *Let $\mathcal{P}_1$ be an $(\alpha, \phi, \lambda_{\max}, H, \delta)$-pre cluster decomposition, and $\mathcal{P}_2$ be its associated cluster decomposition. Let $f : \mathcal{P}_2 \mapsto \mathcal{P}_1$ be the mapping that maps each cluster $C \in \mathcal{P}_2$ to the cluster it is contained in in $\mathcal{P}_1$. Let S be a minimum proper cut S of cut-size at most $\lambda_{\max}$. Let $C'_1, \ldots, C'_k \in \mathcal{P}_2$ be the clusters that S crosses. Then $|\{f(C'_1), \ldots, f(C'_k)\}| \leq 2$.*

Proof. Recall that $\delta \leq \frac{1}{3}$. Assume by contradiction that $|\{f(C'_1), \ldots, f(C'_k)\}| \geq 3$, and wlog assume the clusters C_1, C_2 and C_3 belong to $\{f(C'_1), \ldots, f(C'_k)\}$. Since S is a minimum proper cut, we know that $w(S, V \setminus S) \leq \lambda_{\max}$. As the clusters are vertex disjoint, $w(S \cap C_1, C_1 \setminus S) + w(S \cap C_2, C_2 \setminus S) + w(S \cap C_3, C_3 \setminus S) \leq w(S, V \setminus S)$ and thus there exists a C_i such that $w(S \cap C_i, C_i \setminus S) \leq \frac{w(S, V \setminus S)}{3}$.

We have two cases. Either C_i satisfies Condition ?a of Definition 6.7.1, which means C_i contains no $(1-\delta)$-boundary sparse cut of volume at most $\frac{\lambda_{\max}}{\phi}$ and of cut-size at most $\lambda_{\max}$, or it satisfies Condition ?b.

1. In the first case, since C_i contains no $(1-\delta)$-boundary-sparse cuts of cut-size at most $\lambda_{\max}$ and of volume at most $\frac{\lambda_{\max}}{\phi}$, and as by Lemma 6.7.6, $S \cap C_i$ has volume at most $\frac{\lambda_{\max}}{\phi}$, we have that $S \cap C_i$ is not $(1-\delta)$-boundary-sparse:

   $$\frac{w(S, V \setminus S)}{3} \geq w(S \cap C_i, C_i \setminus S) \geq (1-\delta) \min\{w(S \cap C_i, V \setminus C_i), w(V \setminus (S \cap C_i), V \setminus C_i)\}$$

   And hence either $w(S \cap C_i, V \setminus C_i) \leq \frac{w(S, V \setminus S)}{3(1-\delta)} \leq \frac{w(S, V \setminus S)}{2}$ and $S \cap C_i$ is a cut of cut-size at most $w(S \cap C_i, C_i \setminus S) + w(S \cap C_i, V \setminus C_i) \leq \frac{w(S, V \setminus S)}{3} + \frac{w(S, V \setminus S)}{2}$, a contradiction to S being a minimum proper cut (remember that C_i is connected by definition of a pre-cluster decomposition and thus $S \cap C_i$ is a proper cut), or $w(C_i \setminus S, V \setminus C_i) \leq \frac{w(S, V \setminus S)}{3(1-\delta)} \leq \frac{w(S, V \setminus S)}{2}$ and $C_i \setminus S$ is a cut of cut-size at most $\frac{w(S, V \setminus S)}{3} + \frac{w(S, V \setminus S)}{2}$, also a contradiction to S being a minimum proper cut.

2. If C_i satisfies Condition ?b, then let $\{A_1, \ldots, A_\kappa\} \subsetneq \mathcal{P}_2$ be the partition of C_i obtained by fragmenting C_i, and let $A_j \in \{C'_1, \ldots, C'_k\} \cap \{A_1, \ldots, A_\kappa\}$ be one of the fragmented clusters that S crosses. Then $S \cap A_j$ is a proper cut. By Theorem 6.6.1, there is a subset of indices $Q \subseteq [\kappa]$ such that $A = \cup_{q \in Q} A_q$ satisfies that $S \cap A$ is not $(1-\delta)$-boundary sparse in A. Similarly to the first case, we then have:

$$\frac{w(S, V \setminus S)}{3} \geq w(S \cap C_i, C_i \setminus S) \geq w(S \cap A, A \setminus S) \geq (1-\delta) \min\{w(S \cap A, V \setminus A), w(V \setminus (S \cap A), V \setminus A)\}$$

And hence either $w(S \cap A, V \setminus A) \leq \frac{w(S,V\setminus S)}{3(1-\delta)} \leq \frac{w(S,V\setminus S)}{2}$ and $S \cap A$ is a cut of cut-size at most $w(S \cap A, A \setminus S) + w(S \cap A, V \setminus A) \leq \frac{w(S,V\setminus S)}{3} + \frac{w(S,V\setminus S)}{2}$, a contradiction to S being a minimum proper cut ($S \cap A$ must be a proper cut of A as otherwise it cannot be non $(1-\delta)$-boundary-sparse in A), or $w(A \setminus S, V \setminus A) \leq \frac{w(S,V\setminus S)}{3(1-\delta)} \leq \frac{w(S,V\setminus S)}{2}$ and $A \setminus S$ is a cut of cut-size at most $\frac{w(S,V\setminus S)}{3} + \frac{w(S,V\setminus S)}{2}$, also a contradiction to S being a minimum proper cut.

□

Corollary 6.7.9. *Let $\mathcal{P}_1$ be an $(\alpha, \phi, \lambda_{\max}, H, \delta)$-pre-cluster decomposition, and $\mathcal{P}_2$ be its associated cluster decomposition. Let $f : \mathcal{P}_2 \mapsto \mathcal{P}_1$ be the mapping that maps each cluster $C \in \mathcal{P}_2$ to the cluster that contains it in $\mathcal{P}_1$. Let S be a minimum proper cut S of cut-size at most $\lambda_{\max}$. Let $C'_1, \ldots, C'_k \in \mathcal{P}_2$ be the clusters S crosses, if any. We have 3 cases:*

(1) $|\{f(C'_1), \ldots, f(C'_k)\}| = 0$.

(2) $|\{f(C'_1), \ldots, f(C'_k)\}| = 1$

(3) $|\{f(C'_1), \ldots, f(C'_k)\}| = 2$.

If we are in Case (1) of Corollary 6.7.9, then we trivially are in Case 1. of Proposition 6.7.5 by simply setting $S' = S$, as $\{f(C'_1), \ldots, f(C'_k)\}$ being empty can only mean that S crosses no cluster. If on the other hand we are in either case (2) or (3), we need to uncross S to get S'. Next we show that in those two cases there exists a cut S' of cut cut-size at most $(1+2\delta)\lambda$ that crosses no cluster. The proofs exploit the definition of $(1-\delta)$-boundary sparseness and Lemma 6.7.6.

Lemma 6.7.10. *If we are in Case (2) of Corollary 6.7.9, then there exists a cut S' that is a $(1+2\delta)$-approximation of S, and satisfies either:*

i. $\mathrm{Vol}_{G/(V\setminus C_1)}(S') \leq 4\lambda_{\max}$, *where C_1 is a cluster of $\mathcal{P}_1$, and S' is a cut of $G/(V \setminus C_1)$, or*

ii. $S' \subsetneq V$ *crosses no cluster and S' is a cut of G.*

Proof. Wlog, assume that S is connected. Also, wlog, let C_1 be the unique element of $\{f(C'_1), \ldots, f(C'_k)\}$. We have two cases. Either C_1 satisfies Condition ?a, which means C_1 contains no $(1-\delta)$-boundary sparse cut of volume at most $\frac{\lambda_{\max}}{\phi}$ and of cut-size at most $\lambda_{\max}$, or it satisfies Condition ?b.

1. Since C_1 contains no $(1-\delta)$-boundary sparse cut of volume at most $\frac{\lambda_{\max}}{\phi}$ and of cut-size at most $\lambda_{\max}$, and $S \cap C_1$ has volume at most $\frac{\lambda_{\max}}{\phi}$ by Lemma 6.7.6, we have that $w(S \cap C_1, C_1 \setminus S) \geq (1-\delta)\min\{w(C_1 \cap S, V \setminus C_1), w(C_1 \setminus S, V \setminus C_1)\}$. Let X be the set among $C_1 \cap S$, $C_1 \setminus S$ that satisfies $w(X, V \setminus C_1) = \min\{w(C_1 \cap S, V \setminus C_1), w(C_1 \setminus S, V \setminus C_1)\}$. Note that $w(X, V \setminus C_1) \leq w(S \cap C_1, C_1 \setminus S)/(1-\delta) = w(X, C_1 \setminus X)/(1-\delta)$, which implies that $w(X, V \setminus C_1) - w(X, C_1 \setminus X) \leq \delta \cdot w(X, V \setminus C_1) \leq (\delta/(1-\delta))w(S \cap C_1, C_1 \setminus S) \leq (3\delta/2) \cdot w(S \cap C_1, C_1 \setminus S)$.

   Let's now temporarily set $S' = S \cup X$ if $X \not\subset S$, or $S' = S \setminus X$ otherwise, i.e., we "uncross" S. Then we have that $w(S', V \setminus S') \leq w(S, V \setminus S) - w(X, C_1 \setminus X) + w(X, V \setminus C_1) \leq w(S, V \setminus S) + 3\delta/2 \cdot w(S \cap C_1, C_1 \setminus S) \leq (1 + 3\delta/2)w(S, V \setminus S)$.

   If S' is a proper cut ($\partial S' \neq 0$), we are in Case ?

   If $\partial S' = 0$, let K be the connected component of G that contains S. Then either $S = X$ (and $S' = \emptyset$) or $S = K \setminus X$ (and $S' = K$). We conclude by Corollary 6.7.7 that we are in Case ?.

2. If C_1 satisfies Condition ?b, then let $\{A_1, \ldots, A_\kappa\} \subsetneq \mathcal{P}_2$ be the partition of C_i obtained by fragmenting C_i, and let $A_j \in \{C'_1, \ldots, C'_k\} \cap \{A_1, \ldots, A_\kappa\}$ be one of the fragmented clusters that S crosses. Then $S \cap A_j$ is a proper cut. By Theorem 6.6.1, there is a subset of indices $Q \subseteq [\kappa]$ such that $A = \cup_{q \in Q} A_q$ satisfies that $S \cap A$ is not $(1-\delta)$-boundary sparse in A.

   Similarly to the first case, we have that $w(S \cap A, A \setminus S) \geq (1-\delta)\min\{w(A \cap S, V \setminus A), w(A \setminus S, V \setminus A)\}$. Let X be the set among $A \cap S$, $A \setminus S$ that satisfies $w(X, V \setminus A) = \min\{w(A \cap S, V \setminus A), w(A \setminus S, V \setminus A)\}$. Note that $w(X, V \setminus A) \leq w(S \cap A, A \setminus S)/(1-\delta) = w(X, A \setminus X)/(1-\delta)$, which implies that $w(X, V \setminus A) - w(X, A \setminus X) \leq \delta \cdot w(X, V \setminus A) \leq (\delta/(1-\delta))w(S \cap A, A \setminus S) \leq (3\delta/2) \cdot w(S \cap A, A \setminus S)$.

   Let's now temporarily set $S' = S \cup X$ if $X \not\subset S$, or $S' = S \setminus X$ otherwise, i.e., we "uncross" S. Then we have that $w(S', V \setminus S') \leq w(S, V \setminus S) - w(X, A \setminus X) + w(X, V \setminus A) \leq w(S, V \setminus S) + 3\delta/2 \cdot w(S \cap A, A \setminus S) \leq (1 + 3\delta/2)w(S, V \setminus S)$.

   If S' is a proper cut ($\partial S' \neq 0$), we are in Case ?

   If $\partial S' = 0$, let K be the connected component of G that contains S. Then either $S = X$ (and $S' = \emptyset$) or $S = K \setminus X$ (and $S' = K$).

   We conclude by Corollary 6.7.7 that we are in Case ?.

□

Lemma 6.7.11. *If we are in Case 3 of Corollary 6.7.9, then there exists a cut S' of either $G/(V \setminus C_1)$ or $G/(V \setminus C_2)$ ($\{C_1, C_2\} = \{f(C'_1), \ldots, f(C'_k)\}$) that is a $(1 + \frac{3}{4}\delta)$-approximation of S, and satisfies $\mathrm{Vol}_{G/(V \setminus C_i)}(S') \leq 4\lambda_{\max}$ (where $j \in \{1, 2\}$ depending on where S' exists).*

Proof. Let $\lambda = w(S, V \setminus S)$. We have that $w(C_1 \cap S, C_1 \setminus S) + w(C_2 \cap S, C_2 \setminus S) \leq w(S, V \setminus S) = \lambda$. Assume thus wlog that $w(C_1 \cap S, C_1 \setminus S) \leq \frac{\lambda}{2}$.

Again, we have two cases:

1. If C_1 satisfies Condition ?a, it contains no $(1-\delta)$-boundary sparse cuts of volume at most $\frac{\lambda_{\max}}{\phi}$ of cut-size at most $\lambda_{\max}$, and thus we have that $S \cap C_1$ is not $(1-\delta)$-boundary sparse in C_1. Thus $w(S \cap C_1, C_1 \setminus S) \geq (1-\delta)\min\{w(C_1 \cap S, V \setminus C_1), w(C_1 \setminus S, V \setminus C_1)\}$. Let X be the set among $C_1 \cap S$, $C_1 \setminus S$ that satisfies $w(X, V \setminus C_1) = \min\{w(C_1 \cap S, V \setminus C_1), w(C_1 \setminus S, V \setminus C_1)\}$. It follows that $w(X, C_1 \setminus X) = w(S \cap C_1, C_1 \setminus S) \geq (1-\delta)w(X, V \setminus C_1)$. We thus have that $w(X, V \setminus X) = w(X, V \setminus C_1) + w(X, C_1 \setminus X) \leq ((1/(1-\delta)+1)w(X, C_1 \setminus X) \leq (\frac{2-\delta}{2(1-\delta)})\lambda = (1 + \frac{\delta}{2-2\delta})\lambda \leq (1 + 3\delta/4)\lambda$.

   We conclude by Corollary 6.7.7.

2. If C_1 satisfies Condition ?b, then let $\{A_1, \ldots, A_\kappa\} \subsetneq \mathcal{P}_2$ be the partition of C_1 obtained by fragmenting C_1, and let $A_j \in \{C'_1, \ldots, C'_k\} \cap \{A_1, \ldots, A_\kappa\}$ be one of the fragmented clusters that S crosses. Then $S \cap A_j$ is a proper cut. By Theorem 6.6.1, there is a subset of indices $Q \subseteq [\kappa]$ such that $A = \cup_{q \in Q} A_q$ satisfies that $S \cap A$ is not $(1-\delta)$-boundary sparse in A, and $A_j \subseteq A$.

   Similarly to the first case, we have that $S \cap A$ is not $(1-\delta)$-boundary sparse in A. Thus $w(S \cap A, A \setminus S) \geq (1-\delta)\min\{w(A \cap S, V \setminus A), w(A \setminus S, V \setminus A)\}$. Let X be the set among $A \cap S$, $A \setminus S$ that satisfies $w(X, V \setminus A) = \min\{w(A \cap S, V \setminus A), w(A \setminus S, V \setminus A)\}$. It follows that $w(X, A \setminus X) = w(S \cap A, A \setminus S) \geq (1-\delta)w(X, V \setminus A)$. We thus have that $w(X, V \setminus X) = w(X, V \setminus A) + w(X, A \setminus X) \leq ((1/(1-\delta)+1)w(X, A \setminus X) \leq (\frac{2-\delta}{2(1-\delta)})\lambda = (1 + \frac{\delta}{2-2\delta})\lambda \leq (1 + 3\delta/4)\lambda$. We conclude by Corollary 6.7.7 that we are in Case ?.

□

Note that both lemmata show the existence of a *proper cut* S', i.e., S' is required to be both nonempty and different from V. This concludes the proof of Proposition 6.7.5.

6.7.2 Cluster Hierarchy

The $(\alpha, \phi, \lambda_{\max}, H, \delta)$-cluster decomposition and the mirror clusters are the building blocks of the cluster hierarchy, defined as follows:

Definition 6.7.12 (Cluster Hierarchy)**.** *Let $\hbar$ be a positive integer. An h-cluster hierarchy is a set of graphs $G_1, \ldots, G_\hbar$ with $G_1 = G$ with $\mathrm{Vol}(G_\hbar) \leq \frac{\lambda_{\max}}{\phi}$, together with $\hbar$ many $(\alpha, \phi, \lambda_{\max}, H, \delta)$-cluster decompositions, one for each $G_j, j \in [\hbar]$, and a mirror cluster for each cluster of the pre-cluster decomposition of each graph. Moreover, each $G_j, j > 1$ is obtained from G_{j-1} by contracting each cluster in the cluster decomposition.*

The main point of the $\hbar$-cluster hierarchy is to ensure that we only have to maintain local cuts and their cut-sizes, in other words, to transform any minimum cut into a local cut, if not on this level of the hierarchy then further down in the hierarchy.

We will use the $\hbar$-cluster hierarchy as follows: Let S be a minimum proper cut of cut-size in $[\lambda_{\min}, \lambda_{\max}]$. Assume for the moment, that we have access to all local cuts. By Proposition 6.7.5 there exists in G_1 an approximate cut S'_1 that is either local or that crosses no cluster. In the former case, we are done. In the latter, S'_1 induces a cut in G_2. Again by

Proposition 6.7.5 there exists in G_2 an approximation S_2' of S_1' that is either local or that crosses no cluster. Again, if it is local, we are done. If it is not, then we move on to G_3. Following this logic, we arrive the following proposition:

Proposition 6.7.13. *Let $\hbar$ be a positive integer. In an $\hbar$-cluster hierarchy of G, there exists a local cut S' in a mirror cluster C of the pre-cluster decomposition of a collapsed graph G_j or in the final collapsed graph, that is a $(1+2\delta)^{\hbar}$ approximation of the minimum proper cut S. Moreover, there exists a corresponding cut S in G such that S and S' have the same cut-size. If, in addition, $\delta < \frac{1}{2\partial S}$, then $\partial S = \partial S'$, that is, S' is a minumum proper cut.*

Proof. It suffices to apply Proposition 6.7.5 iteratively starting at G until we end up in Case 2. of that proposition. If we never arrive in Case 2, it follows that the recursion reaches the graph $G_\hbar$, which has volume $\frac{\lambda_{\max}}{\phi}$. Thus, all its cuts are local. Once a local cut is found, to find the corresponding cut in G, since all graphs (including mirror clusters) in the hierarchy are contractions of graphs from the level above, it suffices to uncontract the graph. Uncontracting all the way to the top gives the result. The approximation value comes from the fact that on each level, the cut-size of the cut increases by a factor of at most $(1+2\delta)$. □

The goal of further sections will thus be to compute and maintain such a cluster hierarchy.

6.8 The Mirror Cuts Data Structure

In this section, we present the Mirror Cuts Data Structure, which maintains the mirror graph and mirror cuts under a sequence of batch edge updates. A batch update is an update that includes many edge insertions and deletions at once.

It is necessary to consider batch updates, as the data structure is designed to handle the refinement of clusters, that is, replacing a cluster C by $S \subsetneq C$ and $C \setminus S$. Typically, this is done by deleting all of the edges between S and $C \setminus S$. Doing so one by one proves to not work, as our data structure relies on the LocalKCut algorithm, which is only guaranteed to be correct if at every request, one can ensure that the minimum cut of the cluster is large enough. However, by deleting the edges one by one, the two clusters are only considered two different ones by the LocalKCut when they are not connected anymore, and thus C is still a cluster, with minimum cut exactly 1 just before the deletion of the last edge. This breaks our data structure.

Hence, we introduce batch updates, so all of the edge deletions can be forwarded to the LocaKCut algorithm *before* any request to it is made.

The results in this section rely on the results of Section 6.5, and we assume that the data structure we present in Section 6.11 supports our cluster decomposition. First, we remind the reader about mirror clusters, mirror graphs and mirror cuts:

Definition 6.7.4 (Mirror clusters, graphs and cuts)**.** *The* mirror cluster *C' of a cluster C in graph G is the graph $G/(V \setminus C)$, that is, the graph where all nodes outside of C have been contracted into one node.*

A mirror graph *$G_{\mathcal{P}_1}$ of G according to partition $\mathcal{P}_1$ is the graph that is the collection of all mirror clusters of the clusters in $\mathcal{P}_1$.*

A mirror cut S_v *of a vertex* $v \in V$ *of a graph* G*, given a partition* $\mathcal{P}_1$*, a volume* ν *and a maximum cut value* $\lambda_{\max}$*, is defined as follows. Let* $C_v \in \mathcal{P}_1$ *be the cluster of* G *such that* $v \in C_v$*. Then* S_v *is a proper cut of smallest cut-value among proper local cuts* $S \subsetneq C_v$ *of cut-value at most* $\lambda_{\max}$ *and that contain* v*. Note that many cuts can be mirror cuts of* v*, and that such a cut may also not exist.*

Theorem 6.8.1 (Mirror Cuts)**.** *Let* $G_{\mathcal{P}_1} = (V, E)$ *be a fully-dynamic mirror graph under batch updates (updates with up to* m *many insertions or deletions) of a graph* G *and partition* $\mathcal{P}_1$ *and* $\lambda_{\max}, \nu, \beta \in \mathbb{N}_+$ *satisfying* $\beta \leq \lambda_{\max} \leq \nu$*, and no mirror cluster* $C' \subseteq G_{\mathcal{P}_1}$ *strictly contains a cut* S *with* $\partial_{C'} S \leq \frac{\lambda_{\max}}{\beta}$*.*

There exists a data structure that maintains, for each vertex $v \in V$*, one of its mirror cuts* S_v*. It can return in constant time the value of the smallest mirror cut* $S_{v_{\min}}$*, a pointer to* $v_{\min}$ *and in* $|S_{v_{\min}}|$ *pointers to the vertices of* $S_{v_{\min}}$*.*

The algorithm runs in $\tilde{O}((m+n)\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ *preprocessing time,* $\tilde{O}(\beta^4(\lambda_{\max}\nu)^{O(\beta)})$ *update time per edge update,* $\tilde{O}(K \cdot \beta^4(\lambda_{\max}\nu)^{O(\beta)})$ *update time for an update with* K *edge updates, and* $O(1)$ *request time.*

The idea is as follows. We run an instance of LocalKCut (Theorem 6.5.1) on $G_{\mathcal{P}_1}$, with parameters $\lambda_{\max}, \nu, \beta$, updating it as updates arrive. As no mirror cluster $C' \subseteq G_{\mathcal{P}_1}$ strictly contains a cut S with $\partial_{C'} S \leq \frac{\lambda_{\max}}{\beta}$, any request on a node $v \in C'_v$ yields all cuts $S \subsetneq C'_v$ with $\mathrm{Vol}(S) \leq \nu$, $\partial_{C'_v} S \leq \lambda_{\max}$ and $v \in S$.

When an edge is deleted, what can happen is that this edge used to contribute to a cut that was too large to be the only mirror cut of a node v, and thus was not necessarily the one maintained by the data structure, but it now needs to be one. This cut is easily found by formulating a request on both endpoints of that edge, and then can be stored at any vertex it includes for which it is a mirror cut.

When an edge is inserted, it might contribute to a cut that was a mirror cut, and thus maintained, but is not anymore after the insertion. In fact, there might be many such cuts, but we show below that there cannot be too many of them. We must then formulate a request to LocalKCut from all those vertices whose maintained cut has changed due to that edge insertion.

Finally, we deal with set deletions by simply deleting all the edges in the set in a batch update.

6.8.1 Data Structure and Algorithm Description

We store these cuts using the following data structure:

- For each mirror cut, we keep pointers to all the vertices for which it is the mirror cut. We also keep pointers to all the vertices it contains. We moreover keep the cut-size of the cut. We delete the cut once it is not a mirror cut anymore, as described in Algorithm 6.8.1.
- For each vertex in a cluster C we keep a pointer to *one of its* mirror cuts and its corresponding cut-size. We also keep pointers to all the mirror cuts of other vertices that it is contained in.

- We also maintain a heap of the mirror cuts, prioritized by their cut-sizes.

Algorithm 6.8.1 (Deleting a mirror cut)**.** *Whenever the mirror cut of a node u changes from S to S', we remove u from the list of nodes stored at S, for which S is the mirror cut. We check if this list becomes empty. If it is the case, we delete all pointers from and to the nodes it contains and then delete the mirror cut and update the heap accordingly.*

Lemma 6.8.2. *Algorithm 6.8.1 takes $O(\nu)$ time as a mirror cuts has volume $O(\nu)$.*

The high-level description of the Mirror Cuts Data Structure is as follows: During preprocessing of this data structure, we build this data structure. To process an edge insertion, we note that the edge insertion might increase the cut-size of the mirror cuts it is incident to. We must therefore request LocalKCut on all the vertices whose stored cut has been affected by this insertion, as the stored cut might not be the mirror cut anymore. Note that we can ensure that not many such nodes exist. To process an edge deletion, we note that the deletion can decrease the cut-size of a cut that was not a mirror cut before the deletion, but becomes one after this deletion. We can find all such cuts by running LocalKCut from the endpoints of the deleted edge.

We next give the details of this algorithm.

Algorithm 6.8.2 (Mirror Cuts Data Structure)**.**

Input:

1. *A mirror graph $G_{\mathcal{P}_1}$ of graph G, where edges can be inserted and deleted in batches*
2. *A maximum cut value $\lambda_{\max}$*
3. *A maximum volume value ν*
4. *An approximation ratio β*

Data Structure:

1. *For every node, we maintain its Mirror Cut and the corresponding value.*
2. *For every Mirror Cut, we maintain pointers from and to the vertices it contains in a sorted list.*
3. *A LocalKCut data structure for $G_{\mathcal{P}_1}$ with parameters $\lambda_{\max}, \nu, \beta$*
4. *A heap on the vertices prioritized by the cut-size of the associated mirror cut*

Preprocessing:

1. *Instantiate the LocalKCut data structure*
2. *Process every node using Algorithm 6.8.3.*

3. *Instantiate the heap*

Handling updates:

1. *Update the LocalKCut data structure.*
2. *For every insertion of an edge (u, v):*
   a) *Update the value of all maintained mirror cuts that contain exactly one of its endpoints u or v. Update the heap accordingly.*
   b) *Mark for processing all the nodes u whose maintained mirror cut has seen its value increase because of the update.*
3. *For every deletion of an edge (u, v):*
   a) *Update the value of all maintained mirror cuts that contain either u or v but not both. Update the heap accordingly.*
   b) *Mark for processing the endpoints of the deleted edge.*
4. *Call Algorithm 6.8.3 for all the nodes marked for processing.*

Algorithm 6.8.3 (Processing)**.** *Input: a vertex v.*

1. *Issue a LocalKCut query for v, which returns a set of cuts $\mathcal{S}$.*
2. *For every cut $S \in \mathcal{S}$ with $\partial_{G_{\mathcal{P}_1}} S \neq 0$, for every $u \in S$, if S is a smaller cut than the one maintained for u:*
   (a) *Build a data structure for S if S was no mirror cut before the update.*
   (b) *Store S as mirror cut of u.*
   (c) *Remove the old mirror cut of u*

We first show the correctness of the above algorithm and then analyze its running time.

6.8.2 Correctness

Lemma 6.8.3. *The Mirror Cuts Data Structure (Algorithm 6.8.2) maintains a mirror cut of G for every vertex $v \in G$ according to partition $\mathcal{P}_1$.*

Proof. We will prove this by induction on the number t of batch updates. The base case is clear, as after preprocessing, after all nodes are processed, the minimum local cut around each node is found, as guaranteed by Theorem 6.5.1 and then stored. For the induction step, assume that after update t, for $t \geq 0$, we have that all cuts stored are mirror cuts.

Let u be a node and let S be its maintained mirror cut after the batch of updates t. Note that S could still be a mirror cut of u at time $t + 1$, in which case there is nothing to do, and no processing can change S into another cut. Indeed, Algorithm 6.8.3 will only overwrite it if

it finds a smaller local cut, which is impossible. Assume next that there exists a local cut S' such that $u \in S'$ and $\partial_{t+1}S' < \partial_{t+1}S$. By definition, we have that $\partial_t S \leq \partial_t S'$. Then either we have that the cut value of S has increased, or the one of S' has decreased. In the first case, an edge insertion (u, v) with $u \in S'$, $v \notin S'$ forces the processing of u. S' is then found by processing u, as S' has small volume and thus is found by the request to LocalKCut. In the second case, an edge deletion (u, v) with $u \in S'$, $v \notin S'$ forces the processing of u, which finds S'. Both processing steps will update the cut associated with u. □

6.8.3 Running time analysis

Lemma 6.8.4. *Processing a vertex with Algorithm 6.8.3 takes $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time.*

Proof. Each time we process a node, we make a request to LocalKCut. By Theorem 6.5.1, each request takes $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time. It outputs at most $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ many cuts, and each cut can update at most ν many vertices, which in turn could each enforce the deletion of one maintained cut, which takes $O(\nu)$ time by Lemma 6.8.2. Hence, the deletions of maintained cuts can take time $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$.

Moreover, each cut found by LocalKCut might need to be stored. In the worst case, each cut needs to be created in the Mirror Cuts data structure, and be linked to all its vertices, this takes time $O(\nu)$. Overall, the running time is $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$. □

Corollary 6.8.5. *Preprocessing of the Mirror Cuts Data Structure (Algorithm 6.8.2) on a mirror graph $G_{\mathcal{P}_1}$ with m edges and n nodes takes $\tilde{O}((m+n)\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time.*

Proof. By Theorem 6.5.1, Step 1 of preprocessing takes $\tilde{O}(m\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time. By Lemma 6.8.4, Step 2 takes $\tilde{O}(n\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time. Step 3 takes $\tilde{O}(n)$ time. □

Corollary 6.8.6. *Handling an edge deletion in Steps 3 and 4 in the Mirror Cuts Data Structure (Algorithm 6.8.2) takes $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time.*

Proof. Let $e = (u, v)$ be a deleted edge. We start by arguing that there are not many mirror cuts that need to be updated in Step 3a. Consider a mirror cut S containing node u, that is, there exists a node u' such that S is the mirror cut of u', and $u \in S$. All of these cuts are found by a request on node u to LocalKCut, which takes $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time, and thus there are at most $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ such cuts. Hence, deleting an edge can affect at most $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ many cuts whose cut-size have to be updated. Finding those cuts in the data structure is trivial as one can access all cuts including u or v from the endpoints of the deleted edge and check in $\tilde{O}(1)$ whether they include both or only one vertex by looking at the sorted list of nodes in each cut, and thus Step 3a takes $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time.

For Step 3b, the deleted edge marks two vertices for processing in Step 4. Processing both endpoints of the deleted edge takes $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time by Lemma 6.8.4, hence the result follows. □

Corollary 6.8.7. *Handling an edge insertion in Steps 2 and 4 in the Mirror Cuts Data Structure (Algorithm 6.8.2) takes $\tilde{O}(\beta^4(\lambda_{\max}\nu)^{O(\beta)})$ time.*

Proof. Let $e = (u, v)$ be an inserted edge. We start by arguing that there are not many mirror cuts that need to be updated in Step 2a. As in the case for edge deletions, consider a mirror cut S containing node u, that is, there exists a node u' such that S is the mirror cut of u', and $u \in S$. All of these cuts would be found by a request on node u to LocalKCut, which takes $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time, and thus there are at most $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ such cuts. Hence, inserting an edge can affect at most $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ many cuts whose cut-size have to be updated. Finding those cuts in the data structure is trivial as one can access all cuts including u or v from the endpoints of the deleted edge and check in $\tilde{O}(1)$ whether they include both or only one vertex by looking at the sorted list of nodes in each cut, and thus Step 2a takes $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time.

Each affected cut in Step 2a induces the marking in Step 2b for processing in Step 4 of all of the nodes it is the mirror cut of, which there might be $O(\nu)$ of. Hence, inserting an edge can affect at most $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ many nodes that all need to be processed, which takes $\tilde{O}(\beta^2(\lambda_{\max}\nu)^{O(\beta)})$ time each by Lemma 6.8.4, hence the result follows. □

Lemma 6.8.8. *Handling a batch of updates with K edge updates takes $\tilde{O}\left(K \cdot \beta^4(\lambda_{\max}\nu)^{O(\beta)}\right)$ amortized update time.*

Proof. Step 1 takes $\tilde{O}\left(\beta^4(\lambda_{\max}\nu)^{O(\beta)}\right)$ per edge by Theorem 6.5.1. The times for Step 2 and Step 3 are discussed in the two corrolaries just above. □

6.9 Static $(1+\delta)$-Approximate Minimum Proper Cut Algorithm for a Restricted Minimum Cut Range

For this section, we assume that the data structure we present in Section 6.11 supports our cluster decomposition.

Before diving into the dynamic algorithm, we present a simple static algorithm that we will dynamize in later sections.

Theorem 6.9.1. *Let G be an unweighted undirected graph. Let $c > 0$. Let $\lambda_{\min}$ and $\lambda_{\max}$ be parameters with $\lambda_{\max} \leq 1.2\lambda_{\min} = 2^{O(\log^{3/4-c} n)}$, and assume that the minimum proper cut of G is at least $\lambda_{\min}$. Let $\phi = 2^{-\Theta(\log^{3/4} n)}$ and $\delta = 2^{-\Theta(\log^{3/4-c} n)}$. Then, there is an algorithm that, if $\mathrm{Vol}(G) \neq O(\frac{\lambda_{\max}}{\phi})$:*

1. *Computes a pre-cluster decomposition $\mathcal{P}_1$ of G.*
2. *Computes a corresponding cluster decomposition $\mathcal{P}_2$*
3. *Creates the mirror graph $G_{\mathcal{P}_1}$.*
4. *Calls itself recursively on the contracted graph $G/\mathcal{P}_2$.*
5. *If the minimum proper cut of G is at most $\lambda_{\max}$, returns a $(1+\delta)$-approximation of the minimum proper cut value.*

6. *If the minimum proper cut of G is at most $\lambda_{\max}$, maintains pointers to the cut that achieves the minimum proper cut, and can return on request the nodes of the side S of smaller volume in time $\tilde{O}(|S|)$ or the cut-edges in time $\lambda_{\max} n^{o(1)}$.*

7. *If the value of the minimum proper cut is strictly larger than $\lambda_{\max}$, report "$\lambda > \lambda_{\max}$".*

If $\mathrm{Vol}(G) = O(\frac{\lambda_{\max}}{\phi})$, the algorithm simply returns the minimum proper cut.

This algorithm runs in $m^{1+o(1)}$ time.

The main idea is as follows: we first decompose the graph into expanders, then decompose the expanders further into smaller connected graphs, called *clusters*, so that every minimum proper cut S in the graph can be uncrossed for each cluster that it cuts. This step is decomposed into two substeps, one is to compute the pre-cluster decomposition $\mathcal{P}_1$, and then compute the cluster decomposition $\mathcal{P}_2$. In the end, S can be approximated either by a cut that consists of a union of (full) clusters or by a local cut. We then contract the clusters and recurse on the resulting graph, as clustering only once reduces the number of edges by an $n^{o(1)}$ factor only, and thus we need to recurse multiple times before we get a graph of volume $n^{o(1)}$ on which we can run any static algorithm.

We describe below first the data structure that the algorithm maintains during its computation on the static graph. We then describe the algorithm, called ApproxMinCut algorithm, which basically instantiates the data structures it uses to compute the partition $\mathcal{P}_2$ and then calls itself recursively on the contracted graph. The main work in initializing and updating these data structures is done in the subroutine UpdatePartition , which is presented afterwards. Note that we maintain one ApproxMinCut data structure for each $\lambda_{\max}$, i.e., for each i.

Recall that $\alpha = \frac{1}{\mathrm{poly}\log n}$, $\phi = 2^{-\Theta(\log^{3/4} n)}$ and that $\delta \leq \frac{1}{2\lambda_{\max}}$ with $\delta \leq 0.04$, which holds by our above choice of $c > 0$, $\delta = 2^{O(\log^{3/4-c} n)} \leq 0.04$. We set $\lambda_{\min} = 2^{\Theta(\log^{3/4-c} n)}$ and $\lambda_{\max} \leq 1.2\lambda_{\min}$, $H = 38^h = 2^{-O(\log^{1/2} n)}$, $\phi = \frac{\phi'}{38^h} = 2^{-\Theta(\log^{3/4} n)}$, $\rho = 2^{\Theta(\log^{1/2} n)}$ and $\alpha = \frac{1}{\mathrm{poly}\log n}$.

Note that $\frac{\alpha}{\phi} \geq 1$. We also set $\gamma = 6$.

Algorithm 6.9.1 (Static Algorithm)**.** *The static algorithm works as follows:*

Input:

1. *An unweighted undirected graph G with possibly parallel edges.*
2. *An integer $\lambda_{\min}$ such that the minimum proper cut of G is no less than $\lambda_{\min}$.*
3. *An integer $\lambda_{\max}$ with $\lambda_{\max} \leq 1.2\lambda_{\min} = 2^{\Theta(\log^{3/4-c} n)}$ with $c > 0$.*

ApproxMinCut *Data Structure:*

If G has volume $\Omega(\frac{\lambda_{\max}}{\phi})$:

1. *An approximation value $\beta = 8$.*

2. *An expansion parameter* $\phi = \frac{\phi'}{38^h} == 2^{-\Theta(\log^{3/4} n)}$
3. *A two-level partition* $\mathcal{P}_1, \mathcal{P}_2$ *of* G *defined by a labeling of the edges as "intercluster", "intracluster" or "fragmented".*
4. *A labeling of each cluster* C *as either "well-connected", "poorly-connected", or "fragmented".*
5. *A labeling of each node as "checked" or "unchecked".*
6. *An instance of Algorithm 6.11.1 which maintains the mirror graph* $G_{\mathcal{P}_1}$ *and the contracted graph* $G/\mathcal{P}_2$*. (See Theorem 6.11.1).*
7. *An instance of LocalKCut (Algorithm 6.5.1) on* G *where "intercluster" edges have been removed, and* $\lambda_{\max}, \nu = 4\frac{\lambda_{\max}}{\phi}, \beta$*. (See Theorem 6.5.1)*
8. *An instance of Algorithm 6.8.2, the Mirror Cuts Data Structure, on* $G_{\mathcal{P}_1}$ *and* $\lambda_{\max}, \nu = 4\frac{\lambda_{\max}}{\phi}, \beta$*. (See Theorem 6.8.1)*
9. *A recursive call of the dynamic algorithm on* G *where "intercluster" edges have been removed, with* $\lambda_{\min} = \lceil 1.2^i \rceil$ *and* $\lambda_{\max} = \lfloor 1.2^{i+1} \rfloor$ *for all* $i \in \mathbb{N}_0$ *satisfying* $1.2^i \leq \frac{\lambda_{\max}}{8}$*.*
10. *A recursive call of Algorithm 6.9.1 on the contracted graph* $G/\mathcal{P}_2$*.*

Rules for cluster classifications:

1. *Initially, every cluster is "well-connected".*
2. *Any cluster* C *that satisfies* $\partial C \leq 3\lambda_{\max}$ *becomes "poorly-connected"*
3. *In* $\mathcal{P}_2$*, any cluster that is created by the fragmenting algorithm (Algorithm 6.6.1) becomes "fragmented" (note that a "poorly-connected" cluster that goes through the fragmenting algorithm untouched still becomes "fragmented"). This overrides other rules.*

Processing:

(1) If the graph has volume $O(\frac{\lambda_{\max}}{\phi})$*, run a connectivity algorithm then a static minimum cut algorithm ([82]) on all connected components. If the graph has no edges, output "*$\lambda > \lambda_{\max}$*". In any case, stop this call to the algorithm.*

(2) Compute an (α, ϕ')*-boundary-linked expander decomposition of* G*, using the algorithm in [77].*

(3) Run UpdatePartition *(given below) to instantiate the* ApproxMinCut *data structure.*

(4) Run Algorithm 6.9.2 on each expander, which computes a pre-cluster decomposition $\mathcal{P}_1$*. Every time Algorithm 6.9.2 splits a cluster it calls* UpdatePartition *to guarantee that the* ApproxMinCut *data structure reflects the current* $\mathcal{P}_1$*.*

(5) Run the fragmenting algorithm (Algorithm 6.6.1) on each "poorly connected" cluster from $\mathcal{P}_1$ to get a cluster decomposition $\mathcal{P}_2$. Update the data structure accordingly.

(6) Call Algorithm 6.9.1 recursively on the contracted graph $G/\mathcal{P}_2$.

(7) Return the smallest cut value found among the recursive call in Step ? and the request on the Mirror Cuts Data Structure (Algorithm 6.8.2), if this value is smaller than $\lambda_{\max}$. Store the pointer to the cut that achieved this value (if the algorithm is required to return cuts).

If either no cut is found or the smallest found cut is strictly larger than $\lambda_{\max}$, then report "$\lambda > \lambda_{\max}$".

UpdatePartition:

1. *If not yet instantiated, instantiate $\mathcal{P}_1$ by setting the label of every edge to "intracluster".*
2. *Update $\mathcal{P}_1$, by changing the labels of the intercluster edges to "intercluster". Update or instantiate Items 4 and 6. Mark the vertices adjacent to a newly "intercluster" edge as "unchecked".*
3. *Update or instantiate the recursive calls of the* ApproxMinCut *algorithm that maintains Item 9 of the* ApproxMinCut *data structure in increasing order of i, i.e. for all suitable i starting from 0.*

   Stop this call to UpdatePartition *whenever a recursive call returns a cut, cut the partition along this cut and run* UpdatePartition *from scratch.*
4. *Update or instantiate every other structure in the* ApproxMinCut *data structure.*

6.9.1 Correctness of Algorithm 6.9.1

First, let's argue about the base case. For a graph with volume $O(\frac{\lambda_{\max}}{\phi})$, the result holds by correctness of the algorithms we call. This trivially satisfies the requirements of our algorithm.

Let us now argue in the general case. Note that if $\lambda > \lambda_{\max}$, the algorithm cannot find a cut smaller than $\lambda_{\max}$ (as such a cut does not exist), and will report that $\lambda > \lambda_{\max}$.

If $\lambda \leq \lambda_{\max}$, then we need to show that for any minimum proper cut S in G, we can find a $(1+2\delta)$-approximate mincut S' in G with either the calls to the Mirror Cuts Data Structure (Algorithm 6.8.2) in Step 8 or the recursive call of Step ?. For this we rely on the following lemma.

Lemma 6.9.2. *In Algorithm 6.9.1, the contracted graph of Item ? and the mirror clusters of Item 8 preserve the cut values of the corresponding non-contracted cuts of G and given a cut in one of them, one can reconstruct the corresponding cut in G.*

Proof. This is immediate as they are contractions of the original graph, and thus preserve cuts. □

As we prove in Lemma 6.9.8 in Section 6.9.2, the graph is decomposed into a valid cluster pre-decomposition $\mathcal{P}_1$. As we show in Section 6.8, Algorithm 6.8.2 in Item 8 of our data structure correctly finds all local cuts in the mirror graph $G_{\mathcal{P}_1}$ at this level of the cluster hierarchy. Step ? of the ApproxMinCut algorithm calls itself recursively, which ensures the full cluster hierarchy is built until only one or two nodes are left. In the case where $\lambda_{\min} \leq \lambda \leq \lambda_{\max}$, by Proposition 6.7.13, the minimum proper cut can be approximated by a local cut in a mirror graph or in the most-collapsed graph of the cluster hierarchy. Since all such local cuts are computed in Item 8, and their minimum is returned in Step ?, this completes the correctness proof if $\lambda \leq \lambda_{\max}$. Note that ApproxMinCut algorithm only returns a value in Step ? and that it only returns the value of an existing cut in G. In the case where $\lambda > \lambda_{\max}$, since the algorithm only returns the value of an existing cut, it either outputs a value strictly larger than $\lambda_{\max}$, or reports that $\lambda > \lambda_{\max}$. This concludes correctness.

6.9.2 Decomposing Expanders

In this subsection, we present an algorithm, called Algorithm 6.9.2, that implements Step ? in Algorithm 6.9.1. It takes as input an expander decomposition, and outputs a decomposition such that every cluster C with $\partial C \geq 3\lambda_{\max}$ contains no $(1-\delta)$-boundary-sparse cut S with $\partial_G(S) \leq \lambda_{\max}$ and $\mathrm{Vol}(S) \leq \lambda_{\max}/\phi$. It also maintains or instantiates an instance of the Mirror Cuts Data Structure (Algorithm 6.8.2) for $G_{\mathcal{P}_1}$ by calling UpdatePartition when needed.

This algorithm is very similar to the one in [59], but works with a different data structure, described in Section 6.11. We include the new analysis here for completeness.

Algorithm 6.9.2 takes as input an expander decomposition, initializes the data structures, and then repeatedly calls Algorithm 6.9.3 on each cluster until no cluster with an unchecked vertex exists. Algorithm 6.9.3 performs the actual cut finding and graph cutting.

Each node stores a boolean value, which represents whether it is *checked* or *unchecked*. Intuitively, node v being checked means that LocalKCut has already been executed at v and the cut-size of the returned cuts has already been taken into account, while being unchecked means that LocalKCut needs to be executed at v. Algorithm 6.9.2 runs as subroutine Algorithm 6.9.3 that takes a cluster decomposition, a cluster C of that decomposition together with a set of unchecked vertices in C as input and (a) either cuts the cluster along a $(1-\delta)$-boundary sparse cut or (b) certifies that no cut is $(1-\delta)$-boundary sparse. Algorithm 6.9.2 repeats this procedure until no cluster with minimum cut at least $\lambda_{\min}$ containing a $(1-\delta)$-boundary-sparse cut exists. Our algorithm will maintain the following invariant:

Invariant 6.9.1. *For every cluster C and every cut $S \subsetneq C$ such that S is $(1-\delta)$-boundary-sparse in C, $\mathrm{Vol}(S) \leq \frac{\lambda_{\max}}{\phi}$, and $\lambda_{\min} \leq w(S, C \setminus S) \leq \lambda_{\max}$, there exists a node $v \in S$ such that v is unchecked.*

The goal of the algorithm is not to miss any $(1-\delta)$-boundary-sparse cut. We use procedure LocalKCut to find such cuts. *Checking node v* refers to executing LocalKCut with v as starting point. As we will show, it suffices to make sure that for every cut that might be boundary-sparse there is at least one node in the cut on which the subroutine LocalKCut is requested to guarantee that the cut is found.

Note that if a set $S \subseteq C$ has no edges leaving C, then $w(S, V \setminus C) = 0 \leq w(S, C \setminus S)$ and, thus, S cannot be $(1-\delta)$-boundary-sparse in C. Thus, vertices in C that are not incident to a boundary edge of C do not need to be checked in C, i.e., we do not need to mark them as unchecked. Hence, only vertices that are incident to a boundary edge are marked as unchecked, all others are marked as checked.

We call the decomposition of the vertex set after the algorithm terminates the *pre-cluster decomposition*.

Algorithm 6.9.2 (Decomposing Expanders)**.** *Input:*

- *an (α, ϕ')-boundary-linked expander decomposition $V_1, \ldots, V_\ell$, specified by edge labels.*
- *Access to a LocalKCut instance on the graph where intercluster edges have been removed.*
- *The parameters $\lambda_{\max}$, $\lambda_{\min}$, ν, δ.*

1. *set $\mathcal{C} \leftarrow \{V_1, \ldots, V_\ell\}$.*
2. *Mark every node incident to an inter-expander edge as unchecked, and all other nodes as checked.*
3. *While there exists in $\mathcal{C}$ a cluster C satisfying $\partial C \geq 3\lambda_{\max}$ and containing an unchecked vertex, we run the Find and Cut subroutine (Algorithm 6.9.3) on C, which might change $\mathcal{C}$.*

Algorithm 6.9.3 (Find and Cut Subroutine)**.** *Input:*

- *a pointer to a graph G, with a partition $\mathcal{P}_1$ of G defined by a labeling of the edges.*
- *a pointer to a set $C \in \mathcal{P}_1$*
- *a set of unchecked vertices satisfying Invariant 6.9.1*
- *an instance of LocalKCut on G where intercluster edges have been removed.*

For every unchecked vertex v in C:

1. *Make a request to LocalKCut on v. Let $\mathcal{S}$ be the set of outputs from all LocalKCuts.*
2. *For every $S \in \mathcal{S}$:*

   Check if S is $(1-\delta)$-boundary-sparse. If it is the case, cut the cluster along S, and run UpdatePartition *thereby removing C and creating two new clusters S and $C \setminus S$ in $\mathcal{P}_1$, and mark any node incident to a newly cut edge as unchecked. The node v remains unchecked. End the current call to Algorithm 6.9.3.*
3. *Mark the node v as checked and move on to the next unchecked vertex in C.*

Correctness

We need to show that Invariant 6.9.1 holds at the termination of Algorithm 6.9.2. For that, we start by showing that the invariant holds before the while loop in Algorithm 6.9.2, and that each step of Algorithm 6.9.3 which is called in the while loop of Algorithm 6.9.2 maintains Invariant 6.9.1. Thus, the invariant holds after Algorithm 6.9.2 has terminated. As there are no unchecked nodes left in clusters C with $\partial C \leq 3\lambda_{\max}$ at that point in time, it follows that every cluster C with $\partial C \leq 3\lambda_{\max}$ contains no $(1-\delta)$-boundary-sparse cut S with $\partial_G(S) \leq \lambda_{\max}$ and $\mathrm{Vol}(S) \leq \lambda_{\max}/\phi$.

Lemma 6.9.3. *Right after Step 2 in Algorithm 6.9.2, Invariant 6.9.1 holds.*

Proof. Right after Step 2 every cluster is an expander in the expander decomposition. Thus, any cut S in a cluster C can only be $(1-\delta)$-boundary-sparse if it contains a node v that is incident to an inter-expander edge, that is, if $w(S, V \setminus V_i) \neq 0$ as we have $(1-\delta)w(S, V \setminus V_i) > w(S, V_i \setminus S) \geq 0$, by the definition of boundary-sparsity. As all the nodes incident to all inter-expander edges were marked as unchecked in Step 2, Invariant 6.9.1 holds. □

Lemma 6.9.4. *Cutting a cluster C along a $(1-\delta)$-boundary-sparse cut C', then marking every node adjacent to an edge in $E(C', C \setminus C')$ as "unchecked" maintains Invariant 6.9.1.*

Proof. To show Invariant 6.9.1 we need to show that every $(1-\delta)$-boundary sparse cut S in C' with $\mathrm{Vol}(S) \leq \frac{\lambda_{\max}}{\phi}$ and $w(S, C' \setminus S) \leq \lambda_{\max}$ contains an unchecked vertex. By symmetry the same claim follows for $C \setminus C'$.

Assume by contradiction that there exists a $(1-\delta)$-boundary sparse cut S with $\mathrm{Vol}(S) \leq \frac{\lambda_{\max}}{\phi}$ and $w(S, C' \setminus S) \leq \lambda_{\max}$ such that no vertex in S is unchecked after marking every node adjacent to an edge in $E(C', C \setminus C')$ as "unchecked" , i.e. every vertex in S is checked. As cutting does not change the marking of the vertices, this implies that (a) every vertex in S was also checked in C, and (b) no edge incident to S was cut, i.e., became a new inter-cluster edge as the endpoints of such edges are marked "unchecked" in Step 2 of Algorithm 6.9.3. Specifically, there exists no edge from a node in S to a node in $C \setminus C'$. To achieve a contradiction we will show that S must have been (i) a $(1-\delta)$-boundary-sparse cut in C, (ii) that S has volume at most $\lambda_{\max}/\phi$, and (iii) $\lambda_{\min} \leq w(S, C \setminus S) \leq \lambda_{\max}$. As, by assumption, Invariant 6.9.1 was satisfied in C before Step 2, it follows that S must have contained an unchecked vertex in C, leading to a contradiction.

It remains to show (i) to (iii). We first show (i). Recall that $C' \subset C$. Then we know that $w(S, C' \setminus S) = w(S, C \setminus S)$, as otherwise there would be an edge from a node in S to a node in $C \setminus C'$. Thus $w(S, C \setminus S) = w(S, C' \setminus S)$ and $w(S, V \setminus C') = w(S, V \setminus C)$. Since C' is a $(1-\delta)$-boundary-sparse cut in C, we have that $w(C', C \setminus C') \leq (1-\delta)w(C \setminus C', V \setminus C) < w(C \setminus C', V \setminus C)$. Also since S is $(1-\delta)$-boundary sparse in C', it holds that $w(S, C' \setminus S) \leq (1-\delta)w(C' \setminus S, V \setminus C')$.

Therefore,

$$\begin{aligned} w(S, C \setminus S) = w(S, C' \setminus S) &\leq (1-\delta) w(C' \setminus S, V \setminus C') \\ &\leq (1-\delta)\left(w(C' \setminus S, V \setminus C) + w(C' \setminus S, C \setminus C')\right) \\ &\leq (1-\delta)\left(w(C' \setminus S, V \setminus C) + w(C', C \setminus C')\right) \\ &< (1-\delta)\left(w(C' \setminus S, V \setminus C) + w(C \setminus C', V \setminus C)\right) \\ &= (1-\delta) w(C \setminus S, V \setminus C) \end{aligned}$$

Furthermore, since S is $(1-\delta)$-boundary sparse in C', it also holds that

$$w(S, C \setminus S) = w(S, C' \setminus S) \leq (1-\delta) w(S, V \setminus C') = w(S, V \setminus C)$$

Thus S is $(1-\delta)$-boundary sparse in C.

We next show (ii). As $S \subset C' \subset C$, it follows that the volume of S is the same, and thus, it is at most $\lambda_{\max}/\phi$.

Finally we show (iii). As there exists no edge from a node in S to a node in $C \setminus C'$ it follows that $w(S, C' \setminus S) = w(S, C \setminus S)$ and, thus, $w(S, C \setminus S) = w(S, C' \setminus S) \leq \lambda_{\max}$.

Thus (i) to (iii) hold, completing the proof of the lemma. □

Lemma 6.9.5. *If* UpdatePartition *cuts along a cut C' in a cluster C, C' is $(1-\delta)$-boundary sparse in C.*

Proof. UpdatePartition cuts along cuts that are returned by a recursive call from Item 9, that is, where the cut-size is at most $\frac{\lambda_{\max}}{8}$. More specifically, in a cluster C, it cuts along a cut C' which satisfies $w(C', C \setminus C') \leq \frac{\lambda_{\max}}{8}$. However, $w(C', C \setminus C') + w(C', V \setminus C) \geq \lambda_{\min}$ by the assumptions of Theorem 6.9.1, and therefore, $w(C', V \setminus C) \geq \lambda_{\min} - \frac{\lambda_{\max}}{8} \geq 0.7\lambda_{\max}$ (where we use that $\lambda_{\max} \leq 1.2\lambda_{\min}$). Similarly, $w(C \setminus C', V \setminus C) \geq 0.7\lambda_{\max}$. This proves that C' is $(1-\delta)$-boundary-sparse in C for $\delta \leq 0.82$, which holds for our choice of δ. □

Lemma 6.9.6. *Each execution of Algorithm 6.9.3 maintains Invariant 6.9.1 if there are no cuts of cut-size at most $\frac{\lambda_{\max}}{8}$ in the cluster C given as second parameter.*

Proof. **Step 1:** Step 1 of Algorithm 6.9.3 neither affects the marking of vertices nor the cluster decomposition and, thus, does not affect Invariant 6.9.1.

Step 2: We will next show that the invariant is maintained in Step 2. In Step 2 a cluster C is cut along a $(1-\delta)$-boundary-sparse cut $C' \subsetneq C$ into a new cluster C' and into $C \setminus C'$ and the endpoints of new inter-cluster edges are marked as unchecked, whether this happens in Algorithm 6.9.3 itself or the call to UpdatePartition (Lemma 6.9.5), which by Lemma 6.9.4 maintains the invariant.

Step 3: It remains to show that the invariant is maintained in Step 3. If this step is executed, (a) the status of only one node can change, namely the one of v and (b) the execution of the for-loop for v modified neither the cluster decomposition nor the status of any other unchecked node. Thus Invariant 6.9.1 continues to hold for all clusters that do not contain v. We need to argue that it also holds for the (unique) cluster C_v that contains v. For that, we will start by making the following claim, which is crucial to ensure that LocalKCut finds all the cuts necessary for our analysis:

Claim 6.9.7. *For any cluster C and any subset $\varnothing \subsetneq S \subsetneq C$, at any point outside of* UpdatePartition*, we have that $\partial_C S \geq \frac{\lambda_{\max}}{8}$.*

Proof. Note that apart from inside UpdatePartition , the only two steps in the algorithm where $\mathcal{P}_1$ changes is in Steps ? and ?. Any change is immediately followed by a call to UpdatePartition , which then runs until $\partial_C S \geq \frac{\lambda_{\max}}{8}$ for all $C \in \mathcal{P}_1$, $S \subseteq C$, hence the claim. □

We now continue arguing about Step 3. Assume by contradiction that a cut S exists in cluster C_v at the end of Step 3 such that S (a) is $(1-\delta)$-boundary sparse in C_v, (b) has volume at most $\lambda_{\max}/\phi$, (c) satisfies $w(S, C_v \setminus S) \leq \lambda_{\max}$, and (d) contains only checked vertices. As Invariant 6.9.1 held at the beginning of Step 3, S must have had an unchecked vertex then and as v is the only vertex whose status changed since then, v must belong to S.

By Claim 6.9.7, in C there are no cuts of cut-size at most $\frac{\lambda_{\max}}{8}$, and we have that $\partial_C S \leq \lambda_{\max}$. Hence S is a 8-approximate minimum cut in C, and a call to LocalKCut with vertex v and $\beta = 8$ is guaranteed to find S. and that LocalKCut is run with approximation parameter $\beta = 8$ by Theorem 6.5.1. But then, as S is found, Algorithm 6.9.3 would have cut it instead in Step 2 instead of proceeding to Step 3 and of unchecking v, and thus we have a contradiction. □

Now we can finish the correctness proof.

Lemma 6.9.8. *It holds that (a) at the termination of Algorithm 6.9.2 Invariant 6.9.1 holds; (b) the partition $\mathcal{P}_1$ computed by Algorithm 6.9.2 satisfies that no cluster C with $\partial C \geq 3\lambda_{\max}$ contains a $(1-\delta)$-boundary-sparse cut of cut-size at most $\lambda_{\max}$ and volume at most $\frac{\lambda_{\max}}{\phi}$, and thus $\mathcal{P}_1$ is a pre-cluster decomposition.*

Proof. Since we only uncheck a vertex if it is contained in no cuts that are $(1-\delta)$-boundary-sparse with cut value at most $\lambda_{\max}$ and volume at most $\frac{\lambda_{\max}}{\phi}$, this lemma relies on Invariant 6.9.1. By Lemma 6.9.3, the invariant holds at the beginning of the while loop in Algorithm 6.9.2.

We now have to show that the invariant holds until the end of the while loop. This is now immediate as the while loop is simply a call to Algorithm 6.9.3, and Algorithm 6.9.3 does not affect the invariant by Lemma 6.9.6.

The fact that the while loops in Algorithm 6.9.2 ends when each cluster C either satisfies $\partial C < 3\lambda_{\max}$ or contains no unchecked vertex, together with Invariant 6.9.1, concludes the proof. □

Bounding the number of inter-cluster edges

This subsection is dedicated to the analysis of the number of inter-cluster edges in our final cluster decomposition, i.e., it proves Proposition 6.9.9.

Proposition 6.9.9. *Let M be the number of inter-expander edges after the expander decomposition in Algorithm 6.9.1. Then there are $\tilde{O}\left(\frac{M}{\delta^3}\right)$ many inter-cluster edges in the cluster decomposition after the execution of Step ? of Algorithm 6.9.2.*

Consider the following rooted forest F in which each node represents a cluster of G at *some point* during the computation: the roots are the expanders output by the expander decomposition. For each node representing a cluster, its two children (if any) are the clusters formed by splitting the corresponding cluster. By slight abuse of notation we do not distinguish between clusters of the cluster decomposition and nodes in the forest.

Note that a cluster is always split along a $(1-\delta)$-boundary-sparse cut, whether it is in UpdatePartition, Algorithm 6.9.3 or Algorithm 6.6.1.

We first show that the boundary-size of a node is always smaller than the one of its parent by at least $\delta\lambda_{\min}/2$.

Lemma 6.9.10. *Let C be a node in F and let $S, C \setminus S$ be its children. Then we have that $\partial C \geq \max\{\partial S, \partial(C \setminus S)\} + (\delta\lambda_{\min})/2$.*

Proof. Since S is a $(1-\delta)$-boundary-sparse cut, we have that

$$w(S, C \setminus S) \leq (1-\delta)\min\{w(S, V \setminus C), w(C \setminus S, V \setminus C)\},$$

which implies that $w(C \setminus S, V \setminus C) > w(C \setminus S, S)$. As it holds that $w(C \setminus S, V \setminus C) + w(C \setminus S, S) = \partial(C \setminus S) \geq \lambda_{\min}$, it follows that $w(C \setminus S, V \setminus C) \geq \frac{\lambda_{\min}}{2}$.

The boundary-sparseness of S also implies that

$$\partial S = w(S, V\setminus C) + w(S, C\setminus S) \leq w(S, V\setminus C) + (1-\delta)w(C\setminus S, V\setminus C) = \partial C - \delta w(C\setminus S, V\setminus C)$$

i.e.,

$$\partial C \geq \partial S + \delta w(C \setminus S, V \setminus C) \geq \partial S + \delta\lambda_{\min}/2.$$

The same bound holds by symmetry for $\partial(C \setminus S)$ and the result follows. □

Nonte that the lemma implies that $\partial C > \partial S$ and $\partial(C \setminus S)$. Thus, along any path in F from a root to a leaf the boundary-size of the clusters strictly decreases. Specifically, if a cluster has boundary-size less than some value, let's say $3\lambda_{\max}$, then all its descendants in F has boundary-size less than $3\lambda_{\max}$. We now look only at the subforest F' of F created from F by deleting from F all internal nodes with boundary at most $3\lambda_{\max}$. More formally, F' is defined as follows: All roots of F are also in F', and moreover, for every $C \in F'$ with $\partial C \geq 3\lambda_{\max}$, all its children in F are also in F'. However, for $C \in F'$ with $\partial C < 3\lambda_{\max}$, their children do not belong to F', i.e., C is a leaf in F'. As discussed, $F \setminus F'$ contains no node of boundary larger than $3\lambda_{\max}$.

Lemma 6.9.11. *The total number of inter-cluster edges of the leaves of F' is at most $O(\frac{M}{\delta})$, where M is the number of inter-expander edges output by the expander decomposition.*

Proof. We count the number of inter-cluster edges by bounding the sum of the boundary-sizes for every cluster that is a descendant of a root cluster V_i. Any inter-cluster edge that is created at some descendant of V_i is also an inter-cluster edge at a leaf descendant of V_i in F' since clusters are never merged. Thus, it suffices to bound the sum of the boundary-sizes of the

leaf descendants of V_i in F'. Let $\mathcal{C}_f$ contain exactly these clusters. We bound this sum by splitting $\mathcal{C}_f$ into two sums that we will analyze separately:

$$\sum_{C\in\mathcal{C}_f}\partial C = \sum_{C\in\mathcal{C}_f,\partial C\leq 2.2\lambda_{\max}}\partial C + \sum_{C\in\mathcal{C}_f,\partial C>2.2\lambda_{\max}}\partial C$$

and we define

$$S_1 := \sum_{C\in\mathcal{C}_f,\partial C\leq 2.2\lambda_{\max}}\partial C$$

and

$$S_2 := \sum_{C\in\mathcal{C}_f,\partial C>2.2\lambda_{\max}}\partial C.$$

To proceed we will analyze the potential function $\Phi(\mathcal{C}) := \sum_{C\in\mathcal{C}}\Phi(C)$ were $\Phi(C) = \max\{0,\partial C - 2.1\lambda_{\max}\}$. Right after the computation of the expander decomposition the cluster decomposition of V_i only consists of V_i itself (that is, an expander from the expander decomposition). While computing the cluster decomposition, the algorithm might partition a descendant C of V_i, replacing it by two new clusters C_1 and C_2 and making C_1 and C_2 its children in F'. We show next that the sum of the potentials of the two children clusters is at least $(\delta\lambda_{\min})/2$ smaller than the potential of C. To do so we analyze three cases:

1. If both C_1 and C_2 satisfy $\partial C_1, \partial C_2 \geq 2.1\lambda_{\max}$, then the total potential changes by:

$$\begin{aligned}\Phi(C_1)+\Phi(C_2)-\Phi(C) &= \partial C_1 - 2.1\lambda_{\max} + \partial C_2 - 2.1\lambda_{\max} - \partial C + 2.1\lambda_{\max}\\ &= -2.1\lambda_{\max} + 2w(C_1,C_2) \leq -2.1\lambda_{\max} + 2\lambda_{\max} \leq -0.1\lambda_{\max}\end{aligned}$$

2. If wlog C_1 and C_2 satisfy $\partial C_1 \leq 2.1\lambda_{\max} \leq \partial C_2$, then the total potential changes by:

$$\Phi(C_1)+\Phi(C_2)-\Phi(C) = 0+\partial C_2-2.1\lambda_{\max}-\partial C+2.1\lambda_{\max} = \partial C_2-\partial C \leq -(\delta\lambda_{\min})/2$$

   where the last inequality follows from Lemma 6.9.10.

3. If both C_1 and C_2 satisfy $\partial C_1, \partial C_2 \leq 2.1\lambda_{\max}$, then the total potential changes by:

$$\Phi(C_1)+\Phi(C_2)-\Phi(C) = 0+0-\partial C+2.1\lambda_{\max} \leq -0.9\lambda_{\max}$$

It follows that the potential drops by at least $(\delta\lambda_{\min})/2$ each time when a cluster is split into two smaller clusters.

We now bound the number of clusters in $\mathcal{C}_f$ as follows: Since the potential drops by $\delta\lambda_{\min}/2$ each time a cluster is split, and splitting a cluster only increases the number of clusters by one, we know that the total number of clusters in $\mathcal{C}_f$ is at most $\frac{2\Phi(\{V_i\})}{\delta\lambda_{\min}} \leq \frac{2\partial V_i}{\delta\lambda_{\min}}$. As every cluster contributing to S_1 contributes at most $2.2\lambda_{\max}$, it follows that S_1 is at most $\lambda_{\max}\frac{4.4\partial V_i}{\delta\lambda_{\min}} = O\left(\frac{\partial V_i\lambda_{\max}}{\delta\lambda_{\min}}\right)$.

For the second sum, S_2, note that if $\partial C \geq 2.2\lambda_{\max}$, then $\partial C = O(\Phi(C))$. Recall that the sum of potentials of the two children of a node in F' is less than the potential of the parent.

It follows by induction that for any set of descendants of V_i such that none is an ancestor of the other, the sum of their potentials is less than the potential of V_i. As the clusters in C_f fulfill this condition, it follows that $S_2 = \sum_{\substack{C \in \mathcal{C}_f \\ \partial C > 2.2\lambda_{\max}}} O(\Phi(C)) = O(\Phi(\{V_i\})) = O\left(\frac{\partial V_i \lambda_{\max}}{\delta \lambda_{\min}}\right)$.

Summing over all expanders, and using that $\lambda_{\max} = O(\lambda_{\min})$, we get the result. □

We now look at the forest $F'' = (F \setminus F') \cup \text{leaves}(F')$. The leaves of F' are the roots of F''. By the lemma above, we know that the sum of the boundaries of the roots of F'' is at most $O\left(\frac{M}{\delta}\right)$. We moreover know that any cluster C with children in F'' satisfies $\partial C < 3\lambda_{\max}$, and thus the decomposition is made by fragmenting (Algorithm 6.6.1). This gives the following lemma, which is a direct consequence of Theorem 6.6.1:

Lemma 6.9.12. *The sum of the boundaries of the leaves of F'' is at most $\tilde{O}\left(\frac{M}{\delta^3}\right)$, where M is the number of inter-expander edges output by the expander decomposition.*

Running Time Analysis

Now we are ready to analyze the running time of the static algorithm.

Definition 6.9.13 (Responsible Vertex)**.** *Let u be a vertex. We say that u is* responsible *for a cluster S if, in Algorithm 6.9.3, S was cut from its parent cluster after being found by LocalKCut requested on vertex u.*

Lemma 6.9.14. *Let u be a vertex. Vertex u is responsible for at most $O(\frac{1}{\phi})$ many clusters in F.*

Proof. The first cut u is responsible for will reduce the volume of the cluster u is in to at most $\frac{\lambda_{\max}}{\phi}$. Then every further cut will reduce this volume by at least $\lambda_{\min}$ as it cuts a at least $\lambda_{\min}$ edges out of the cluster. As $\lambda_{\max}/\lambda_{\min} = O(1)$ the lemma follows. □

This lemma is important, as after checking a node, it is not guaranteed to become unchecked, as if a boundary-sparse cut of cut-size at least $\lambda_{\min}$ is found, a cut is made, but the node stays unchecked. Hence this bounds how many times a vertex is being checked, that is, how many times we need to run LocalKCut on any individual vertex. It follows that we can bound the total number of call to LocalKCut.

Lemma 6.9.15. *The total number of calls to LocalKCut in Algorithm 6.9.2 is $\tilde{O}\left(\frac{M}{\delta^3}\frac{1}{\phi}\right)$, where $M = O(m\phi)$ is the number of inter-expander edges.*

Proof. A vertex only becomes unchecked when it is incident to an inter-cluster edge. By Lemma 6.9.12 we have $O\left(\frac{M}{\delta^3}\right)$ inter-cluster edges, and, thus, this bounds the total number of times that a vertex becomes unchecked. From each unchecked vertex u we run a call to LocalKCut requested on u in Item 1 of Algorithm 6.9.3. As Lemma 6.9.14 shows, after $O(\frac{1}{\phi})$ such executions of Item 1 vertex u must become checked. Thus the total number of calls to LocalKCut is as stated in the lemma. □

We remind the reader of the following constants: $c > 0$, $\lambda_{\max} \leq 1.2\lambda_{\min} = 2^{O(\log^{3/4-c} n)}$, $\delta \leq \frac{1}{2\lambda_{\max}}$ with $\delta \leq 0.04.$, $\phi = 2^{-\Theta(\log^{3/4} n)}$ and $\alpha = \frac{1}{\text{poly}\log n}$.

Let us now discuss the number of recursive calls. Remember that G is the original graph with m edges. We say that an instance of the algorithm is on level ℓ if it is run on a graph G' that is obtained from G by contracting it ℓ times in Item 10. For example, the instance A_0 of original call contracts the graph and calls recursively the instance A_1, which calls instance A_2 on the same contracted graph with smaller $\lambda_{\min}$ and $\lambda_{\max}$ in Item 9. This instance then calls instance A_3 on another contracted graph, and A_3 calls A_4 in Item 9. A_0 is on level 0, A_1 and A_2 are on level 1, A_3 and A_4 are on level 2.

We also assign label i to each node in that tree where its $\lambda_{\max}$ parameter is $\lfloor 1.2^{i+1} \rfloor$.

Lemma 6.9.16. *The number of levels of Algorithm 6.9.1 is $O(\log^{1/4} n)$.*

Proof. Let A be the number of edges input to an instance of the algorithm. Step ? of the algorithm creates $M = \tilde{O}(A\phi)$ many interexpander edges, as proven in [77]. Lemma 6.9.12 shows that then the number of intercluster edges, which is the number of edges in the input graph of the $\ell+1$-th recursive call is $\tilde{O}(A\phi \cdot \frac{1}{\delta^3}) \leq \frac{A}{2^{a\log^{3/4} n}}$ for some $a \in \mathcal{R}_+$. Hence, the number of edges at the r-th level of the cluster decomposition is at most $\frac{m}{2^{ar\log^{3/4} n}}$. This shows that after $r = \frac{1}{a}\log^{1/4} n$ levels, the number of edges drops to less than $\tilde{O}(1)$. □

Lemma 6.9.17. *Let $\mathcal{A}_i$ be an algorithm indexed on i that calls itself recursively, where $\mathcal{A}_i$ calls $\mathcal{A}_j$ each at most once for every $0 \leq j < i$. Overall, a call to $\mathcal{A}_i$ runs at most 2^i many instances of the algorithm.*

Proof. We prove the statement by induction on i. The case $i = 0$ is straightforward. For the inductive step, assume that a call to $\mathcal{A}_j$ runs 2^j many instances of the algorithm for all $j \leq i$, and let us prove the statement for the case $i+1$.

Let us consider a call to $\mathcal{A}_{i+1}$. We thus have a call to all of $\mathcal{A}_j$ for $0 \leq j < i+1$. By the induction hypothesis, we have that the total number of instances called recursively is at most $\sum_{0\leq j<i+1} 2^j = 2^{i+1} - 1$. Adding the instance $\mathcal{A}_{i+1}$ to that total concludes the proof. □

Lemma 6.9.18. *Let us consider an instance labeled i on level ℓ. This instance calls overall at most 2^i many instances on level ℓ.*

Proof. Remember that a call on such an instance calls recursively on the same graph with $\lambda_{\min} = \lceil 1.2^i \rceil$ and $\lambda_{\max} = \lfloor 1.2^{i+1} \rfloor$ for all $i \in \mathbb{N}_0$ with $1.2^i \leq \lambda_{\max}/8$. We can ignore all recursive calls on the contracted graphs, as those will only create calls on higher levels.

Let us look at the tree of recursive calls, Note that any children labeled j of a node labeled i must satisfy $j + 10 \leq i$ as we have $1.2^j \leq \frac{\lfloor 1.2^{i+1} \rfloor}{8} \leq \frac{1.2^{i+1}}{8} \leq \frac{1.2^{i+1}}{1.2^{11}}$. We relax this condition and note that $j + 1 \leq i$. Then, by Lemma 6.9.17, this creates at most 2^i many recursive calls. □

Lemma 6.9.19. *The total number of recursive calls is $n^{o(1)}$.*

Proof. Note that every instance on level ℓ calls immediately exactly one instance on level $\ell+1$ in the limit of $\ell = O(\sqrt[4]{\log n})$ many levels, by Lemma 6.9.16. Moreover, every call on level ℓ and index i generates overall 2^i many instances on level ℓ, by Lemma 6.9.18.

Therefore, the initial call on level 0 and label $i = O(\log \lambda)$ creates at most 2^i many instances on level 0. By induction, it is straightforward to see that there are at most $2^{i(\ell+1)}$ many instances on level ℓ. Plugging in $1.2^i = O(\lambda_{\max})$ we have that the total number of recursive calls is equal to $\lambda_{\max}^{O(\sqrt[4]{\log n})} = 2^{O(\log^{1-c} n)}$. □

Theorem 6.9.20. *The total running time of Algorithm 6.9.1 is $O(m^{1+o(1)})$.*

Proof. Let us first analyze one level of the recursion: Step ? takes $O(m^{1+o(1)})$ as proven in [77]. It creates $M = m\phi$ inter-expander edges.

Step ? takes $O(m)$ time.

By Lemma 6.9.15, Step ? runs LocalKCut a total of $\tilde{O}\left(\frac{M}{\delta^3}\frac{1}{\phi}\right)$ times, each instance taking $n^{o(1)}$, for a total of $O(m^{1+o(1)})$ time overall.

By Theorem 6.6.1, Step ? takes $O(n^{o(1)})$ per intercluster edge at most. Steps ?, and ? take $O(m)$ time (excluding the recursive call).

By Theorem 6.11.1, the rest of the data structure can be maintained in $O(m^{1+o(1)})$, as the number of updates is upper-bounded by m and the volume is $n^{o(1)}$

By Lemma 6.9.16, the number of levels is $n^{o(1)}$. Hence the claim follows. □

6.10 The Dynamic Algorithm

The dynamic algorithm works as follows: at the beginning, the algorithm runs the static algorithm, which sets up the data structure. In particular, on each layer, it provides a pre-cluster decomposition $\mathcal{P}_1$ and a corresponding cluster decomposition $\mathcal{P}_2$.

We then maintain this data structure on each layer, by realizing that each intracluster edge insertion and intercluster edge deletion cannot create any $(1-\delta)$-boundary-sparse cut, while for any other update, only a limited number of calls to LocalKCut suffice to find all newly created $(1-\delta)$-boundary-sparse cuts.

In fact, we will simply mark a few nodes after each update as unchecked, so as to make sure that Invariant 6.9.1 holds, and then run the corresponding part of the algorithm again. The goal is to have after each update a cluster decomposition as in Proposition 6.7.5.

Remember that we set $c > 0$, $\lambda_{\max} \leq 1.2\lambda_{\min} = 2^{O(\log^{3/4-c} n)}$, $\delta \leq \frac{1}{2\lambda_{\max}}$ with $\delta \leq 0.04$. $\phi = 2^{-\Theta(\log^{3/4} n)}$ and $\alpha = \frac{1}{\text{poly}\log n}$. Note that $\frac{\alpha}{\phi} \geq 1$ and that we assume $\delta \leq 0.04$. We further set $\psi = 2^{\Theta(\log^{1/2} n)}$, $h = \Theta(\log^{1/2} m)$, and $\rho = 2^{\Theta(\log^{1/2} n)}$.

For this section, we assume that the data structure we present in Section 6.11 supports our cluster decomposition.

We describe below the main structure of the dynamic algorithm:

Algorithm 6.10.1. *Data Structure: The data structure is the same as the one of the static algorithm (Algorithm 6.9.1).*

Preprocessing:

noitemsep Run Algorithm 6.9.1

Processing updates:

1. *If the graph has $O(\frac{\lambda_{\max}}{\phi})$ nodes, run a connectivity algorithm then a static algorithm ([82]) on all connected components, and return the minimum cut found. If the graph has no edges, output "$\lambda > \lambda_{\max}$". In any case, stop this call to the algorithm.*
2. *Run Algorithm 6.10.2. Algorithm 6.10.2 is the dynamic algorithm that maintains the pre-cluster decomposition and the cluster decomposition*
3. *Maintain the contracted graph where each cluster output by Algorithm 6.10.2 is contracted into a node, and call recursively Algorithm 6.10.1 on that contracted graph (as described in Section 6.11)*
4. *Output the value of the minimum proper cut found among the cut returned by the recursive call and the mirror clusters, if that value falls in $[\lambda_{\min}, \lambda_{\max}]$. Store pointers to the corresponding cut. If the smallest cut found is larger than $\lambda_{\max}$ (or no cut is found), report "$\lambda > \lambda_{\max}$".*

Since the cluster decomposition satisfies the conditions of Proposition 6.7.5 at every point, this proves correctness.

The subsections below will prove the following lemma:

Lemma 6.10.1. *In Algorithm 6.10.2:*

1. *The amortized recourse[2] per updated edge is $\tilde{O}\left(\frac{\rho}{\delta^3}\lambda_{\max}\right)$*
2. *At every time step, and every level, if $\tilde{m}$ is the number of edges input on that level, the number of intercluster edges in $\mathcal{P}_2$ is at most $\tilde{O}\left(\frac{\tilde{m}\phi}{\delta^3}\right)$, and thus the recursive call will have as input at most $\tilde{O}\left(\frac{\tilde{m}\phi}{\delta^3}\right)$ edges.*
3. *The amortized update time is $n^{o(1)}$ on each level – excluding recursive calls.*

Let us now discuss the number of recursive calls. Remember that G is the original graph with m edges. As for the static algorithm, we say that an instance of the algorithm is on level ℓ if it is run on a graph G' that is obtained from G by contracting it ℓ times in Item 10 of Algorithm 6.9.1. For example, the instance A_0 of original call contracts the graph and calls recursively the instance A_1, which calls instance A_2 on the same contracted graph with smaller $\lambda_{\min}$ and $\lambda_{\max}$ in Item 9. This instance then calls instance A_3 on another contracted graph,

[2]the recourse is the number of modifications to the edges of the contracted graph

and A_3 calls A_4 in Item 9. A_0 is on level 0, A_1 and A_2 are on level 1, A_3 and A_4 are on level 2.

We also assign label i to each node in that tree where its $\lambda_{\max}$ parameter is $\lfloor 1.2^{i+1} \rfloor$.

Lemma 6.10.2. *The number of levels of Algorithm 6.10.1 is $O(\log^{1/4} n)$.*

Proof. Let A be the number of edges input to an instance of the algorithm. By Lemma 6.10.1, the number of edges on the next level is $\tilde{O}\left(\frac{\tilde{m}\phi}{\delta^3}\right)$. Hence, the number of edges at the r-th level of the cluster decomposition is at most $\frac{m}{2^{ar\log^{3/4} n}}$ for some $a \in \mathcal{R}$. This shows that after $r = \frac{1}{a}\log^{1/4} n$ levels, the number of edges drops to less than $\tilde{O}(1)$. □

Lemma 6.10.3. *Let us consider an instance labeled i on level ℓ. This instance calls overall at most 2^i many instances on level ℓ.*

Proof. Remember that a call on such an instance calls recursively on the same graph with $\lambda_{\min} = \lceil 1.2^i \rceil$ and $\lambda_{\max} = \lfloor 1.2^{i+1} \rfloor$ for all $i \in \mathbb{N}_0$ with $1.2^i \leq \lambda_{\max}/8$. We can ignore all recursive calls on the contracted graphs, as those will only create calls on higher levels.

Let us look at the tree of recursive calls, Note that any children labeled j of a node labeled i must satisfy $j + 10 \leq i$ as we have $1.2^j \leq \frac{\lfloor 1.2^{i+1} \rfloor}{8} \leq \frac{1.2^{i+1}}{8} \leq \frac{1.2^{i+1}}{1.2^{11}}$. We relax this condition and note that $j + 1 \leq i$. Then, by Lemma 6.9.17, this creates at most 2^i many recursive calls. □

Lemma 6.10.4. *The total number of recursive calls is $n^{o(1)}$.*

Proof. Note that every instance on level ℓ calls immediately exactly one instance on level $\ell+1$ in the limit of $\ell = O(\sqrt[4]{\log n})$ many levels, by Lemma 6.10.2. Moreover, every call on level ℓ and index i generates overall 2^i many instances on level ℓ, by Lemma 6.10.3.

Therefore, the initial call on level 0 and label $i = O(\log \lambda)$ creates at most 2^i many instances on level 0. By induction, it is straightforward to see that there are at most $2^{i(\ell+1)}$ many instances on level ℓ. Plugging in $1.2^i = O(\lambda_{\max})$ we have that the total number of recursive calls is equal to $\lambda_{\max}^{O(\sqrt[4]{\log n})} = 2^{O(\log^{1-c} n)}$. □

Lemma 6.10.5. *The amortized recourse computed over all recursion levels is $n^{o(1)}$.*

Proof. The total recourse on the i-th level is $\tilde{O}\left((\frac{\rho}{\delta^3}\lambda_{\max})^{i-1}\right)$, as per Lemma 6.10.1. As $i = O(\log^{\frac{1}{4}} n)$ by the lemma above, we get that $\tilde{O}\left((\frac{\rho}{\delta^3}\lambda_{\max})^{i-1}\right) = 2^{O(\log^{1-c} n)} = n^{o(1)}$. Summing over all levels of recursion gives us the desired result. □

Lemma 6.10.6. *The amortized running time over all levels is $n^{o(1)}$ in Algorithm 6.10.2.*

Proof. Each recursion level receives an amortized number of $n^{o(1)}$ updates as per the previous lemma, and requires $n^{o(1)}$ amortized time to process the update, by Lemma 6.10.1. Hence, on each level, after one update on the top level, the time to process that update is $n^{o(1)}$. Summing over all levels gives us the desired result. □

Lemma 6.10.7. *In Algorithm 6.10.2, the total approximation ratio after all calls is 1.*

Proof. Each recursive call degrades the quality of the solution by a factor $(1+2\delta)$ at most, by Proposition 6.7.5.

Note that since we chose $\delta < \frac{1}{2\lambda_{\max}}$, all cuts smaller than $\lambda_{\max}$ are thus preserved by our algorithm. This shows that our algorithm is exact on every level, and thus is exact overall. □

6.10.1 Proof of Lemma 6.10.1

In this subsection, we are given a graph with updates, and our goal is to maintain a cluster decomposition of this graph such that no cluster contains a cut of cut-size at most $\lambda_{\max}$ that is $(1-\delta)$-boundary sparse. As in the static setting the main idea is to maintain a dynamic expander decomposition which we refine further.

Definition 6.10.8 (Edge incident to a cluster)**.** *We say that an edge $e=(u,v)$ is* incident *to a cluster C if either u or v (or both) are in C.*

Algorithm 6.10.2 (Maintaining the cluster decomposition)**.** *Preprocessing: We run the static algorithm on the initial graph, with the preprocessing of Theorem 6.3.5 for the expander decomposition.*

Handling updates: We give the next $O(\frac{m\phi}{\rho})$ updates to the dynamic expander decomposition algorithm. Every change output by that algorithm is then processed in the following way:

1. *Run* UpdatePartition *(from Algorithm 6.9.1), which updates the partition $\mathcal{P}_1$, and the classification of the clusters as "well-connected", "poorly-connected" or "fragmented".*
2. *Mark the endpoints of the updated edges as unchecked.*
3. *Then, for each affected "well-connected" cluster C (i.e. containing an endpoint of an affected edge), find $2\lambda_{\max}$ intercluster edges leaving that cluster whose endpoint in C is checked, and mark their incident vertices in the cluster as unchecked.*
4. *While there exists in $\mathcal{C}$ (the cluster decomposition) a "well-connected" cluster C with an unchecked vertex, we run the Find and Cut subroutine (Algorithm 6.9.3) on C, which might change $\mathcal{C}$.*
5. *For every affected "poorly-connected" or "fragmented" cluster C, revert any "fragmented' edges it contains to "intracluster", and run Algorithm 6.6.1 on C.*

Every $O(\frac{m\phi}{\rho})$ updates, we restart from scratch, i.e., we run the preprocessing step on the full graph.

Rules for Cluster Classification:

1. *By default, every cluster is "well-connected"*
2. *Any cluster C that satisfies $\partial C \leq 3\lambda_{\max}$ becomes "poorly-connected"*
3. *Any cluster C that satisfies $\partial C \geq 6\lambda_{\max}$ becomes "well-connected"*

4. *In $\mathcal{P}_2$, any cluster that is the result of the fragmenting algorithm becomes "fragmented" (note that a "poorly-connected" cluster that goes through the fragmenting algorithm untouched still becomes "fragmented"). This overrides other rules.*

We start by showing the correctness of the algorithm.

Correctness

Lemma 6.10.9. *Let $C \in \mathcal{P}_1$ be a cluster that satisfies Invariant 6.9.1 at time t. Assume that between times t and t', C is subject to updates, and that the Find and Cut subroutine (Algorithm 6.9.3) is not run in between t and t' on the cluster[3]. If we mark every node incident to an edge insertion or deletion as well as the nodes incident to $2\lambda_{\max}$ intercluster edges incident to C apart from the edges inserted between t and t' as unchecked, then C satisfies Invariant 6.9.1 at time t'.*

Proof. Let S be, at time t', a $(1-\delta)$-boundary-sparse cut in C, with $w(S, C \setminus S) \leq \lambda_{\max}$, and $\mathrm{Vol}(S) \leq \frac{\lambda_{\max}}{\phi}$. We need to show that S contains an unchecked vertex at time t'. We have multiple cases:

Case 1: S contains an unchecked vertex at time t:

In that case, the same vertex is still unchecked at time t'. Indeed, since the Find and Cut subroutine was not run between t and t', no unchecked vertex becomes checked between the two times.

Case 2: There is an edge with at least one endpoint in S that was inserted or deleted. That update causes the corresponding endpoint to become unchecked. Since the Find and Cut subroutine was not run between t and t', this endpoint is still unchecked at time t'.

Case 3: If all vertices of S were checked at time t and no edge update was incident to S between t and t': It follows by Invariant 6.9.1 that S was either:

label=(a) Not $(1-\delta)$-boundary sparse at time t, or

lbbel=(b) $w(S, C \setminus S) > \lambda_{\max}$ at time t, or

lcbel=(c) $\mathrm{Vol}(S) > \frac{\lambda_{\max}}{\phi}$ at time t.

As at time t' we have that $w(S, C \setminus S)$ and no edge update was incident to S between t and t', case (b) is impossible.

As at time t' we have that $\mathrm{Vol}(S) \leq \frac{\lambda_{\max}}{\phi}$ and no edge update was incident to S between t and t', case (c) is impossible.

Let us thus look at case (a): At time t, $w(S, C \setminus S) \geq (1-\delta) \cdot \min\{w(S, V \setminus C), w(C \setminus S, V \setminus C)\}$.

Since no edge update was incident to S between t and t', we have that $w(S, V \setminus C)$ and $w(S, C \setminus S)$ remain unchanged between t and t'.

[3]This can either be because the cluster is poorly-connected, fragmented, or because t' is the first time after t where an update is incident to C

At time t', by boundary sparseness, we have that $w(S, C \setminus S) < (1-\delta)w(S, V \setminus C)$. This inequality also holds at time t, and since S is not $(1-\delta)$-boundary sparse at time t, we must have that at time t: $w(S, C \setminus S) \geq (1-\delta)w(C \setminus S, V \setminus C)$.

But, as we are not in case (b), at time t, we have that $\lambda_{\max} \geq w(S, C \setminus S) \geq (1-\delta)w(C \setminus S, V \setminus C)$. Therefore $w(C \setminus S, V \setminus C) < \frac{1}{1-\delta}\lambda_{\max} < 2\lambda_{\max}$ as $\delta \in [0, \frac{1}{2}]$.

We have two cases: (i): $E(C, V \setminus C)$ contains at least $2\lambda_{\max}$ many edges. As at least $2\lambda_{\max}$ many edges in $E(C, V \setminus C)$ have their endpoints in C unchecked, and there are strictly less than $2\lambda_{\max}$ in $E(C \setminus S, V \setminus C)$, at least one edge in $E(S, V \setminus C)$ has its endpoint in C unchecked. That endpoint is in S which concludes the proof.

(ii): $E(C, V \setminus C)$ contains at most $2\lambda_{\max}$ many edges. Since all of those edges have their endpoints unchecked, it remains to show that one of them has an endpoint in S. This is straightforward as S is boundary sparse at time t', which implies that $w(S, V \setminus C) > \frac{1}{1-\delta}w(S, C \setminus S) \geq 0$. □

Remark 6.10.1. *One can apply Lemma 6.10.9 for exactly one update between t and t', which is what we do when the cluster is "well-connected" both before and after the update.*

Corollary 6.10.10. *Assume that Invariant 6.9.1 holds at some time t. Assume the graph is subject to $t' - t$ updates. Then, in the partition $\mathcal{P}_1$ maintained by Algorithm 6.10.2 at time t' no "well-connected" cluster C contains a cut S that is $(1-\delta)$-boundary-sparse and such that $w(S, C \setminus S) \leq \lambda_{\max}$ and $\mathrm{Vol}(S) \leq \frac{\lambda_{\max}}{\phi}$. In other words, $\mathcal{P}_1$ is a pre-cluster decomposition.*

Proof. This is a direct consequence of Lemma 6.9.6 and Lemma 6.10.9, as since there are no cuts at time t' in G that are strictly smaller than $\lambda_{\min}$, and Algorithm 6.10.2 ensures no unchecked vertex remains in any "well-connected" cluster. □

Lemma 6.10.11. *The partition $\mathcal{P}_2$ maintained by the algorithm is a cluster decomposition associated to the pre-cluster decomposition $\mathcal{P}_1$.*

Proof. By Corollary 6.10.10, $\mathcal{P}_1$ is a pre-cluster decomposition. As we rerun the fragmenting algorithm on any cluster that has been modified in $\mathcal{P}_1$, this ensures $\mathcal{P}_2$ is the cluster decomposition associated to $\mathcal{P}_1$. □

This, together with Proposition 6.7.5 shows correctness.

Bounding the number of intercluster edges

We now take a look at the number of intercluster edges just before a new rebuild, and aim to bound that number. For that, as for the static algorithm we build below a forest of all the clusters that existed since the last rebuild, and analyze carefully the number of intercluster edges at each node of this forest.

Formally, let M be the initial number of interexpander edges (after running the first static expander decomposition) and R (for recourse) be the number of changes output by the dynamic expander decomposition. We aim to bound the number of intercluster edges after all of the updates before the next rebuild.

For that, let T be a time after an update. We build a forest F of all the clusters ever created between the last rebuild before T and time T. The parent of each cluster is the cluster it was cut off from when it was created. The roots are the expanders as output by the static expander decomposition. We will denote by $t(C)$ the *time* of cluster C, that is, the update at which C was split (if it is an internal node), or T otherwise, and $\partial C(t)$ the boundary of cluster C after update t. Let $w_t(A, B)$ be the number of edges between A and B at time t. Here, we assume that each update corresponds to one change output by the dynamic expander decomposition.

For each cluster C let $L(C)$ be its boundary-size when it was split into two (if it is an internal cluster) or its boundary-size at time T (if it is a leaf).

We will then, for the analysis, add marks to each cluster in $\mathcal{P}_1$ as follows: A cluster C that is a root gets mark $\oplus$ if $L(C) \geq 3\lambda_{\max}$ and $\odot$ otherwise. Then, in BFS fashion, each cluster C whose parent is marked $\oplus$ gets marked $\oplus$ if $L(C) \geq 3\lambda_{\max}$, and $\odot$ otherwise. If its parent is marked $\odot$, it gets marked $\oplus$ if $L(C) \geq 6\lambda_{\max}$, and $\odot$ otherwise

Intuitively, a $\oplus$ labeled node is a cluster that is "well-connected" at the time it is split in two or at time T, and a $\odot$ is a "poorly-connected" cluster at time T.

The goal is to show that the sum of the labels of the leaves is at most $O(\frac{M+R}{\delta})$. As $M = O(m\phi)$ and $R = O(m\phi)$, this will yield at most $O(\frac{m\phi}{\delta})$ many intercluster edges, which is crucial for bounding the recourse, as well as the running time (which depends on the number of intercluster edges).

We look at the following potential:

$$\Phi(C) = \max\{0, \partial C(t(C)) - 2.1\lambda_{\max}\}$$

and the total potential:

$$\Phi\{\mathcal{C}\} = \sum_{c \in \mathcal{C}} \Phi(C)$$

Let C_0 be a node of F that is marked $\oplus$ that is either a root of F or has a parent labeled $\odot$. We start with $\mathcal{C} = \{C_0\}$ and at each step, we will replace one element from $\mathcal{C}$ marked $\oplus$ with its two children until all elements of $\mathcal{C} = \mathcal{C}_f$ are either leaves of F or marked $\odot$.

Lemma 6.10.12. *Let C be a node in F marked $\oplus$ and $S, C \setminus S$ its children, where S is $(1-\delta)$-boundary sparse in C. Then we have that $\partial C(t(C)) \geq \max\{\partial S(t(C)), \partial(C \setminus S)(t(C))\} + \frac{\delta\lambda_{\min}}{2}$.*

Proof. In this proof, all quantities are considered at time $t(C)$.

Since S is a $(1-\delta)$-boundary-sparse cut, we have that $w(S, C \setminus S) \leq (1-\delta)\min\{w(S, V \setminus C), w(C \setminus S, V \setminus C)\}$. Therefore, $\partial S = w(S, V \setminus C) + w(S, C \setminus S) \leq w(S, V \setminus C) + (1-\delta)w(C \setminus S, V \setminus C) \leq \partial C - \delta w(C \setminus S, V \setminus C)$.

But $w(C \setminus S, V \setminus C) + w(C \setminus S, S) = \partial(C \setminus S) \geq \lambda_{\min}$. Since $w(C \setminus S, V \setminus C) > w(C \setminus S, S)$ (by boundary-sparsity of S), we have that $w(C \setminus S, V \setminus C) \geq \frac{\lambda_{\min}}{2}$ and the result follows by symmetry for $\partial(C \setminus S)$. □

Lemma 6.10.13. *Let $C \in F$ be a set in $\mathcal{C}$ replaced with S and $C \setminus S$. Let $u(C)$, $u(S)$ and $u(C \setminus S)$ be the number of updates to S and $C \setminus S$ respectively for updates between $t(C)$ (excluded) and $t(S), t(C \setminus S)$ (included).*

Then $\Phi(S) + \Phi(C \setminus S) \leq \Phi(C) - \frac{\delta\lambda_{\min}}{2} + u(S) + u(C \setminus S) + 2u(C)$.

Proof. Recall that $\partial C(t(C)) \geq 3\lambda_{\max}$, which imply that $\Phi(C) \geq 0.9\lambda_{\max}$. We then have five cases:

- if both $\partial S(t(S)), \partial(C \setminus S)(t(C \setminus S)) \leq 2.1\lambda_{\max}$:

$$\Phi(S) + \Phi(C \setminus S) = 0 \leq \Phi(C) - \frac{\delta\lambda_{\max}}{2}$$

  as $\Phi(C) - \frac{\delta\lambda_{\max}}{2} \geq 0.9\lambda_{\max} - \frac{\delta\lambda_{\max}}{2} > 0$.

- if both $\partial S(t(S)), \partial(C \setminus S)(t(C \setminus S)) > 2.1\lambda_{\max}$, and S is a cut that stems from the recourse of the expander decomposition or other updates to the partition:

$$\begin{aligned}\Phi(S) + \Phi(C \setminus S) &= \partial S(t(S)) + \partial(C \setminus S)(t(C \setminus S)) - 2 \times 2.1\lambda_{\max} \\ &\overset{(i)}{\leq} \partial S(t(C)) + \partial(C \setminus S)(t(C)) - 2 \times 2.1\lambda_{\max} + u(S) + u(C \setminus S) \\ &\leq \partial C(t(C)) + 2w_{t(C)}(S, C \setminus S) - 2 \times 2.1\lambda_{\max} + u(S) + u(C \setminus S) \\ \leq \Phi(C) + 2\lambda_{\max} + 2u(C) - 2.1\lambda_{\max} + u(S) + u(C \setminus S) &\leq \Phi(C) - \frac{\delta\lambda_{\max}}{2} + u(S) + u(C \setminus S) + 2u(C)\end{aligned}$$

  Where (i) stems from the fact that between times $t(C)$ and $t(S)$, at most $u(S)$ many edges got added to S and therefore ∂S.

- if both $\partial S(t(S)), \partial(C \setminus S)(t(C \setminus S)) > 2.1\lambda_{\max}$, and S is a $(1-\delta)$-boundary-sparse cut of cut-size at most $\lambda_{\max}$:

$$\begin{aligned}\Phi(S) + \Phi(C \setminus S) &= \partial S(t(S)) + \partial(C \setminus S)(t(C \setminus S)) - 2 \times 2.1\lambda_{\max} \\ &\overset{(i)}{\leq} \partial S(t(C)) + \partial(C \setminus S)(t(C)) - 2 \times 2.1\lambda_{\max} + u(S) + u(C \setminus S) \\ &\leq \partial C(t(C)) + 2w_{t(C)}(S, C \setminus S) - 2 \times 2.1\lambda_{\max} + u(S) + u(C \setminus S) \\ \leq \Phi(C) + 2\lambda_{\max} - 2.1\lambda_{\max} + u(S) + u(C \setminus S) &\leq \Phi(C) - \frac{\delta\lambda_{\max}}{2} + u(S) + u(C \setminus S)\end{aligned}$$

  Where (i) stems from the fact that between times $t(C)$ and $t(S)$, at most $u(S)$ many edges got added to S and therefore ∂S.

- if wlog $\partial S(t(S)) \leq 2.1\lambda_{\max}$ and $\partial(C \setminus S)(t(C \setminus S)) > 2.1\lambda_{\max}$, and S is a $(1-\delta)$-boundary-sparse cut:

$$\begin{aligned}\Phi(S) + \Phi(C \setminus S) &= 0 + \partial(C \setminus S)(t(C \setminus S)) - 2.1\lambda_{\max} \\ &\leq 0 + \partial(C \setminus S)(t(C)) + u(C \setminus S) - 2.1\lambda_{\max} \\ &\leq \partial C(t(C)) - w_{t(C)}(S, V \setminus C) + w_{t(C)}(S, C \setminus S) + u(C \setminus S) - 2.1\lambda_{\max} \\ &\overset{(ii)}{\leq} \Phi(C) - \delta w_{t(C)}(S, V \setminus C) + u(C \setminus S)\end{aligned}$$

Where (ii) follows from the $(1-\delta)$ boundary-sparsity of S at time $t(C)$. We conclude using the following claim:

Claim 6.10.14. *For any cluster C and $(1-\delta)$-boundary sparse cut $S \subsetneq C$, we have that $w(S, V \setminus C) \geq \frac{\lambda_{\min}}{2}$.*

Proof. Assume by contradiction that we have $w(S, V \setminus C) < \frac{\lambda_{\min}}{2}$. We are going to show that in that case, $\partial S < \lambda_{\min}$ holds, a contradiction. By boundary sparsity, we have that $w(S, C \setminus S) < (1-\delta)w(S, V \setminus C) < (1-\delta)\frac{\lambda_{\min}}{2}$, and therefore, $\partial S = w(S, C \setminus S) + w(S, V \setminus C) < \lambda_{\min}$. □

- if wlog $\partial S(t(S)) \leq 2.1\lambda_{\max}$ and $\partial(C \setminus S)(t(C \setminus S)) > 2.1\lambda_{\max}$, and S is a cut that stems from the recourse of the expander decomposition or other updates to the partition, that is not boundary-sparse:

$$\begin{aligned}
\Phi(S) + \Phi(C \setminus S) &= 0 + \partial(C \setminus S)(t(C \setminus S)) - 2.1\lambda_{\max} \\
&\leq 0 + \partial(C \setminus S)(t(C)) + u(C \setminus S) - 2.1\lambda_{\max} \\
&\leq \partial C(t(C)) - w_{t(C)}(S, V \setminus C) + w_{t(C)}(S, C \setminus S) + u(C \setminus S) - 2.1\lambda_{\max} \\
&\overset{(ii)}{\leq} \Phi(C) - \delta\lambda_{\max} + 2u(C) + u(C \setminus S)
\end{aligned}$$

Where (ii) follows from the the fact that S is non-boundary sparse and thus $u(C) \geq w_{t(C)}(S, C \setminus S) \geq (1-\delta)w_{t(C)}(S, V \setminus C)$ and thus $u(C) \geq \frac{\lambda_{\min}}{2}$.

□

Lemma 6.10.15. *The sum of all the labels of the leaves of F does not exceed $O\left(\frac{M+R}{\delta}\right)$.*

Proof. Let us consider one cluster C_0 be a node of F that is marked $\oplus$ that is either a root of F or has a parent labeled $\odot$, and we consider all elements of F that are reachable form C_0 only through paths with innner nodes marked $\oplus$. We start with $\Phi(C_0) = O(\partial C_0(t_0) + u(C_0))$, where t_0 is the time of the last rebuild. By Lemma 6.10.13, the potential decreases by at least $\frac{\delta\lambda_{\min}}{2} - u(S) - u(C \setminus S) + 2u(C)$ every time we replace a cluster C by its children S and $C \setminus S$. Therefore, after X many replacements each of which increases the number of clusters by 1, the potential is at most $\Phi(C_0) - X\frac{\delta\lambda_{\min}}{2} + 3\sum_{C \in F(C_0)} u(C) \leq \Phi(C_0) + U(C_0) - X\frac{\delta\lambda_{\min}}{2}$, where $F(C_0)$ is the component of C_0 with inner nodes marked $\oplus$ in F and $U(C_0) = 3\sum_{C \in F(C_0)} u(C)$. The potential being positive, we have that $X = O\left(\frac{\Phi(C_0)+U(C_0)}{\delta\lambda_{\min}}\right)$, and thus we have $O\left(\frac{\Phi(C_0)+U(C_0)}{\delta\lambda_{\min}}\right)$ many clusters descendents of C_0 in F as described above.

Let $\mathcal{C}_f(C_0)$ be the set of all leaves in the component of C_0 in F. Recall that $L(C) = \partial C(t(C)$. We then have:

$$\sum_{C \in \mathcal{C}_f(C_0)} L(C) = \sum_{\substack{C \in \mathcal{C}_f(C_0) \\ L(C) \leq 2.2\lambda_{\max}}} L(C) + \sum_{\substack{C \in \mathcal{C}_f(C_0) \\ L(C) > 2.2\lambda_{\max}}} L(C)$$

In the first sum, we know the that the total number of clusters is at most $O\left(\frac{\Phi(\mathcal{C}_0)+U(C_0)}{\delta\lambda_{\min}}\right)$. For the second sum, note that if $L(C) \geq 2.2\lambda_{\max}$, then $L(C) = O(\Phi(C))$.

$$\leq O\left(\frac{\Phi(\mathcal{C}_0)+U(C_0)}{\delta\lambda_{\min}}\right)2.2\lambda_{\max} + \sum_{\substack{C\in\mathcal{C}_f(C_0)\\ L(C)>2.2\lambda_{\max}}} O(\Phi(C))$$
$$\leq O\left(\frac{\Phi(\mathcal{C}_0)+U(C_0)}{\delta}\right) + O(\Phi(\mathcal{C}_f(C_0))) = O\left(\frac{\Phi(\mathcal{C}_0)+U(C_0)}{\delta}\right)$$

For the last equality, recall that the sum of potentials of the two children of a node in F is less than the potential of the parent to which we add the number of updates made to the children. It follows by induction that for any set of descendants of C_0 such that none is an ancestor of the other, the sum of their potentials is less than the potential of $C_0 + U(C_0)$. As the clusters in $C_f(C_0)$ fulfill this condition, it follows that $\sum_{\substack{C\in\mathcal{C}_f(C_0)\\ \partial C>2.2\lambda_{\max}}} O(\Phi(C)) = O(\Phi(\{C_0\}+U(C_0))$.

Summing over all such choices of C_0 in F, we get that the sum of the boundaries of all leaves of F with a parent marked $\oplus$ does not exceed $O(\Phi(\{\mathcal{C}_0\}+U(C_0))$, where $\mathcal{C}_0$ is the set of all C_0 as described in the beginning of this proof.

Let us now estimate $\Phi(\mathcal{C}_0)$. In $\mathcal{C}_0$, we have the roots in F that are an output from the static expander decomposition, as well as nodes marked $\oplus$ whose parents are marked $\odot$. The sum of the boundaries of the expanders is M initially, and does not exceed $M+R$ when they are cut. The sum of the boudaries of the nodes marked $\oplus$ whose parents are marked $\odot$ is at most $O(R)$, as they have ancestors with boundary-size at most $3\lambda_{\max}$ while they have boundary-size at least $6\lambda_{\max}$. This boundary increase can only be due to the recourse R.

Therefore $\Phi(\mathcal{C}_0) = O(M+R)$.

Finally, notice that all other leaves of F have boundary at most $O(M+R)$ as well, as all nodes that are marked $\odot$ who are roots or have a node marked $\oplus$ have boundary at most $O(M+R)$ by the above analysis. The boundary of their children can only increase due to the partition changing, which is due to recourse, and thus the boundary-size of their descendents is at most $O(M+R)$ as well. □

Lemma 6.10.16. *The number of intercluster edges in $\mathcal{P}_1$ is $O\left(\frac{M+R}{\delta}\right)$.*

The number of intercluster edges in $\mathcal{P}_2$ (that is, edges labeled "intercluster" or "fragmented" in our algorithm) is $\tilde{O}\left(\frac{M+R}{\delta^3}\right)$.

Proof. The first statement is immediate from Lemma 6.10.15.

For the second statement, by Theorem 6.6.1, fragmenting a cluster increases its boundary-size by at most a $\frac{1}{\delta^2}\operatorname{poly}\log n$ factor. Applying that on all "poorly-connected" clusters in $\mathcal{P}_1$ gives the result. □

Running time

Lemma 6.10.17. *The total number of times, over the $O(\frac{m\phi}{\rho})$ updates to the input of Algorithm 6.10.2, we process an unchecked vertex is $O(\frac{M+R}{\delta}\frac{\lambda_{\max}}{\phi})$.*

Proof. Let us first estimate the number of nodes that get marked as unchecked overall (counting a node that has been unchecked multiple times with multiplicity):

We divide the nodes that were unchecked at some point in two categories: nodes that were adjacent to an edge update, and nodes that we adjacent to an edge that became an intercluster edge. Any edge that was an intercluster edge and then deleted belongs to the first category.

Nodes that were adjacent to an edge update are at most R many. Nodes that were adjacent to an edge that was cut are at most $O(\frac{M+R}{\delta})$ many, as there are $O(\frac{M+R}{\delta})$ many such edges in the decomposition $\mathcal{P}_1$ by Lemma 6.10.16, and any intercluster edge in a prior decomposition is either a deleted edge or an intercluster edge in $\mathcal{P}_1$, as we do not merge clusters.

In the worst case, all updates are made to a "well-connected" cluster in G, and thus we need to add $2\lambda_{\max}$ unchecked nodes. Moreover, while processing an unchecked node, we might find a cut that needs to be split from the original cluster. In that case, we have to process the node again, but this can happen at most $\frac{1}{\phi}$ times as per Lemma 6.9.14. □

Lemma 6.10.18. *The total number of times, over the $O(\frac{m\phi}{\rho})$ updates to the input of Algorithm 6.10.2, we run the fragmenting algorithm is $O(\frac{M+R}{\delta})$.*

Proof. Each intercluster edge in $\mathcal{P}_1$ can result in at most two runs of the fragmenting algorithm (one the cluster at each endpoint). Moreover, any edge update can also result in at most two runs of the fragmenting algorithm. Since there are at most $O(\frac{M+R}{\delta})$ of these edges, the result follows. □

Corollary 6.10.19. *The total recourse of the algorithm over the $O(\frac{m\phi}{\rho})$ updates to the input of Algorithm 6.10.2 is $\tilde{O}\left(\frac{m\phi}{\delta^3}\lambda_{\max}\right)$.*

Proof. At any given time, there are at most $O(\frac{m\phi}{\delta})$ many intercluster edges in $\mathcal{P}_1$, by Lemma 6.10.16. Since $\mathcal{P}_1$ only gets refined over time, this yields $O(\frac{m\phi}{\delta})$ recourse.

Moreover, there are at most $O(\frac{M+R}{\delta})$ calls to the fragmenting algorithm, and as many reversions of whatever changes these calls made. Each of these calls make at most $\tilde{O}(\frac{\lambda_{\max}}{\delta^3})$ changes by Theorem 6.6.1, hence the result. □

Corollary 6.10.20. *As $\lambda_{\max} = n^{o(1)}$ and $\phi = n^{-o(1)}$, the total running time of Algorithm 6.10.2 between two rebuilds, excluding recursive calls, is $(M+R)\cdot n^{o(1)} = m^{1+o(1)}$.*

Proof. As per Theorem 6.5.1, amortized update time and request time to LocalKCut is $\tilde{O}\left(\beta^4(\lambda_{\max}\nu)^{O(\beta)}\right)$ with $\beta = O(1)$, and therefore all calls to LocalKCut – outside of the fragmenting algorithm – need $\tilde{O}(\frac{M+R}{\delta}\beta^4(\lambda_{\max}\nu)^{O(\beta)}) = (M+R)\cdot n^{o(1)}$ time. This is also an upper bound to the number of nodes that need to be unchecked, as each LocalKCut is

ran on an unchecked vertex. Furthermore, we need to update the data structure, which takes amortized time $n^{o(1)}$ per update, by Theorem 6.11.1, as the graph has initial volume m and is subject to updates of volume $m\phi$ between two rebuilds. □

This proves the following lemma:

Lemma 6.10.1. *In Algorithm 6.10.2:*

1. *The amortized recourse*[4] *per updated edge is* $\tilde{O}\left(\frac{\rho}{\delta^3}\lambda_{\max}\right)$
2. *At every time step, and every level, if* $\tilde{m}$ *is the number of edges input on that level, the number of intercluster edges in* $\mathcal{P}_2$ *is at most* $\tilde{O}\left(\frac{\tilde{m}\phi}{\delta^3}\right)$*, and thus the recursive call will have as input at most* $\tilde{O}\left(\frac{\tilde{m}\phi}{\delta^3}\right)$ *edges.*
3. *The amortized update time is* $n^{o(1)}$ *on each level – excluding recursive calls.*

Proof. Item 1. is immediate from Corollary 6.10.19 as we can amortize the recourse over all updates between two rebuilds.

Item 2. is a direct application of Lemma 6.10.16, where we note that between two rebuilds, the decomposition is only refined, and thus the number of intercluster edges just before the next rebuild is an upper bound on the number of intercluster edges at any point since the last rebuild.

Item 3. is proven in Corollary 6.10.20. □

6.11 Data Structure

This section is dedicated to proving the following theorem:

Theorem 6.11.1. *Consider a fully-dynamic graph* G*, two (dynamic) partitions* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *of* G*.* G *can be subject to edge insertions or deletions, and* $\mathcal{P}_1$ *can only be subject to splits (that is, replace a cluster* $C \in \mathcal{P}_1$ *by* $C \setminus S$ *and* S *for some* $S \subsetneq C$*), while in* $\mathcal{P}_2$*, in addition to splits, clusters of volume at most* ν *(for some parameter* $\nu \in \mathbb{N}_+$*) are allowed to merge back to larger clusters.* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *are encoded by a labeling of the edges as "intracluster", "intercluster", and "fragmented", that is,* $\mathcal{P}_1$ *is formed by the connected components formed by looking only at the edges labeled "intracluster" and "fragmented", while* $\mathcal{P}_2$*, a refinement of* $\mathcal{P}_1$*, is formed by the connected components formed by looking only at the edges labeled "intracluster". Each vertex, if it is a vertex obtained by the contraction of other vertices, has a label representing the inner volume of the contracted cluster it represents.*

There exists a data structure that maintains the mirror graph $G_{\mathcal{P}_1}$ *and the contracted graph* $G/\mathcal{P}_2$ *of* G*.* $G/\mathcal{P}_2$ *should have, as vertex labels, the inner volume of the cluster the vertex represents. Over* U *many edge updates, it has* $\tilde{O}(m + U\nu)$ *running time (preprocessing + update handling). Moreover, for any request on a set* $S \subseteq C$ *for* $C \in \mathcal{P}_2$*, it can output in time* $\tilde{O}(\mathrm{Vol}(S))$ *whether or not* S *is* $(1-\delta)$*-boundary-sparse. It can also store for each node, edge and cluster a set of constant size of parameters accessible in constant time.*

[4] the recourse is the number of modifications to the edges of the contracted graph

We need a data structure that enables us to efficiently compute all the values and get the correct pointers at all times. This includes, for every cluster C, access to its boundary edges, and its boundary-size. We should also be able to compute efficiently, for any cut S of volume at most $O(\nu)$ inside C, whether or not it is boundary sparse, that is we need $w(S, C \setminus S)$ and $w(S, V \setminus C)$. We also need for each node to know in which cluster it is, and whether it is checked or unchecked.

For that, we have the following algorithm:

Algorithm 6.11.1.

Input: A fully-dynamic graph G under edge insertions, deletions, update of vertex and edge labels.

Maintained variables:

- *for each node:*
  - *a pointer to the cluster it is currently in*
  - *a pointer to each of the edges it is incident to*
  - *other node parameters*
- *for each edge:*
  - *a pointer to its endpoints*
  - *Whether it is an "intercluster", "intracluster" or "fragmented" edge*
  - *other edge parameters*
- *for each cluster C in $\mathcal{P}_1$ or $\mathcal{P}_2$:*
  - *a list of the nodes that constitute this cluster*
  - *the boundary-size of the cluster*
  - *the inner volume of the cluster*
  - *a list of all edges on its boundary*
  - *other cluster parameters*
- *for each cluster C in $\mathcal{P}_1$:*
  - *a mirror cluster $C' = G/(G \setminus C)$*
- *an instance of Minimum Spanning Forest (Theorem 6.5.5) on intracluster and fragmented edges which can output whether or not two nodes are in the same $\mathcal{P}_1$ component in $\tilde{O}(1)$ time.*
- *an instance of Minimum Spanning Forest (Theorem 6.5.5) on intracluster edges which can output whether or not two nodes are in the same $\mathcal{P}_2$ component in $\tilde{O}(1)$ time.*
- *a contracted graph $G/\mathcal{P}_2$*

Preprocessing:

1. *Instantiate the Minimum Spanning Forests*
2. *Compute the connected components (or clusters) of the graph G where only edges labeled "intracluster" or "fragmented" are considered, to get $\mathcal{P}_1$.*
3. *Compute the connected components (or clusters) of the graph G where only edges labeled "intracluster" are considered, to get $\mathcal{P}_2$.*
4. *Traverse each cluster in $\mathcal{P}_1$, creating its mirror cluster.*
5. *Create a node to represent each cluster in $\mathcal{P}_1$ or $\mathcal{P}_2$ and add pointers from and to all the nodes it contains.*
6. *Traverse each cluster in $\mathcal{P}_2$ to create the contracted graph.*

Handling a split (in either $\mathcal{P}_1$ or P_2 or both) specified by a set of edges changing their labels:

1. *Update the Minimum Spanning Forests*
2. *Choose an updated edge, explore in lockstep the connected component on each side of that edge (for both/either $\mathcal{P}_1$ and/or $\mathcal{P}_2$). Once one connected component S is completely uncovered, stop the exploration.*
3. *Create a new node for the new (smaller) cluster, and update all the pointers to the nodes it contains to the new cluster, as well as the edges on its boundary. This also updates the old cluster into the new bigger cluster.*
4. *Update the edges in the mirror graph if $\mathcal{P}_1$ changes.*
5. *Update the edges in the contracted graph if $\mathcal{P}_2$ changes.*
6. *Compute the boundary and inner volume of the smaller cluster, store them and use it to update the boundary and inner volume of the bigger cluster.*

Handling an edge deletion:

1. *Update the inner volume of the cluster it is in, and the corresponding label in the contracted graph.*
2. *Update the mirror cluster in which the edge is, if necessary*
3. *Update the Minimum Spanning forest*
4. *If the two endpoints are now in different connected components of the minimum spanning forest, run the steps in "Handling a split specified by a set of edges changing their label".*

Handling an "intracluster" or "fragmented" edge insertion in $\mathcal{P}_1$, or an "intracluster" edge insertion in $\mathcal{P}_2$:

1. *In $\mathcal{P}_1$, if one of its endpoints is an isolated vertex, add it to the cluster of the other endpoint, updating the pointers accordingly*
2. *In $\mathcal{P}_2$, look if the two endpoints are in the same connected component. If they are not, explore in lockstep the connected components of both endpoints. When one component is completely explored, delete the corresponding cluster and add the vertices to the cluster of the other endpoint. Update the pointers accordingly.*
3. *Update the inner volume of the corresponding cluster and that volume in the contracted graph*
4. *Update the mirror graph accordingly*
5. *Update the Minimum Spanning Forest*

Handling any other update:

1. *Trivially update all data structures related to that update.*

We start by arguing that only allowing cluster splits (ignoring merging of components in $\mathcal{P}_2$ for the moment) allows us to ensure the update time can be amortized to the preprocessing time. We call the *union graph between two updates* t_1 and t_2 the graph that is the union of all edges that appears at least once between updates t_1 and t_2, that is, an edge that either exists already at update t_1, or is inserted at any point between times t_1 and t_2.

One main argument that is useful throughout is the fact that if we maintain this data structure on a decomposition of the graph, such that the only allowed operations are to rebuild from scratch the whole decomposition or to split a cluster such that splitting a cluster only takes time proportional to the smaller side of the cut, then the data structure can be maintained efficiently:

Lemma 6.11.2. *Let $\mathcal{P}$ be a (dynamic) partition of G, where G has U many updates and initially contains m edges. Suppose that $\mathcal{P}$ starts as $\mathcal{P} = \{V\}$, and can be subject to splits during those updates, that is, an element C in $\mathcal{P}$ can be replaced by $C \setminus S$ and S, for some $S \subsetneq C$. Let $\mathcal{D}$ be a data structure on G and $\mathcal{P}$ with preprocessing time mT such that it takes time $T \cdot \min\{\mathrm{Vol}(S), \mathrm{Vol}(C \setminus S)\}$ to handle a split, for some $T > 0$.*

Then over all the updates, maintaining $\mathcal{D}$ takes $\tilde{O}((m+U)T)$ time overall.

Proof. Let us consider the union graph over the updates. This graph has $m + U$ edges at most. For any set X, let $\mathrm{Vol}^*(X)$ be the volume of X in the union graph. Then, at any point in time, $\mathrm{Vol}(X) \leq \mathrm{Vol}^*(X)$.

Let us now consider a split of C into S and $C \setminus S$, and assume it happens after update t. Assume that, w.l.o.g., at time t, we have that $\mathrm{Vol}(S) \leq \mathrm{Vol}(C \setminus S)$. Then, at time t, we have that $\mathrm{Vol}(S) \leq \mathrm{Vol}^*(S)$ and $\mathrm{Vol}(C \setminus S) \leq \mathrm{Vol}^*(C \setminus S)$. In particular, $\mathrm{Vol}(S) \leq \mathrm{Vol}^*(C \setminus S)$ and thus $\mathrm{Vol}(S) \leq \min\{\mathrm{Vol}^*(S), \mathrm{Vol}^*(C \setminus S)\}$.

Hence, if we look at the union graph, each split costs at most the side of the smallest volume (up to a factor T). Whenever a split is made, assign to each edge on the smaller side of the cut

a cost of T, where the sum of all the costs is thus higher than the time spent processing the split. Consider the half-edges (also called stubs), where each half-edge of an edge is assigned to one of the incident vertices. Then the volume of a node is the number of half-edges assigned to it, and the volume of a subset is the number of half-edges assigned to one of its vertices. Since each half-edge can only be on the smallest side of the split $\log(m+U)$ times, each half-edge is assigned at most cost $\log(m+U)T$. Summing over all half-edges and adding the preprocessing time concludes the proof. □

Corollary 6.11.3. *Let $\mathcal{P}$ be a dynamic partition of G, where G has updates of total volume U and initially contains m edges. Suppose that $\mathcal{P}$ starts as $\mathcal{C} = \{V\}$ and can be subject to splits during those updates, as well as merges where the volume of the smaller cluster being merged is at most ν.*

Let $\mathcal{D}$ be a data structure on G and $\mathcal{P}$ that with preprocessing time mT such that it takes time $T \cdot \min\{\mathrm{Vol}(S), \mathrm{Vol}(C \setminus S)\}$ to handle a split, for some $T > 0$. Then, over all updates, maintaining $\mathcal{D}$ takes $\tilde{O}((m+U\nu)T)$ time overall.

Proof. Whenever a merge is observed, simply simulate it by deleting the smaller cluster to merge, and inserting it again inside the cluster it is merged to. □

Lemma 6.11.4. *Algorithm 6.11.1 handles a split of a cluster C into S and $C \setminus S$ with $\mathrm{Vol}(S) \leq \mathrm{Vol}(C \setminus S)$ in $\tilde{O}(\mathrm{Vol}(S))$ time.*

Proof. Let us first consider that the split is encoded by a change of labels of the edges:

Step 1 takes $O(\mathrm{Vol}(S)\log^4 n)$ by Theorem 6.5.5, as there are at most $\mathrm{Vol}(S)$ many edges deleted from the instance. Steps 2, 3 4 and 5 take $O(\mathrm{Vol}(S))$ time as well. In Step 6, computing the boundary and the inner volume of S takes $O(\mathrm{Vol}(S))$ time, and the inner volume and boundary-size of the new bigger cluster $C \setminus S$ can be computed in $O(\mathrm{Vol}(S))$ time from the values for C using $\mathbf{vol}^{\mathsf{in}}(C) = \mathbf{vol}^{\mathsf{in}}(C \setminus S) + \mathbf{vol}^{\mathsf{in}}(S) + w(S, C \setminus S)$ and $\partial C = \partial S + \partial C \setminus S - w(S, C \setminus S)$.

If the split is encoded as an intracluster edge deletion, note that the first three steps take $\tilde{O}(1)$ time, and the fourth step is the same as the ones analyzed above. □

Lemma 6.11.5. *For $\mathcal{P}_1$, Algorithm 6.11.1 handles any update other that a split in $\tilde{O}(1)$ time.*

Proof. This is immediate as every step takes $\tilde{O}(1)$ time. □

Lemma 6.11.6. *For $\mathcal{P}_2$, Algorithm 6.11.1 handles any update other that a split in $\tilde{O}(\nu)$ time.*

Proof. This is immediate as every step takes $\tilde{O}(1)$ time, apart from when the update merges two clusters back together, which takes time $O(\bar{\nu})$ where $\bar{\nu}$ is the volume of the smaller of the two clusters. as $\bar{\nu} \leq \nu$, the result follows. □

These three lemmata together with Lemma 6.11.2 prove the running time claimed in Theorem 6.11.1. It remains to show that we can compute whether a cut is $(1-\delta)$boundary sparse efficiently.

Lemma 6.11.7. *With our data structure, checking whether a cut S in cluster C is $(1-\delta)$-boundary sparse takes $O(\mathrm{Vol}(S))$ time.*

Proof. We need to compute three quantities: $w(S, C \setminus S)$, $w(S, V \setminus C)$ and $w(C \setminus S, V \setminus C)$. The first two ones can be trivially computed in $O(\mathrm{Vol}(S))$. The third one can be computed using the value stored at C as ∂C: $\partial C = w(C \setminus S, V \setminus C) + w(S, V \setminus C)$. □

6.12 Dealing with Weighted Graphs

In this section, we will present a way to sample a graph where edges have integer weights. For that, we extend the weight function described above so that $w(e)$ is the weight of edge e. Again, we will rely on the edge-sampling approach of Karger [93] to solve the problem.

The idea is as follows: If we estimate the minimum cut to be between $\lambda_i = 1.1^i$ and $\lambda_{i+1} = 1.1^{i+1}$, sampling for each edge e a binomial random variable $X(e)$ with parameters $w(e)$ and $p_i = \frac{54 \ln n}{\epsilon^2 \lambda_i}$, and define $\tilde{G}_i$ to be the graph obtained from G by replacing each edge with $X(e)$ parallel edges. Then, with high probability, every cut in $\tilde{G}_i$ has a value that is at most a $(1-\epsilon)$ factor away from its expectation.

However, a binomial distribution cannot trivially be sampled efficiently. The naive strategy, that is, to sample $w(e)$ many Binomial random variables with probability p_i, takes $O(w(e))$ time, which can be too large. Instead, we note that we only need to preserve the cuts whose value is at most $\lambda_{\max}$ in $\tilde{G}_i$. Indeed, for any cut that has value strictly larger than $\lambda_{\max}$ in $\tilde{G}_i$, we do not mind that its value is a $(1-\epsilon)$ factor away from its expected value, we only need that it stays above $\lambda_{\max}$ so that our algorithm discards it. Therefore, we only need to exactly compute $Y(e)$, the random variable defined as follows: $Y(e) = X(e)\mathbb{1}[X(e) \leq \lambda_{\max}] + (\lambda_{\max}+1)\mathbb{1}[X_e > \lambda_{\max}]$, and define G_i to be the graph obtained from G by replacing each edge with $Y(e)$ parallel edges.

The rest of this section formalizes these ideas.

Definition 6.12.1 (Karger sparsification with integer weights)**.** *In the case of integer weights, the sampled graph G_i is defined as follows. Let $p_i = \frac{54 \ln n}{\epsilon^2 \lambda_i}$ and $\lambda_{\max} = 1.1(1+\epsilon)\frac{54 \ln n}{\epsilon^2}$. For every edge $e \in E(G)$, let $X(e)$ be an independent Binomial random variable of parameters $w(e), p_i$. Define $Y(e)$ as follows:*

$$Y(e) = \begin{cases} X(e) & \text{if } X(e) \leq \lambda_{\max} \\ \lambda_{\max}+1 & X(e) > \lambda_{\max} \end{cases}$$

Let $\tilde{G}_i$ be the graph obtained from G by replacing each edge with $X(e)$ parallel edges, and G_i to be the graph obtained from G by replacing each edge with $Y(e)$ parallel edges.

Note that we will only compute $Y(e)$ for every edge e, and thus only compute G_i, not $\tilde{G}_i$. However, $\tilde{G}_i$ and X will be useful for our analysis.

Lemma 6.12.2. *In G_i, let S be a cut of value $c \leq \lambda_{\max}$. Then, with high probability, its value in G is in $[(1-\epsilon)\frac{c}{p_i}, (1+\epsilon)\frac{c}{p_i}]$. Moreover, if S also is the minimum cut of G_i, then with high probability, it is a minimum cut of G.*

Proof. If $c \leq \lambda_{\max}$, then for every cut-edge e, we have that $Y(e) \leq \lambda_{\max}$, and thus $X(e) \leq \lambda_{\max}$. Therefore, the value of the cut is the same in G_i and $\tilde{G}_i$. By Theorem 6.2.5, we thus have that if c' is the value of the cut S in G, then with high probability, $(1-\epsilon)p_i c' \leq c \leq (1+\epsilon)p_i c'$. This translates to $c' \leq \frac{c}{p_i(1-\epsilon)} \leq \frac{c}{p_i}(1+\epsilon)$ and $c' \geq \frac{c}{p_i(1+\epsilon)} \geq \frac{c}{p_i}(1-\epsilon)$.

In the case where S is also the minimum cut of G_i, note that any other cut S' falls in one of two cases: Either all its cut edges e satisfy $Y(e) \leq \lambda_{\max}$ and thus $\partial_{\tilde{G}_i} S' = \partial_{G_i} S' \geq \partial_{G_i} S = \partial_{\tilde{G}_i} S$. Or there exists a cut edge e such that $Y(e) = \lambda_{\max} + 1$ and thus $\partial_{\tilde{G}_i} S' \geq X(e) > \lambda_{\max} \geq \partial_{G_i} S = \partial_{\tilde{G}_i} S$. Therefore S is also a minimum cut in $\tilde{G}_i$, and by Theorem 6.2.5, it corresponds to the minimum cut in G. □

Lemma 6.12.3. *We can sample G_i in $O(m\lambda_{\max}^2)$ time.*

Proof. For each edge e of weight $w(e)$, we will show that we can sample $Y(e)$ in $\lambda_{\max}^2$ time. For that, note that:

$$\mathbb{P}\left(Y(e) = k\right) = \begin{cases} \binom{w(e)}{k} {p_i}^k (1-p_i)^k & \text{if } k \leq \lambda_{\max} \\ \lambda_{\max} + 1 & \text{otherwise} \end{cases}$$

Writing $\binom{w(e)}{k} = \frac{w(e)\times\cdots\times(w(e)-k+1)}{k\times\cdots\times 1}$, we can see that every $\binom{w(e)}{k} {p_i}^k (1-p_i)^k$ can be computed in $O(k) = O(\lambda_{\max})$ time for every $k \leq \lambda_{\max}$. Hence, computing every probability $\mathbb{P}(Y(e) = k)$ for $k \leq \lambda_{\max}$ takes $O(\lambda_{\max}^2)$ time, and $\mathbb{P}(Y(e) = \lambda_{\max} + 1)$ can be deduced in $O(\lambda_{\max})$ time.

To sample $Y(e)$, all we have to do is thus sample uniformly at random $y(e)$ in $[0, 1]$, and set $Y(e)$ to be the smallest integer j such that $\sum_{i=0}^{j} \mathbb{P}(Y(e) = k) > y(e)$. □

We now discuss how to sample G_i in the case where real weights are allowed, but no parallel edges are. First note that any edge of weight at most $\frac{\epsilon\lambda_i}{n^2}$ can be discarded. Indeed, for any cut S, at most n^2 edges cross the cut, and thus at most $\epsilon\lambda_i$ weight could have been discarded. This preserves the $(1+\epsilon)$-approximation.

We then replace each weight with the closest integer multiple of $\frac{1}{x_i}$, where $x_i = \left\lceil \frac{n^2}{\epsilon\lambda_i} \right\rceil$. Indeed, this ensures that the difference between any old weight and newly assigned weight is at most $\frac{\epsilon\lambda_i}{n^2}$. Similarly to before, this ensures that it only affects the cut-size of the cut by a $(1+\epsilon)$ factor.

Finally, we multiply each edge weight by x_i to get integer weights, sample as above with $p_i = \frac{54 \ln n}{\epsilon^2 \lambda_i x_i}$. Notice how we reduce p_i to correct for the fact that the weights got multiplied by x_i.

Definition 6.12.4 (Karger sparsification with real weights)**.** *In the case of real weights, the sampled graph G_i is defined as follows. Define $x_i = \left\lceil \frac{n^2}{\epsilon\lambda_i} \right\rceil$ and $\hat{w}_i(e) \in \mathbb{N}$ such that $\frac{\hat{w}_i(e)}{x_i} \leq w(e) < \frac{\hat{w}_i(e)+1}{x_i}$. Let $p_i = \frac{54 \ln n}{\epsilon^2 \lambda_i x_i}$ and $\lambda_{\max} = 1.1(1+\epsilon)\frac{54 \ln n}{\epsilon^2}$. For every edge $e \in E(G)$, let $X(e)$ be an independent Binomial random variable of parameters $\hat{w}_i(e), p_i$. Define $Y(e)$ as follows:*

$$Y(e) = \begin{cases} X(e) & \text{if } X(e) \leq \lambda_{\max} \\ \lambda_{\max} + 1 & X(e) > \lambda_{\max} \end{cases}$$

Let $\hat{G}_i$ be G with $\hat{w}_i$ as a weight function. Let $\tilde{G}_i$ be the graph obtained from G by replacing each edge with $X(e)$ parallel edges, and G_i to be the graph obtained from G by replacing each edge with $Y(e)$ parallel edges.

Lemma 6.12.5. *In G_i, let S be a cut of value $\lambda_{\min} \leq c \leq \lambda_{\max}$. Then, with high probability, its value in G is in $[(1-\epsilon)\frac{c}{p_i x_i}, (1+\epsilon)^3 \frac{c}{p_i x_i}]$. Moreover, if S also is the minimum cut of G_i, then with high probability, it is a $(1+\epsilon)^2$-approximation of the minimum cut of G.*

Proof. If $c \leq \lambda_{\max}$, then for every cut-edge e, we have that $Y(e) \leq \lambda_{\max}$, and thus $X(e) \leq \lambda_{\max}$. Therefore, the value of the cut is the same in G_i and $\tilde{G}_i$. By Theorem 6.2.5, we thus have that if c' is the value of the cut S in $\hat{G}_i$, then with high probability, $(1-\epsilon)p_i c' \leq c \leq (1+\epsilon)p_i c'$. This translates to $c' \leq \frac{c}{p_i(1-\epsilon)} \leq \frac{c}{p_i}(1+\epsilon)$ and $c' \geq \frac{c}{p_i(1+\epsilon)} \geq \frac{c}{p_i}(1-\epsilon)$.

Let us now compare c' with c'', the value of the cut S in G. There are at most n^2 cut edges in S, and thus:

$$\frac{c'}{x_i} = \sum_{e \in E(S, V\setminus S)} \frac{\hat{w}_i(e)}{x_i} \leq \sum_{e \in E(S, V\setminus S)} w(e) = c'' \leq \sum_{e \in E(S, V\setminus S)} \frac{\hat{w}_i(e)+1}{x_i} \leq \frac{c'+n^2}{x_i} \leq \frac{c'}{x_i} + \epsilon\lambda_i$$

However, $c' \geq \frac{c}{p_i}(1-\epsilon) \geq \frac{\lambda_{\min}\epsilon^2\lambda_i x_i}{54 \ln n} \geq \frac{(1-\epsilon)54 \ln n \epsilon^2 \lambda_i x_i}{\epsilon^2 54 \ln n} \geq (1-\epsilon)\lambda_i x_i$ and thus $\frac{c'}{x_i} \geq (1-\epsilon)\lambda_i$. Hence $\frac{c'}{x_i} \leq c'' \leq \frac{c'}{x_i}(1+\frac{\epsilon}{1-\epsilon})$. We therefore have:

$$\begin{cases} c'' \geq \frac{c'}{x_i} \geq \frac{c}{p_i x_i}(1-\epsilon) \\ c'' \leq \frac{c'}{x_i}(1+\frac{\epsilon}{1-\epsilon}) \leq \frac{c}{p_i x_i (1-\epsilon)}(1+\frac{\epsilon}{1-\epsilon}) \leq \frac{c}{p_i x_i}(1+\epsilon)^3 \end{cases}$$

In the case where S is also the minimum cut of G_i, note that any other cut S' falls in one of two cases: Either all its cut edges e satisfy $Y(e) \leq \lambda_{\max}$ and thus $\partial_{\tilde{G}_i} S' = \partial_{G_i} S' \geq \partial_{G_i} S = \partial_{\tilde{G}_i} S$. Or there exists a cut edge e such that $Y(e) = \lambda_{\max} + 1$ and thus $\partial_{\tilde{G}_i} S' \geq X(e) > \lambda_{\max} \geq \partial_{G_i} S = \partial_{\tilde{G}_i} S$. Therefore S is also a minimum cut in $\tilde{G}_i$, and by Theorem 6.2.5, it corresponds to the minimum cut in $\hat{G}_i$. Let a' be the value of the minimum cut C in $\hat{G}_i$, a'' the value of C in G, c' the value of S in $\hat{G}_i$, and c'' the value of S in G. We then have that $c' \leq a'$ with high probability as S is the minimum cut of $\hat{G}_i$, $c'' \leq \frac{c'}{x_i}(1+\frac{\epsilon}{1-\epsilon}) \leq \frac{c'}{x_i}(1+\epsilon)^2 \leq \frac{a'}{x_i}(1+\epsilon)^3$. We also have that $a'' \geq \frac{a'}{x_i}$ as we round down the weights to get $\hat{G}_i$, which concludes the proof. □

Observation 6.12.6. *Note that Lemma 6.12.3 still holds in the weighted case, as we simply add a rounding step before applying the same algorithm, and where the value of $w(e)$ for each edge e does not intervene in the proof.*

Observation 6.12.7. *Note that after sampling, our graph is unweighted, and thus, in all other sections, the graph can be assumed to be unweighted.*

We now discuss how to find a dynamic algorithm that finds an approximate minimum cut in weighted graphs, where U and L are respectively the maximum and minimum weight allowed for the algorithm:

Algorithm 6.12.1 (Global Weighted Dynamic Algorithm)**.** *Define $O(\log_{1.1}(n^2 \cdot \frac{U}{L}))$ many estimates of λ: $\lambda_0, \ldots, \lambda_{\log_{1.1}(n^2 \cdot \frac{U}{L})}$ where $\lambda_i = L \cdot 1.1^i$ for every $i \in [\log_{1.1}(n^2 \cdot \frac{U}{L})]$. Define for every i, G_i, p_i and x_i as described in Definition 6.12.4. Run Algorithm 6.10.1 on every G_i with $\lambda_{\min} = (1-\epsilon)^2 \frac{54 \ln n}{\epsilon^2}, \lambda_{\max} = 1.1(1+\epsilon)\frac{54 \ln n}{\epsilon^2}$.*

For initialization, initialize the instances in increasing order of index i, and stop whenever an algorithm reports a value in their range.

Feed every update to every instance in increasing order of index i, initializing new instances whenever needed. Stop whenever an algorithm reports a value in their range.

Let j be the smallest i such that the return value of Algorithm 6.10.1 on G_i is in $[\lambda_{\min}, \lambda_{\max}]$. Return this value multiplied by $\frac{1}{p_j x_j}$.

Lemma 6.12.8. *For every i such that $\lambda_i \leq \lambda$, if the return value of Algorithm 6.10.1 on G_i is smaller than $\lambda_{\max}$, then this value multiplied by $\frac{1}{p_i x_i}$ is at most a $(1+\epsilon)^4$ multiplicative factor away from the minimum cut value in G.*

Proof. The argument is as follows: we will show that if $\lambda_i \leq \lambda$ and the return value of Algorithm 6.10.1 is smaller than $\lambda_{\max}$, then in G_i the minimum cut is of value between $\lambda_{\min}$ and $\lambda_{\max}$. This in turn ensures that the cut found by Algorithm 6.10.1 is a $(1+\epsilon)$-approximation of the minimum cut of G_i, and the fact that $\lambda_i \leq \lambda$ ensures that p_i is large enough – that is, we sampled enough edges – so that we can ensure with high probability that the found cut in an approximate minimum cut of $\hat{G}_i$, which in turn is an approximate minimum cut of G. Let μ be the value of the minimum cut in $\hat{G}_i$. We have that $\mu \leq \lambda x_i \leq \mu + n^2$ and thus $\mu \geq \lambda x_i - n^2 \geq \lambda x_i - \epsilon \lambda_i x_i \geq (1-\epsilon)\lambda_i x_i$. We have that $p_i = \frac{54 \ln n}{\epsilon^2 \lambda_i x_i} \geq \frac{54 \ln n}{\epsilon^2 \lambda x_i}$, and thus by Theorem 6.2.5, with high probability the minimum cut in G_i is at most a $(1+\epsilon)$ or $(1-\epsilon)$ factor away from $p_i \mu$, and thus is at least $(1-\epsilon)p_i\mu \geq (1-\epsilon)^2 \lambda x_i \frac{54 \ln n}{\epsilon^2 \lambda_i x_i} \geq \lambda_{\min}$.

If moreover Algorithm 6.10.1 finds a cut of value at most $\lambda_{\max}$, this ensures that the minimum cut of G_i is of cut-size at most $\lambda_{\max}$, and thus ensures that Algorithm 6.10.1 found a $(1+\epsilon)$-aproximate minimum cut of G_i. Multiplying the cut value by $\frac{1}{p_i x_i}$ gives the desired result, by Lemma 6.12.5. □

6.13 Acknowledgements

Funded by the European union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (MoDynStruct, No. 101019564) and the Austrian Science Fund (FWF) grant DOI 10.55776/I5982. For open access purposes, the author has applied a CC BY public copyright license to any author-accepted manuscript version arising from this submission.

References

[26] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Almost-Linear Time Algorithms for Decremental Graphs: Min-Cost Flow and More via Duality". In: *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*. IEEE, 2024, pp. 2010–2032. DOI: 10.1109/FOCS61266.2024.00120. URL: https://doi.org/10.1109/FOCS61266.2024.00120.

[39] Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Micha Pilipczuk. "Designing FPT algorithms for cut problems using randomized contractions". In: *SIAM Journal on Computing* 45.4 (2016), pp. 1171–1229.

[58] Antoine El-Hayek, Monika Henzinger, and Jason Li. "Deterministic and Exact Fully-dynamic Minimum Cut of Superpolylogarithmic Size in Subpolynomial Time". In: *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*. Ed. by Kasper Green Larsen and Barna Saha. SIAM, 2026, pp. 613–663. DOI: 10.1137/1.9781611978971.25. URL: https://doi.org/10.1137/1.9781611978971.25.

[59] Antoine El-Hayek, Monika Henzinger, and Jason Li. "Fully Dynamic Approximate Minimum Cut in Subpolynomial Time per Operation". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*. Ed. by Yossi Azar and Debmalya Panigrahi. SIAM, 2025, pp. 750–784. DOI: 10.1137/1.9781611978322.22. URL: https://doi.org/10.1137/1.9781611978322.22.

[75] Gramoz Goranci, Monika Henzinger, Danupon Nanongkai, Thatchaphol Saranurak, Mikkel Thorup, and Christian Wulff-Nilsen. "Fully Dynamic Exact Edge Connectivity in Sublinear Time". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*. Ed. by Nikhil Bansal and Viswanath Nagarajan. SIAM, 2023, pp. 70–86. DOI: 10.1137/1.9781611977554.CH3. URL: https://doi.org/10.1137/1.9781611977554.ch3.

[77] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. "The Expander Hierarchy and its Applications to Dynamic Graph Algorithms". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*. Ed. by Dániel Marx. SIAM, 2021, pp. 2212–2228. DOI: 10.1137/1.9781611976465.132. URL: https://doi.org/10.1137/1.9781611976465.132.

[82] Monika Henzinger, Jason Li, Satish Rao, and Di Wang. "Deterministic Near-Linear Time Minimum Cut in Weighted Graphs". In: *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*. Ed. by David P. Woodruff. SIAM, 2024, pp. 3089–3139. DOI: 10.1137/1.9781611977912.111. URL: https://doi.org/10.1137/1.9781611977912.111.

[86] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity". In: *J. ACM* 48.4 (2001), pp. 723–760. DOI: 10.1145/502090.502095. URL: https://doi.org/10.1145/502090.502095.

[89] Wenyu Jin, Xiaorui Sun, and Mikkel Thorup. "Fully Dynamic Min-Cut of Superconstant Size in Subpolynomial Time". In: *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*. Ed. by David P. Woodruff. SIAM, 2024, pp. 2999–3026. DOI: 10.1137/1.9781611977912.107. URL: https://doi.org/10.1137/1.9781611977912.107.

[91] David R Karger. "Minimum cuts in near-linear time". In: *Journal of the ACM (JACM)* 47.1 (2000), pp. 46–76.

[92] David R Karger and Clifford Stein. "A new approach to the minimum cut problem". In: *Journal of the ACM (JACM)* 43.4 (1996), pp. 601–640.

[93] David R. Karger. "Using Randomized Sparsification to Approximate Minimum Cuts". In: *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*. Ed. by Daniel Dominic Sleator. ACM/SIAM, 1994, pp. 424–432. URL: http://dl.acm.org/citation.cfm?id=314464.314582.

[95] Ken-ichi Kawarabayashi and Mikkel Thorup. "Deterministic global minimum cut of a simple graph in near-linear time". In: *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*. 2015, pp. 665–674.

[101] Jason Li. "Deterministic mincut in almost-linear time". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 384–395.

[102] Jason Li and Debmalya Panigrahi. "Deterministic min-cut in poly-logarithmic max-flows". In: *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2020, pp. 85–92.

[103] Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. "A parameterized approximation scheme for min k-cut". In: *SIAM Journal on Computing* 0 (2022), FOCS20–205.

[112] Chaitanya Nalam and Thatchaphol Saranurak. "Maximal k-Edge-Connected Subgraphs in Weighted Graphs via Local Random Contraction". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*. Ed. by Nikhil Bansal and Viswanath Nagarajan. SIAM, 2023, pp. 183–211. DOI: 10.1137/1.9781611977554.CH8. URL: https://doi.org/10.1137/1.9781611977554.ch8.

[113] C St JA Nash-Williams. "Edge-disjoint spanning trees of finite graphs". In: *Journal of the London Mathematical Society* 1.1 (1961), pp. 445–450.

[123] Mikkel Thorup. "Fully-dynamic min-cut". In: *Combinatorica* 27.1 (2007), pp. 91–127.

[125] Mikkel Thorup and David R Karger. "Dynamic graph algorithms with applications". In: *Scandinavian Workshop on Algorithm Theory*. Springer. 2000, pp. 1–9.

[126] Tijn de Vos and Aleksander B. G. Christiansen. "Tree-Packing Revisited: Faster Fully Dynamic Min-Cut and Arboricity". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*. Ed. by Yossi Azar and Debmalya Panigrahi. SIAM, 2025, pp. 700–749. DOI: 10.1137/1.9781611978322.21. URL: https://doi.org/10.1137/1.9781611978322.21.

CHAPTER 7

What's Next?

In this thesis, I studied how to compute the edge connectivity of a graph using graph algorithms, and multiple distributed algorithm, in both loosely connected graphs (trees) and highly connected ones (complete graphs in population protocols). I would like to further my understanding on the connectivity of a network: how to compute it, how it structures the network, and how to harness this understanding to develop distributed algorithms that fully leverages this connectivity.

7.1 A Better Understanding of Connectivity

The first topic I would like to understand better is connectivity in graph algorithms. I would like to take a look at two open questions on the minimum cut problem.

7.1.1 Bridging the Gap Between the Subpolynomial and the Polynomial Regime

The first problem is about expanding the range of our minimum cut algorithm to larger values of the edge-connectivity.

Problem Definition

Let us remember what the problem at hand is: given an undirected, unweighted graph $G = (V, E)$, subject to edge updates, either edge insertions or edge deletions, the problem is to maintain the *edge connectivity* of the graph, that is, the number of edges one would need to remove to disconnect the graph.

We have seen that none of the state-of-the-art algorithms are optimal for all possible cases, and the case distinction corresponds to different values λ of the edge connectivity. In the case where λ is unbounded in the number of nodes n, the best algorithms were presented by de Vos and Christiansen [126], who achieved a $\tilde{O}(\lambda^{5.5}\sqrt{n})$ worst-case update time, Goranci, Henzinger, Nanongkai, Saranurak, Thorup, and Wulff-Nilsen [75], who achieve a $\tilde{O}(m^{1-1/31})$ amortized update time, and Kenneth-Mordoch and Krauthgamer [96], who give a $\tilde{O}(n^2/\lambda^2)$

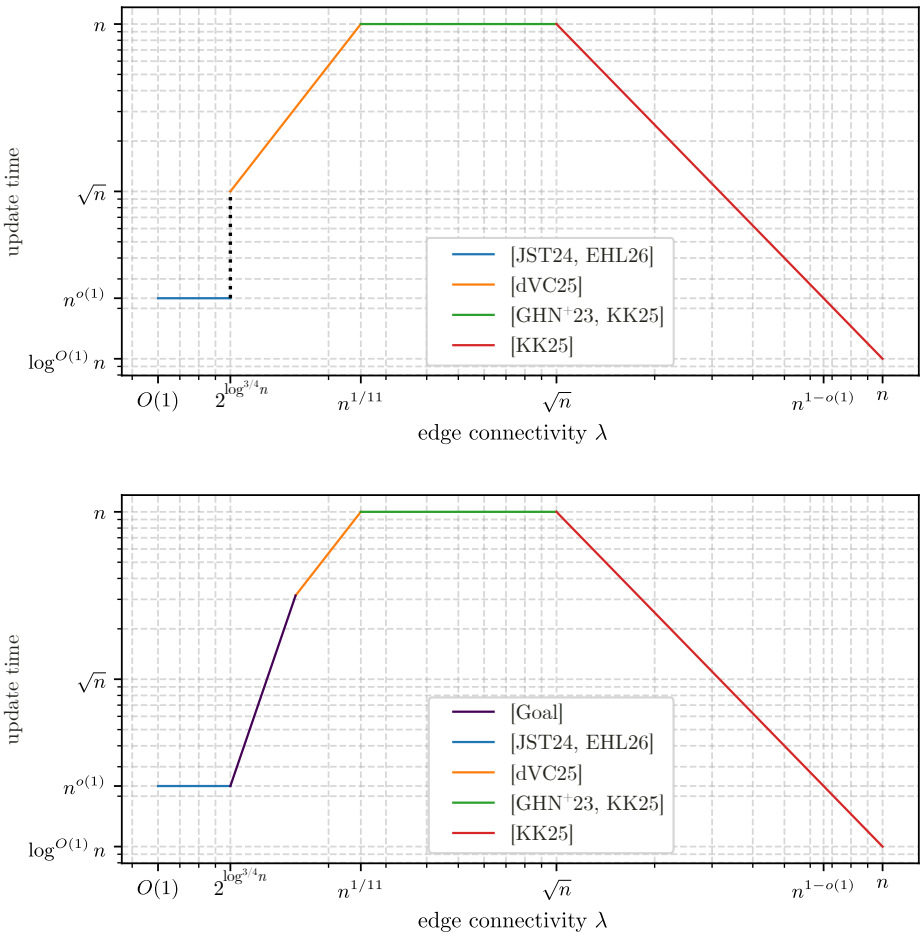


Figure 7.1: Top: Sketch of the state of the art algorithms for the dynamic minimum cut problem, completed from [96]. Bottom: Objective of our next work.

update time algorithm. In contrast, my coauthors Henzinger, and Li, and I [58], show that if the edge connectivity is small, namely if $\lambda = 2^{\log^{3/4-c} n}$ for some $c > 0$, then the update time can be reduced to $n^{o(1)}$.

As sketched in Figure 7.1, note that the bounds $\tilde{O}(\lambda^{5.5}\sqrt{n})$ and $n^{o(1)}$ do not match when $\lambda = 2^{\log^{3/4-c} n}$, as underlined by the discontinuity in the plot. This raises the question:

Question 7.1.1. *Is there a full-dynamic algorithm that solves the minimum cut problem, works for all values of λ, and runs in $\tilde{O}(poly(\lambda))$ time?*

Main Techniques

This problem has been widely studied, and the different results sketched in Figure 7.1, all stem from different techniques which solve the same problem.

A series of results, including Vos and Christiansen [126], and initiated by Thorup and Karger [125], rely on greedy tree packings. A greedy tree packing works as follows: given an unweighted graph G, make it weighted and set all of its edge weights to 0. Then, find a minimum spanning

tree, and increase the weights of all its edges by 1. Repeat that last step for a given number of steps N. The tree packing is the collection of the N trees found during these iterations.

As shown by Thorup and Karger [91, 123], the tree packing has the nice property that if N is large enough, then every minimum cut must *2-respect the tree*, that is, the tree must have at most 2 edges intersecting the minimum cut. This proves particularily useful, as maintaining cuts on a tree makes the problem simpler.

Another useful trick, used namely by Kenneth-Mordoch and Krauthgamer [96], and Goranci, Henzinger, Nanongkai, Saranurak, Thorup, and Wulff-Nilsen [75], and initiated by Kawarabayashi and Thorup [94] relies on a contraction of the graph that preserves all *non-trivial* minimum cuts, that is, minimum cuts which are not isolated vertices. The contraction of the graph contains $\tilde{O}(n/\delta)$ many vertices (where δ) is the minimum degree), which translates to at most $\tilde{O}(n)$ many edges by using Nagamochi and Ibaraki's techniques for sparsifying a graph while preserving minimum cuts [111]. To maintain the edge connectivity, it is the sufficient to maintain the minimum degree of the graph, the contracted graph, and run a static algorithm on the contracted graph on every request.

And finally, the result presented in this thesis, as discussed, relies on the fully-dynamic expander decomposition by Goranci, Räcke, Saranurak, and Tan [77]. The expander decomposition decomposes the graph into well-connected subgraphs, leaving few edges between components. However, the definition of "well-connectedness" of an expander does not match the idea of a "well-connectedness" of edge-connectivity. Therefore, the expanders have to be refined futher, into clusters. The cluster decomposition then has the guarantee that every minimum cut in the graph, can be approximated by by a cut that does not *cross* any cluster, that is, its intersection with the cluster is either the whole cluster, or is empty. Hence, one can contract every cluster, and recurse on the contracted graph.

Ideas for Imporovement

A line of research looks promising: Our result [58] relies on the fully-dynamic expander decomposition by Goranci, Räcke, Saranurak, and Tan [77], which holds for a specific expansion parameter $\phi = 2^{\Theta(\log^{3/4} n)}$. It is fair to assume that the authors went for the largest possible ϕ, as a larger expansion parameter give better-quality expanders, and thus that a wider range of ϕ should be possible. The hope is to find a trade-off, between the quality of the expanders, or the expansion parameter, ϕ, and the number of interexpander edges, which is of the order of $m\phi$, similarly to the static/high recourse case [118]. In our case, we can afford to have lower quality expanders, as our cluster decomposition on top can compensate for this lesser quality. The gains in the number of interexpander edges would then translate into gains in the running time, aiming for a $\tilde{O}(\mathrm{poly}(\lambda))$ update time algorithm.

7.1.2 A Dynamic Directed Minimum Cut

The second open problem pertains to the *directed* case.

Problem Definition

Here, we are given a *directed* graph $G = (V, E)$, and the goal is to find the *directed* minimum cut, that is a partition $\{V_1, V_2\}$ of the vertex set, minimizing the number of edges leaving V_1.

In 1992, Hao and Orlin [80] presented a $\tilde{O}(nm)$ time algorithm to solve the problem. This algorithm stayed for long the best known algorithm, until 2021, when Cen et al. [32] gave a $\tilde{O}(nm^{2/3}+n^2)$ time algorithm using blackbox calls to maxflow, and Chekuri and Quanrud [35] gave a $\tilde{O}(n^2)$ algorithm for unweighted graphs. The later improvements on maxflow by Chen et al. [36] improved the running time of Cen et al.'s algorithm down to $\min\{nm^{2/3+o(1)}, m^{1+o(1)}n^{1/2}\}$. A recent result by Mosenzon [109] was able to bring down the running time in the $(1+\epsilon)$-approximate setting to $\frac{m^{1+o(1)}}{\epsilon}$.

This series of work opens up the work on the dynamic setting for the directed minimum cut problem, where very little is known. In particular, the following question arises:

Question 7.1.2. *Is it possible to develop an algorithm for the dynamic directed minimum cut problem, either in the exact or approximate setting, that would run in $o(m)$ update time?*

Ideas to Explore

A new result raises the hopes for a dynamic directed minimum cut algorithm: the new dynamic directed expander decomposition by Sulser and Gutenberg [122].

Indeed, one could hope to mirror our result on undirected minimum cut, by first using the directed expander decomposition, then finding the correct way to generalise the cluster decomposition techniques to directed graphs, whether it is the LocalKCut algorithm, or the $(1-\epsilon)$-boundary sparseness techniques. As the expander decomposition is only given in the decremental setting, this would yield a decremental algorithm.

A first step might be to find an alternate algorithm to the one Mosenzon suggests. Indeed, their algorithm rely on repeated calls to the maxflow algorithm [36], and this in a non-trivial way. attempts to dynamise this algorithm have resulted in incremental or decremental algorithms [25, 26], however, as the reduction from the directed minimum cut is not immediate, one would need a fully-dynamic version of this algorithm to deal with either an incremental or a decremental directed minimum cut. Therefore, it might be best to ask the following question:

Question 7.1.3. *Is there an algorithm for the* static *directed minimum cut problem, either in the exact or approximate setting, that would run in $m^{1+o(1)}$ time, and does not call the maximum flow algorithm?*

7.2 Leveraging Connectivity for Distributed Routing

One open problem that relates to distributed computing and connectivity is whether or not ideal static routing tables exist, and can be computed.

Problem Definition

The goal of static routing tables is to forward messages, or *packets*, to their destination node in a network. The added difficulty is that the tables should be resilient to a number ℓ of *failures*, given by a set of edges being dropped *after* the tables have been chosen, that is, adversarially. On the network, represented by a graph, a static table is installed on each node, so that, whenever a packet arrives at said node, the node can decide where to forward it next

depending on the edge it was received from, the packet's destination, and the failing edges adjacent to the node. In other words, the static routing tables is a function $f : (u, v, F) \mapsto w$, where $u \in V$ is a node of the graph, $v, w \in \mathcal{N}(u)$ are neighbours of u, and $F \subseteq \{e \in E, u \in e\}$ is a set of edges incident to u.

Feigenbaum, Godfrey, Panda, Schapira, Shenker, and Singla [66] have shown that *perfect* resilience is impossible, that is, static tables that are able to handle any number of edge failures as long as the graph stays connected. Efforts have thus been aimed towards *ideal* resilience, where the goal is to find, in a k-edge connected graph, static routing tables resilient to $\ell = k - 1$ many edge failures.

Chiesa, Nikolaevskiy, Mitrovic, Panda, Gurtov, Madry, Schapira, and Shenker [38] show that $\ell = k - 1$ is achievable for up to $k = 5$, and for general k, $\ell = \left\lfloor \frac{k}{2} \right\rfloor$. This leaves open the question:

Question 7.2.1. *Is it possible to find ideal static routing tables? That is, for any given graph G that is k-edge connected, does a routing scheme resilient to $k - 1$ many failures exist? Is it possible to compute it algorithmically?*

Standard Techniques

The most successful techniques for solving this problem, all stem from one key idea: packing arborescences. Indeed, a key result by Edmonds [56] states that in a k-edge-connected graph, for a given root t, there exist k many arc-disjoint arborescences whose root is t.

Given this tool, the routing scheme is as follows. For each possible destination, let that destination be the root of k arborescences given by Edmonds's result. Give an arbitrary order $\mathcal{O}$ to these arborescences. For each node u, whenever a packet arrives from node v with destination x, look up in which arborescence with root x the arc (v, u) exists (only one such arborescence exists). This arborescence has only one outgoing arc (u, w) from u. If this edge has not failed, forward the package along this arc. If it has failed, forward the package along the next arborescence, according to order $\mathcal{O}$, whose outgoing arc (u, w') has not failed.

Since an edge contains two arcs (one in each direction), each edge failure can have the effect of up to two arborescences having a failure in them. Thus, to get all arborescences to fail, one would need $\left\lceil \frac{k}{2} \right\rceil$ edges. Therefore, the routing tables are $\left\lceil \frac{k}{2} \right\rceil - 1$-resilient.

The arborescence technique proved to be improvable. To get their routing scheme that is $\left\lfloor \frac{k}{2} \right\rfloor$-resilient, Chiesa, Nikolaevskiy, Mitrovic, Panda, Gurtov, Madry, Schapira, and Shenker [38] improved Edmonds's result:

Theorem 7.2.1 ([38])**.** *For any k-connected graph G, and any vertex $t \in V$, there exist k arc-disjoint arborescences $T_1, \ldots, T_k$ rooted at t such that, $T_1, \ldots, T_{\lfloor k/2 \rfloor}$ do not share edges with each other and $T_{\lfloor k/2 \rfloor + 1}, \ldots, T_k$ do not share edges with each other.*

Ideas for Improvement

It would be interesting to see if the structural results on k-edge connected graphs, arising from the graph algorithms studying edge connectivity, can be useful to complement the arborescence technique.

In particular, the cactus representation of a graph seems particularly adapted to this problem. A cactus graph is a connected graph in which every pair of simple cycles share at most one common vertex. Dinitz, Karzanov, and Lomonosov [44] showed the following structural result:

Theorem 7.2.2 ([44]). *Let k be an integer and $G = (V, E)$ a loopless graph for which the minimum cardinality of a cut is k. There is a cactus $C = (U, F)$ and a mapping $\Phi : V \mapsto U$ such that, for every cut of C with connected sides U_1 and U_2, the preimages $\Phi^{-1}(U_1)$ and $\Phi^1(U2)$ are the two sides of a mincut of G. Moreover, every mincut of G arises this way.*

In C, all edges contained in a cycle have weight $\frac{k}{2}$, and all other edges have weight k.

In a cactus graph, computing arborescences together with routing tables that are $k-1$ resilient is straightforward. It stems from two observations:

On a tree, where each edge has weight[1] k, the k arborescences are exact copies of this tree. Here, not only the arborescences do not share *arcs*, they do not share *edges*. Using the circular routing described above, the routing is $k-1$-resilient, given that each edge failure can only make one tree fail.

On a cycle, where each edge has weight $k/2$ (assume k is even), decompose the graph's edges into $k/2$ undirected cycles. Each cycle can now support 2 arborescences, one going in each direction. The ordering of the arborescences for the circular routing must satisfy the property that the two arborescences from the same cycle are consecutive. For the routing to fail, all cycles must become disconnected. One needs two failures per cycle to disconnect it, and thus k overall failures. This makes the routing $k-1$ resilient.

The routing scheme on a cactus tree is then a combination of these two ideas, where the packet travels closer to its destination along tree-like or cycle-like components of the graph, which are all k-edge connected. This thus raises the question:

Question 7.2.2. *Can an arborescence routing scheme given on the cactus representation of a graph, be lifted into a routing scheme for the graph itself?*

Another important structural result is the Gomory-Hu tree representation of a graph, which encapsulates all st-minimum cuts of the graph.

Theorem 7.2.3 ([74]). *For every undirected, possibly weighted graph G, there exists a weighted tree T on the same vertex set, such that for every par of vertices s, t in G, the st-minimum cut in G and the st-minimum cut in T correspond, both value-wise and vertex-set wise.*

Again, finding a routing scheme in the tree is trivial, and the question remains:

Question 7.2.3. *Can an arborescence routing scheme given on the Gomory-Hu tree of a graph, be lifted into a routing scheme for the graph itself?*

[1]Here, we say and edge has weight k, to mean that there are k parallel edges.

7.3 Questions in Population Protocols

During my PhD I focused on the plurality consensus problem: given that every agent begins with an opinion among k many, how can the agents coordinate to find which opinion was the most popular initially? Two big questions remain to be answered for this problem.

7.3.1 The State Complexity of Plurality Consensus

The state complexity problem asks: what is the minimum number of states per agent required to solve the Plurality Consensus problem?

Problem Definition

As a reminder, in population protocols, we are given n agents, each with limited memory. Initially, each agent is given an input among k possible opinions, and their goal is to compute the majority opinion. Computation happens in rounds, where in each round, a scheduler chooses two agents for interaction, and these agents see each other's internal state, and update their state accordingly. The goal is to find a *protocol* that solves this problem, that is, an *input function* that maps each input to a starting internal state, a *transition function* that maps two interacting states to their updates states, and an *output function* that maps each possible internal state to an output.

In this case, we consider a *weakly fair* scheduler, that is the most adversarial scheduler that one could face:

Definition 7.3.1 (Weakly Fair Scheduler)**.** *[12]] The weakly fair scheduler is a scheduler that chooses every pair of agents for interaction infinitely many times.*

And the question is then, with a weakly fair scheduler, how many states are necessary to solve the problem. We have shown, with Breitkopf, Dallot, and Schmid [28, 29], that it can be solved with k^3 many states per agent. Natale and Ramezani [114] gave a $\Omega(k^2)$ lower bound.

This brings the question:

Question 7.3.1. *With a weakly fair scheduler, what is the minimal number of states one needs to solve plurality consensus in population protocols? Can we bridge the gap between the $\Omega(k^2)$ lower bound and the current best k^3-states protocol?*

Ideas to Explore

Our result, together with Breitkopf, Dallot, and Schmid [28, 29], relies on this simple idea: imagine a (vertical) bar chart, with k bars, representing the number of agents holding each opinion. The tallest bar clearly is the bar representing the majority opinion. We develop a protocol that "deactivate" agents by following this histogram from bottom to top, stopping when there is only one opinion left in the population. The agents still active then share their opinion to the inactive agents.

There is a tightness gap in this algorithm: indeed, the memory of some deactivated agents become useless. Indeed, while some of them are only tentatively deactivated, as it is unclear

whether or not they might be reactivated in the future, others are definitively deactivated, and their memory is now only used to store the output. Such a leeway indicates that an improvement can be made, and some memory can be saved.

7.3.2 A Time-Space Tradeoff

The previous problem strictly looked at the space complexity of the problem, which is the only meaningful problem when looking at a weakly fair scheduler. However, this is the usual baseline before considering the time complexity as well, with a different scheduler in mind.

Problem Definition

Here, we look at plurality consensus, this time however, with a *random scheduler*.

Definition 7.3.2 (Random Scheduler)**.** *The random scheduler is a scheduler that chooses, for every round, two agents uniformly at random among all agents, independently from every other round.*

The question is then about time-space tradeoff: In the case where space is unlimited and agents have IDs, this problem is as easy as the easiest population protocol problem: Broadcast. It can thus be solved in $O(n \log n)$ many interactions. We however showed earlier this year with Elsässer and Schmid [57] that one of the most promising protocols fails to break the $O(kn \cdot \log n)$ time barrier, a barrier none of the protocols with less than n states per agent was able to break. It would thus be interesting to better understand this barrier:

Question 7.3.2. *Is it possible to solve the plurality consensus problem, in $o(kn \cdot \log n)$ expeted time, using $o(n)$ many states?*

7.4 Longer Term Questions

The questions discussed above would open up further questions in population protocols, and beyond.

7.4.1 Population Protocols on an Underlying, Well-Connected Graph

All the above questions are about *complete* graphs, that is, the agents are thought to be vertices of a complete graph, and for each interaction, an edge is chosen by a scheduler, the endpoints of which are the agents that interact. Thus, the next task would be to understand population protocols on general graphs, where agents are only able ton interact if they share an edge. There, techniques will differ substantially, as one cannot rely on any pair of agents to be able to meet. Most probably, techniques from graph algorithms could prove useful. In particular, I would be interested in exploring protocols on k-edge connected graphs, applying my knowledge on edge connectivity algorithms.

7.4.2 Towards Emerging Behaviour

Beyond information dissemination and population protocols, I would be interested in studying models of distributed computing where agents are limited in either computing power or communication bandwidth. Programmable matter and computational biology are an example of this, where agents have *both* limited computing power and communication bandwidth, from which a global behavior emerge from simple interactions. The beeping model is another example, where agents are limited to only one kind of message they are able to send out. This model of computation is a strong candidate to explain emerging behavior of cooperative animals.

References

[12] James Aspnes, Joffroy Beauquier, Janna Burman, and Devan Sohier. "Time and Space Optimal Counting in Population Protocols". In: *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*. Ed. by Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone. Vol. 70. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017, 13:1–13:17. ISBN: 978-3-95977-031-6. DOI: 10.4230/LIPIcs.OPODIS.2016.13. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.OPODIS.2016.13.

[25] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. "Incremental approximate maximum flow on undirected graphs in subpolynomial update time". In: *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2024, pp. 2980–2998.

[26] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Almost-Linear Time Algorithms for Decremental Graphs: Min-Cost Flow and More via Duality". In: *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*. IEEE, 2024, pp. 2010–2032. DOI: 10.1109/FOCS61266.2024.00120. URL: https://doi.org/10.1109/FOCS61266.2024.00120.

[28] Tom-Lukas Breitkopf, Julien Dallot, Antoine El-Hayek, and Stefan Schmid. "Brief Announcement: Minimizing Energy Solves Relative Majority with a Cubic Number of States in Population Protocols". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2025, Hotel Las Brisas Huatulco, Huatulco, Mexico, June 16-20, 2025*. Ed. by Alkida Balliu and Fabian Kuhn. ACM, 2025, pp. 549–552. DOI: 10.1145/3732772.3733512. URL: https://doi.org/10.1145/3732772.3733512.

[29] Tom-Lukas Breitkopf, Julien Dallot, Antoine El-Hayek, and Stefan Schmid. "Ranking Opinions with Few States in Population Protocols". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2026,Egham, United Kingdom, July 6-10, 2026*. ACM, 2026. DOI: 10.1145/3796701.3815913. URL: https:%20//doi.org/10.1145/3796701.3815913.

[32] Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Kent Quanrud. "Minimum Cuts in Directed Graphs via Partial Sparsification". In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 1147–1158. DOI: 10.1109/FOCS52979.2021.00113. URL: https://doi.org/10.1109/FOCS52979.2021.00113.

[35] Chandra Chekuri and Kent Quanrud. "Faster Algorithms for Rooted Connectivity in Directed Graphs". In: *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 49:1–49:16. DOI: 10.4230/

LIPICS.ICALP.2021.49. URL: https://doi.org/10.4230/LIPIcs.ICALP.2021.49.

[36] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time". In: *J. ACM* 72.3 (2025), 19:1–19:103. DOI: 10.1145/3728631. URL: https://doi.org/10.1145/3728631.

[38] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Aurojit Panda, Andrei V. Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. "The quest for resilient (static) forwarding tables". In: *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524552. URL: https://doi.org/10.1109/INFOCOM.2016.7524552.

[44] Efim A Dinitz, Alexander V Karzanov, and Michael V Lomonosov. "On the structure of the system of minimum edge cuts of a graph". In: *Issledovaniya po Diskretnoi Optimizatsii* (1976), pp. 290–306.

[56] Jack Edmonds. "Edge-disjoint branchings". In: *Combinatorial algorithms* (1973), pp. 91–96.

[57] Antoine El-Hayek, Robert Elsässer, and Stefan Schmid. "An Almost Tight Lower Bound for Plurality Consensus with Undecided State Dynamics in the Population Protocol Model". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2025, Hotel Las Brisas Huatulco, Huatulco, Mexico, June 16-20, 2025*. Ed. by Alkida Balliu and Fabian Kuhn. ACM, 2025, pp. 532–540. DOI: 10.1145/3732772.3733505. URL: https://doi.org/10.1145/3732772.3733505.

[58] Antoine El-Hayek, Monika Henzinger, and Jason Li. "Deterministic and Exact Fully-dynamic Minimum Cut of Superpolylogarithmic Size in Subpolynomial Time". In: *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*. Ed. by Kasper Green Larsen and Barna Saha. SIAM, 2026, pp. 613–663. DOI: 10.1137/1.9781611978971.25. URL: https://doi.org/10.1137/1.9781611978971.25.

[66] Joan Feigenbaum, Brighten Godfrey, Aurojit Panda, Michael Schapira, Scott Shenker, and Ankit Singla. "Brief announcement: on the resilience of routing tables". In: *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*. Ed. by Darek Kowalski and Alessandro Panconesi. ACM, 2012, pp. 237–238. DOI: 10.1145/2332432.2332478. URL: https://doi.org/10.1145/2332432.2332478.

[74] Ralph E Gomory and Tien Chung Hu. "Multi-terminal network flows". In: *Journal of the Society for Industrial and Applied Mathematics* 9.4 (1961), pp. 551–570.

[75] Gramoz Goranci, Monika Henzinger, Danupon Nanongkai, Thatchaphol Saranurak, Mikkel Thorup, and Christian Wulff-Nilsen. "Fully Dynamic Exact Edge Connectivity in Sublinear Time". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*. Ed. by Nikhil Bansal and Viswanath Nagarajan. SIAM, 2023, pp. 70–86. DOI: 10.1137/1.9781611977554.CH3. URL: https://doi.org/10.1137/1.9781611977554.ch3.

[77] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. "The Expander Hierarchy and its Applications to Dynamic Graph Algorithms". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*. Ed. by Dániel Marx. SIAM, 2021, pp. 2212–2228. DOI: 10.1137/1.9781611976465.132. URL: https://doi.org/10.1137/1.9781611976465.132.

[80] Jianxiu Hao and James B. Orlin. "A Faster Algorithm for Finding the Minimum Cut in a Directed Graph". In: *J. Algorithms* 17.3 (1994), pp. 424–446. DOI: 10.1006/JAGM.1994.1043. URL: https://doi.org/10.1006/jagm.1994.1043.

[91] David R Karger. "Minimum cuts in near-linear time". In: *Journal of the ACM (JACM)* 47.1 (2000), pp. 46–76.

[94] Ken-ichi Kawarabayashi and Mikkel Thorup. "Deterministic Edge Connectivity in Near-Linear Time". In: *J. ACM* 66.1 (2019), 4:1–4:50. DOI: 10.1145/3274663. URL: https://doi.org/10.1145/3274663.

[96] Yotam Kenneth-Mordoch and Robert Krauthgamer. "Simple Algorithms for Fully Dynamic Edge Connectivity". In: *2026 Symposium on Simplicity in Algorithms, SOSA 2026, Vancouver, BC, Canada, January 12-14, 2026*. Ed. by Sepehr Assadi and Eva Rotenberg. SIAM, 2026, pp. 394–403. DOI: 10.1137/1.9781611978964.31. URL: https://doi.org/10.1137/1.9781611978964.31.

[109] Ron Mosenzon. "Almost-optimal approximation algorithms for global minimum cut in directed graphs". In: *Proceedings of the 58th Annual ACM Symposium on Theory of Computing*. 2026, pp. 106–117.

[111] Hiroshi Nagamochi and Toshihide Ibaraki. "A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph". In: *Algorithmica* 7.5&6 (1992), pp. 583–596. DOI: 10.1007/BF01758778. URL: https://doi.org/10.1007/BF01758778.

[114] Emanuele Natale and Iliad Ramezani. "On the Necessary Memory to Compute the Plurality in Multi-agent Systems". In: *Algorithms and Complexity - 11th International Conference, CIAC 2019*. Ed. by Pinar Heggernes. Vol. 11485. Lecture Notes in Computer Science. Springer, 2019, pp. 323–338. DOI: 10.1007/978-3-030-17402-6_27. URL: https://doi.org/10.1007/978-3-030-17402-6_27.

[118] Thatchaphol Saranurak and Di Wang. "Expander Decomposition and Pruning: Faster, Stronger, and Simpler". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 2616–2635. DOI: 10.1137/1.9781611975482.162. URL: https://doi.org/10.1137/1.9781611975482.162.

[122] Aurelio L. Sulser and Maximilian Probst Gutenberg. "Near-Optimal Algorithm for Directed Expander Decompositions". In: *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, Aarhus, Denmark, July 8-11, 2025*. Ed. by Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis. Vol. 334. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, 132:1–132:20. DOI: 10.4230/LIPICS.ICALP.2025.132. URL: https://doi.org/10.4230/LIPIcs.ICALP.2025.132.

[123] Mikkel Thorup. "Fully-dynamic min-cut". In: *Combinatorica* 27.1 (2007), pp. 91–127.

[125] Mikkel Thorup and David R Karger. "Dynamic graph algorithms with applications". In: *Scandinavian Workshop on Algorithm Theory*. Springer. 2000, pp. 1–9.

[126] Tijn de Vos and Aleksander B. G. Christiansen. "Tree-Packing Revisited: Faster Fully Dynamic Min-Cut and Arboricity". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*. Ed. by Yossi Azar and Debmalya Panigrahi. SIAM, 2025, pp. 700–749. DOI: 10.1137/1.9781611978322.21. URL: https://doi.org/10.1137/1.9781611978322.21.

Bibliography

[1] Yehuda Afek and Eli Gafni. "Asynchrony from synchrony". In: *Distributed Computing and Networking: 14th International Conference, ICDCN 2013, Mumbai, India, January 3-6, 2013. Proceedings 14*. Springer. 2013, pp. 225–239.

[2] Yehuda Afek, Eli Gafni, Sergio Rajsbaum, Michel Raynal, and Corentin Travers. "The k-simultaneous consensus problem". In: *Distributed Comput.* 22.3 (2010), pp. 185–195. DOI: 10.1007/s00446-009-0090-8. URL: https://doi.org/10.1007/s00446-009-0090-8.

[3] Mohamad Ahmadi, Fabian Kuhn, Shay Kutten, Anisur Rahaman Molla, and Gopal Pandurangan. "The Communication Cost of Information Spreading in Dynamic Networks". In: *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*. IEEE, 2019, pp. 368–378. DOI: 10.1109/ICDCS.2019.00044. URL: https://doi.org/10.1109/ICDCS.2019.00044.

[4] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. "Time-Space Trade-offs in Population Protocols". In: *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'17*. 2017, pp. 2560–2579.

[5] Dan Alistarh, James Aspnes, and Rati Gelashvili. "Space-Optimal Majority in Population Protocols". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'18*. 2018, pp. 2221–2239.

[6] Dan Alistarh and Rati Gelashvili. "Polylogarithmic-Time Leader Election in Population Protocols". In: *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, ICALP'15, Part II*. 2015, pp. 479–491.

[7] Dan Alistarh and Rati Gelashvili. "Recent Algorithmic Advances in Population Protocols". In: *SIGACT News* 49.3 (2018), pp. 63–73. DOI: 10.1145/3289137.3289150. URL: https://doi.org/10.1145/3289137.3289150.

[8] Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. "Fast and Exact Majority in Population Protocols". In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC'15*. Ed. by Chryssis Georgiou and Paul G. Spirakis. ACM, 2015, pp. 47–56. ISBN: 978-1-4503-3617-8. DOI: 10.1145/2767386.2767429. URL: http://doi.acm.org/10.1145/2767386.2767429.

[9] Talley Amir, James Aspnes, Petra Berenbrink, Felix Biermeier, Christopher Hahn, Dominik Kaaser, and John Lazarsfeld. "Fast Convergence of k-Opinion Undecided State Dynamics in the Population Protocol Model". In: *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*. Ed. by Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu. ACM, 2023, pp. 13–23. DOI: 10.1145/3583668.3594589. URL: https://doi.org/10.1145/3583668.3594589.

[10] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. "Computation in networks of passively mobile finite-state sensors". In: *Distributed Computing* 18.4 (2006), pp. 235–253.

[11] Dana Angluin, James Aspnes, and David Eisenstat. "Fast computation by population protocols with a leader". In: *Distributed Computing* 21.3 (Sept. 2008), pp. 183–199.

[12] James Aspnes, Joffroy Beauquier, Janna Burman, and Devan Sohier. "Time and Space Optimal Counting in Population Protocols". In: *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*. Ed. by Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone. Vol. 70. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017, 13:1–13:17. ISBN: 978-3-95977-031-6. DOI: 10.4230/LIPIcs.OPODIS.2016.13. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.OPODIS.2016.13.

[13] James Aspnes and Eric Ruppert. "An introduction to population protocols". In: *Middleware for Network Eccentric and Mobile Applications*. Ed. by Benoît Garbinato, Hugo Miranda, and Luís Rodrigues. Springer-Verlag, 2009, pp. 97–120.

[14] John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. "Towards robust and efficient computation in dynamic peer-to-peer networks". In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. Ed. by Yuval Rabani. SIAM, 2012, pp. 551–569. DOI: 10.1137/1.9781611973099.47. URL: https://doi.org/10.1137/1.9781611973099.47.

[15] Gregor Bankhamer, Petra Berenbrink, Felix Biermeier, Robert Elsässer, Hamed Hosseinpour, Dominik Kaaser, and Peter Kling. "Fast Consensus via the Unconstrained Undecided State Dynamics". In: *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*. Ed. by Joseph (Seffi) Naor and Niv Buchbinder. SIAM, 2022, pp. 3417–3429. DOI: 10.1137/1.9781611977073.135. URL: https://doi.org/10.1137/1.9781611977073.135.

[16] Gregor Bankhamer, Petra Berenbrink, Felix Biermeier, Robert Elsässer, Hamed Hosseinpour, Dominik Kaaser, and Peter Kling. "Population Protocols for Exact Plurality Consensus: How a small chance of failure helps to eliminate insignificant opinions". In: *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*. Ed. by Alessia Milani and Philipp Woelfel. ACM, 2022, pp. 224–234. DOI: 10.1145/3519270.3538447. URL: https://doi.org/10.1145/3519270.3538447.

[17] Surender Baswana, Manoj Gupta, and Sandeep Sen. "Fully Dynamic Maximal Matching in O(log n) Update Time (Corrected Version)". In: *SIAM J. Comput.* 47.3 (2018), pp. 617–650. DOI: 10.1137/16M1106158. URL: https://doi.org/10.1137/16M1106158.

[18] Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Riccardo Silvestri. "Plurality Consensus in the Gossip Model". In: *Proceedings of the* 26*th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'15*. 2015, pp. 371–390.

[19] Stav Ben-Nun, Tsvi Kopelowitz, Matan Kraus, and Ely Porat. "An $O(\log^{3/2} n)$ Parallel Time Population Protocol for Majority with $O(\log n)$ States". In: *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*. Ed. by Yuval Emek and Christian Cachin. ACM, 2020, pp. 191–199. DOI: 10.1145/3382734.3405747. URL: https://doi.org/10.1145/3382734.3405747.

[20] Petra Berenbrink, Robert Elässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. "A population protocol for exact majority with $O(\log^{5/3} n)$ stabilization time and $O(\log n)$ states". In: *Proceedings of the 32nd International Symposium on Distributed Computing, DISC'18*. 2018.

[21] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Domink Kaaser Peter Kling, and Tomasz Radzik. "Time-space trade-offs in population protocols for the majority problem". In: *Distributed Computing* 34.2 (2021). Full version available at arXiv:1805.04586, pp. 91–111.

[22] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. "Towards Optimal Distributed Consensus (Extended Abstract)". In: *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. IEEE Computer Society, 1989, pp. 410–415. DOI: 10.1109/SFCS.1989.63511. URL: https://doi.org/10.1109/SFCS.1989.63511.

[23] Aaron Bernstein, Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Deterministic Dynamic Maximal Matching in Sublinear Update Time". In: *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*. Ed. by Michal Koucký and Nikhil Bansal. ACM, 2025, pp. 132–143. DOI: 10.1145/3717823.3718153. URL: https://doi.org/10.1145/3717823.3718153.

[24] Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. "Brief Announcement: Population Protocols for Leader Election and Exact Majority with $O(\log^2 n)$ States and $O(\log^2 n)$ Convergence Time". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC'17*. Full version available at arXiv:1705.01146. 2017, pp. 451–453.

[25] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. "Incremental approximate maximum flow on undirected graphs in subpolynomial update time". In: *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2024, pp. 2980–2998.

[26] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Almost-Linear Time Algorithms for Decremental Graphs: Min-Cost Flow and More via Duality". In: *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*. IEEE, 2024, pp. 2010–2032. DOI: 10.1109/FOCS61266.2024.00120. URL: https://doi.org/10.1109/FOCS61266.2024.00120.

[27] Jan van den Brand, Yang P. Liu, and Aaron Sidford. "Dynamic Maxflow via Dynamic Interior Point Methods". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*. Ed. by Barna Saha and Rocco A. Servedio. ACM, 2023, pp. 1215–1228. DOI: 10.1145/3564246.3585135. URL: https://doi.org/10.1145/3564246.3585135.

[28] Tom-Lukas Breitkopf, Julien Dallot, Antoine El-Hayek, and Stefan Schmid. "Brief Announcement: Minimizing Energy Solves Relative Majority with a Cubic Number of States in Population Protocols". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2025, Hotel Las Brisas Huatulco, Huatulco, Mexico, June 16-20, 2025*. Ed. by Alkida Balliu and Fabian Kuhn. ACM, 2025, pp. 549–552. DOI: 10.1145/3732772.3733512. URL: https://doi.org/10.1145/3732772.3733512.

[29] Tom-Lukas Breitkopf, Julien Dallot, Antoine El-Hayek, and Stefan Schmid. "Ranking Opinions with Few States in Population Protocols". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2026,Egham, United Kingdom, July 6-10, 2026*. ACM, 2026. DOI: 10.1145/3796701.3815913. URL: https:%20//doi.org/10.1145/3796701.3815913.

[30] Luca Cardelli and Attila Csiksz-Nagy. "The cell cycle switch computes approximate majority". In: *Nature Scientific Reports* 2 (2012), p. 656.

[31] Arthur Cayley. "A theorem on trees". In: *Quart. J. Math.* 23 (1889), pp. 376–378.

[32] Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Kent Quanrud. "Minimum Cuts in Directed Graphs via Partial Sparsification". In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 1147–1158. DOI: 10.1109/FOCS52979.2021.00113. URL: https://doi.org/10.1109/FOCS52979.2021.00113.

[33] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. "Approximate Consensus in Highly Dynamic Networks: The Role of Averaging Algorithms". In: *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*. Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Vol. 9135. Lecture Notes in Computer Science. Springer, 2015, pp. 528–539. DOI: 10.1007/978-3-662-47666-6_42. URL: https://doi.org/10.1007/978-3-662-47666-6%5C_42.

[34] Bernadette Charron-Bost and André Schiper. "The Heard-Of model: computing in distributed systems with benign faults". In: *Distributed Comput.* 22.1 (2009), pp. 49–71. DOI: 10.1007/s00446-009-0084-6. URL: https://doi.org/10.1007/s00446-009-0084-6.

[35] Chandra Chekuri and Kent Quanrud. "Faster Algorithms for Rooted Connectivity in Directed Graphs". In: *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 49:1–49:16. DOI: 10.4230/LIPICS.ICALP.2021.49. URL: https://doi.org/10.4230/LIPIcs.ICALP.2021.49.

[36] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time". In: *J. ACM* 72.3 (2025), 19:1–19:103. DOI: 10.1145/3728631. URL: https://doi.org/10.1145/3728631.

[37] Yuen-Jyue Chen, Neil Dalchau, Niranjan Srnivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. "Programmable chemical controlers made from DNA". In: *Nature Nanotechnology* 8.10 (2013), pp. 755–762.

[38] Marco Chiesa, Ilya Nikolaevskiy, Slobodan Mitrovic, Aurojit Panda, Andrei V. Gurtov, Aleksander Madry, Michael Schapira, and Scott Shenker. "The quest for resilient (static) forwarding tables". In: *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524552. URL: https://doi.org/10.1109/INFOCOM.2016.7524552.

[39] Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Micha Pilipczuk. "Designing FPT algorithms for cut problems using randomized contractions". In: *SIAM Journal on Computing* 45.4 (2016), pp. 1171–1229.

[40] Andrea Clementi, Mohesen Ghaffari, Luciano Gualà, Emanuele Natale, Francesco Pasquale, and Giacomo Scornavacca. "A Tight Analysis of the Parallel Undecided-State Dynamics with Two Colors". In: *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science, MFCS'18*. 2018, 28:1–28:15.

[41] Andrea E. F. Clementi, Pierluigi Crescenzi, Carola Doerr, Pierre Fraigniaud, Francesco Pasquale, and Riccardo Silvestri. "Rumor spreading in random evolving graphs". In: *Random Struct. Algorithms* 48.2 (2016), pp. 290–312. DOI: 10.1002/rsa.20586. URL: https://doi.org/10.1002/rsa.20586.

[42] Étienne Coulouma, Emmanuel Godard, and Joseph G. Peters. "A characterization of oblivious message adversaries for which Consensus is solvable". In: *Theor. Comput. Sci.* 584 (2015), pp. 80–90. DOI: 10.1016/j.tcs.2015.01.024. URL: https://doi.org/10.1016/j.tcs.2015.01.024.

[43] Frank Den Hollander. "Probability theory: The coupling method". In: *Lecture notes available online (https://pub.math.leidenuniv.nl/probability/lecturenotes/CouplingLectures.pdf)* (2012).

[44] Efim A Dinitz, Alexander V Karzanov, and Michael V Lomonosov. "On the structure of the system of minimum edge cuts of a graph". In: *Issledovaniya po Diskretnoi Optimizatsii* (1976), pp. 290–306.

[45] Michael Dinitz, Jeremy T. Fineman, Seth Gilbert, and Calvin Newport. "Smoothed Analysis of Information Spreading in Dynamic Networks". In: *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*. Ed. by Christian Scheideler. Vol. 246. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 18:1–18:22. DOI: 10.4230/LIPICS.DISC.2022.18. URL: https://doi.org/10.4230/LIPIcs.DISC.2022.18.

[46] Stefan Dobrev and Imrich Vrto. "Optimal Broadcasting in Hypercubes with Dynamic Faults". In: *Inf. Process. Lett.* 71.2 (1999), pp. 81–85. DOI: 10.1016/S0020-0190(99)00093-9. URL: https://doi.org/10.1016/S0020-0190(99)00093-9.

[47] Stefan Dobrev and Imrich Vrto. "Optimal Broadcasting in Tori with Dynamic Faults". In: *Parallel Process. Lett.* 12.1 (2002), pp. 17–22. DOI: 10.1142/S0129626402000781. URL: https://doi.org/10.1142/S0129626402000781.

[48] Benjamin Doerr. "Probabilistic Tools for the Analysis of Randomized Optimization Heuristics". In: *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*. Ed. by Benjamin Doerr and Frank Neumann. Natural Computing Series. Springer, 2020, pp. 1–87. DOI: 10.1007/978-3-030-29414-4_1. URL: https://doi.org/10.1007/978-3-030-29414-4%5C_1.

[49] Benjamin Doerr and Mahmoud Fouz. "Asymptotically optimal randomized rumor spreading". In: *International Colloquium on Automata, Languages, and Programming (ICALP)*. Springer. 2011, pp. 502–513.

[50] Danny Dolev and H. Raymond Strong. "Authenticated Algorithms for Byzantine Agreement". In: *SIAM J. Comput.* 12.4 (1983), pp. 656–666. DOI: 10.1137/0212045. URL: https://doi.org/10.1137/0212045.

[51] David Doty, Mahsa Eftekhari, Leszek Gasieniec, Eric E. Severson, Przemyslaw Uznanski, and Grzegorz Stachowiak. "A time and space optimal stable population protocol solving exact majority". In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 1044–1055. DOI: 10.1109/FOCS52979.2021.00104. URL: https://doi.org/10.1109/FOCS52979.2021.00104.

[52] David Doty and David Soloveichik. "Stable leader election in population protocols requires linear time". In: *Proceedings of the 29th International Symposium on Distributed Computing, DISC'15*. 2015. DOI: 10.1007/978-3-662-48653-5_17. URL: http://dx.doi.org/10.1007/978-3-662-48653-5_17.

[53] Moez Draief and Milan Vojnovic. "Convergence Speed of Binary Interval Consensus". In: *Proceedings of the 29th IEEE International Conference on Computer Communications, INFOCOM'10*. 2010, pp. 1792–1800. DOI: 10.1109/INFCOM.2010.5461999. URL: http://dx.doi.org/10.1109/INFCOM.2010.5461999.

[54] Rick Durrett and Dong Yao. "Susceptible–infected epidemics on evolving graphs". In: *Electronic Journal of Probability* 27 (2022), pp. 1–66.

[55] Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, Zhifeng Sun, and Emanuele Viola. "On the Complexity of Information Spreading in Dynamic Networks". In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*. Ed. by Sanjeev Khanna. SIAM, 2013, pp. 717–736. DOI: 10.1137/1.9781611973105.52. URL: https://doi.org/10.1137/1.9781611973105.52.

[56] Jack Edmonds. "Edge-disjoint branchings". In: *Combinatorial algorithms* (1973), pp. 91–96.

[57] Antoine El-Hayek, Robert Elsässer, and Stefan Schmid. "An Almost Tight Lower Bound for Plurality Consensus with Undecided State Dynamics in the Population Protocol Model". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2025, Hotel Las Brisas Huatulco, Huatulco, Mexico, June 16-20, 2025*. Ed. by Alkida Balliu and Fabian Kuhn. ACM, 2025, pp. 532–540. DOI: 10.1145/3732772.3733505. URL: https://doi.org/10.1145/3732772.3733505.

[58] Antoine El-Hayek, Monika Henzinger, and Jason Li. "Deterministic and Exact Fully-dynamic Minimum Cut of Superpolylogarithmic Size in Subpolynomial Time". In: *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*. Ed. by Kasper Green Larsen and Barna Saha. SIAM, 2026, pp. 613–663. DOI: 10.1137/1.9781611978971.25. URL: https://doi.org/10.1137/1.9781611978971.25.

[59] Antoine El-Hayek, Monika Henzinger, and Jason Li. "Fully Dynamic Approximate Minimum Cut in Subpolynomial Time per Operation". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*. Ed. by Yossi Azar and Debmalya Panigrahi. SIAM, 2025, pp. 750–784. DOI: 10.1137/1.9781611978322.22. URL: https://doi.org/10.1137/1.9781611978322.22.

[60] Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Asymptotically Tight Bounds on the Time Complexity of Broadcast and Its Variants in Dynamic Networks". In: *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*. Ed. by Yael Tauman Kalai. Vol. 251. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 47:1–47:21. DOI: 10.4230/LIPICS.ITCS.2023.47. URL: https://doi.org/10.4230/LIPIcs.ITCS.2023.47.

[61] Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Brief Announcement: Broadcasting Time in Dynamic Rooted Trees is Linear". In: *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*. Ed. by Alessia Milani and Philipp Woelfel. ACM, 2022, pp. 54–56. DOI: 10.1145/3519270.3538460. URL: https://doi.org/10.1145/3519270.3538460.

[62] Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. "Broadcast and Consensus in Stochastic Dynamic Networks with Byzantine Nodes and Adversarial Edges". In: *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*. Ed. by Dan Alistarh. Vol. 319. LIPIcs. Schloss Dagstuhl -

Leibniz-Zentrum für Informatik, 2024, 21:1–21:15. DOI: 10.4230/LIPICS.DISC.2024.21. URL: https://doi.org/10.4230/LIPIcs.DISC.2024.21.

[63] Faith Ellen, Barun Gorain, Avery Miller, and Andrzej Pelc. "Constant-Length Labeling Schemes for Deterministic Radio Broadcast". In: *ACM Trans. Parallel Comput.* 8.3 (2021), 14:1–14:17. DOI: 10.1145/3470633. URL: https://doi.org/10.1145/3470633.

[64] Robert Elsässer and Tomasz Radzik. "Recent Results in Population Protocols for Exact Majority and Leader Election". In: *Bull. EATCS* 126 (2018). URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/549/546.

[65] Patrick T Eugster, Rachid Guerraoui, A-M Kermarrec, and Laurent Massoulié. "Epidemic information dissemination in distributed systems". In: *Computer* 37.5 (2004), pp. 60–67.

[66] Joan Feigenbaum, Brighten Godfrey, Aurojit Panda, Michael Schapira, Scott Shenker, and Ankit Singla. "Brief announcement: on the resilience of routing tables". In: *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*. Ed. by Darek Kowalski and Alessandro Panconesi. ACM, 2012, pp. 237–238. DOI: 10.1145/2332432.2332478. URL: https://doi.org/10.1145/2332432.2332478.

[67] Pierre Fraigniaud and Emmanuel Lazard. "Methods and problems of communication in usual networks". In: *Discret. Appl. Math.* 53.1-3 (1994), pp. 79–133. DOI: 10.1016/0166-218X(94)90180-5. URL: https://doi.org/10.1016/0166-218X(94)90180-5.

[68] Matthias Függer, Thomas Nowak, and Kyrill Winkler. "On the radius of nonsplit graphs and information dissemination in dynamic networks". In: *Discret. Appl. Math.* 282 (2020), pp. 257–264. DOI: 10.1016/j.dam.2020.02.013. URL: https://doi.org/10.1016/j.dam.2020.02.013.

[69] Hugo Rincon Galeana, Ami Paz, Stefan Schmid, Ulrich Schmid, and Kyrill Winkler. "The Time Complexity of Consensus Under Oblivious Message Adversaries". In: *14th Innovations in Theoretical Computer Science (ITCS)*. 2023.

[70] Hugo Rincon Galeana, Ulrich Schmid, Kyrill Winkler, Ami Paz, and Stefan Schmid. "Topological Characterization of Consensus Solvability in Directed Dynamic Networks". In: *arXiv preprint arXiv:2304.02316* (2023).

[71] Yu Gao, Yang P. Liu, and Richard Peng. "Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao". In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 516–527. DOI: 10.1109/FOCS52979.2021.00058. URL: https://doi.org/10.1109/FOCS52979.2021.00058.

[72] Leszek Gasieniec, David D. Hamilton, Russell Martin, Paul G. Spirakis, and Grzegorz Stachowiak. "Deterministic Population Protocols for Exact Majority and Plurality". In: *20th International Conference on Principles of Distributed Systems, OPODIS 2016*. Ed. by Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone. Vol. 70. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 14:1–14:14. DOI: 10.4230/

LIPIcs.OPODIS.2016.14. URL: https://doi.org/10.4230/LIPIcs.OPODIS.2016.14.

[73] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. "Distributed MST and Routing in Almost Mixing Time". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*. Ed. by Elad Michael Schiller and Alexander A. Schwarzmann. ACM, 2017, pp. 131–140. DOI: 10.1145/3087801.3087827. URL: https://doi.org/10.1145/3087801.3087827.

[74] Ralph E Gomory and Tien Chung Hu. "Multi-terminal network flows". In: *Journal of the Society for Industrial and Applied Mathematics* 9.4 (1961), pp. 551–570.

[75] Gramoz Goranci, Monika Henzinger, Danupon Nanongkai, Thatchaphol Saranurak, Mikkel Thorup, and Christian Wulff-Nilsen. "Fully Dynamic Exact Edge Connectivity in Sublinear Time". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*. Ed. by Nikhil Bansal and Viswanath Nagarajan. SIAM, 2023, pp. 70–86. DOI: 10.1137/1.9781611977554.CH3. URL: https://doi.org/10.1137/1.9781611977554.ch3.

[76] Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. "Incremental Exact Min-Cut in Polylogarithmic Amortized Update Time". In: *ACM Trans. Algorithms* 14.2 (2018), 17:1–17:21. DOI: 10.1145/3174803. URL: https://doi.org/10.1145/3174803.

[77] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. "The Expander Hierarchy and its Applications to Dynamic Graph Algorithms". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*. Ed. by Dániel Marx. SIAM, 2021, pp. 2212–2228. DOI: 10.1137/1.9781611976465.132. URL: https://doi.org/10.1137/1.9781611976465.132.

[78] Spencer Greenberg and Mehryar Mohri. "Tight lower bound on the probability of a binomial exceeding its expectation". In: *Statistics & Probability Letters* 86 (2014), pp. 91–98.

[79] Manoj Gupta and Richard Peng. "Fully Dynamic (1+ e)-Approximate Matchings". In: *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, Berkeley, CA, USA, October, 26-29, 2013*. IEEE Computer Society, 2013, pp. 548–557. DOI: 10.1109/FOCS.2013.65. URL: https://doi.org/10.1109/FOCS.2013.65.

[80] Jianxiu Hao and James B. Orlin. "A Faster Algorithm for Finding the Minimum Cut in a Directed Graph". In: *J. Algorithms* 17.3 (1994), pp. 424–446. DOI: 10.1006/JAGM.1994.1043. URL: https://doi.org/10.1006/jagm.1994.1043.

[81] Sandra Mitchell Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. "A survey of gossiping and broadcasting in communication networks". In: *Networks* 18.4 (1988), pp. 319–349. DOI: 10.1002/net.3230180406. URL: https://doi.org/10.1002/net.3230180406.

[82] Monika Henzinger, Jason Li, Satish Rao, and Di Wang. "Deterministic Near-Linear Time Minimum Cut in Weighted Graphs". In: *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*. Ed. by David P. Woodruff. SIAM, 2024, pp. 3089–3139. DOI: 10.1137/1.9781611977912.111. URL: https://doi.org/10.1137/1.9781611977912.111.

[83] Monika Rauch Henzinger. "A Static 2-Approximation Algorithm for Vertex Connectivity and Incremental Approximation Algorithms for Edge and Vertex Connectivity". In: *J. Algorithms* 24.1 (1997), pp. 194–220. DOI: 10.1006/JAGM.1997.0855. URL: https://doi.org/10.1006/jagm.1997.0855.

[84] Monika Rauch Henzinger and Michael L. Fredman. "Lower Bounds for Fully Dynamic Connectivity Problems in Graphs". In: *Algorithmica* 22.3 (1998), pp. 351–362. DOI: 10.1007/PL00009228. URL: https://doi.org/10.1007/PL00009228.

[85] Wassily Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30.

[86] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity". In: *J. ACM* 48.4 (2001), pp. 723–760. DOI: 10.1145/502090.502095. URL: https://doi.org/10.1145/502090.502095.

[87] Juraj Hromkovi, Ralf Klasing, Burkhard Monien, and Regine Peine. "Dissemination of information in interconnection networks (broadcasting & gossiping)". In: *Combinatorial network theory*. Springer, 1996, pp. 125–212.

[88] Wenyu Jin and Xiaorui Sun. "Fully Dynamic s-t Edge Connectivity in Subpolynomial Time (Extended Abstract)". In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 861–872. DOI: 10.1109/FOCS52979.2021.00088. URL: https://doi.org/10.1109/FOCS52979.2021.00088.

[89] Wenyu Jin, Xiaorui Sun, and Mikkel Thorup. "Fully Dynamic Min-Cut of Superconstant Size in Subpolynomial Time". In: *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*. Ed. by David P. Woodruff. SIAM, 2024, pp. 2999–3026. DOI: 10.1137/1.9781611977912.107. URL: https://doi.org/10.1137/1.9781611977912.107.

[90] Bruce M. Kapron, Valerie King, and Ben Mountjoy. "Dynamic graph connectivity in polylogarithmic worst case time". In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*. Ed. by Sanjeev Khanna. SIAM, 2013, pp. 1131–1142. DOI: 10.1137/1.9781611973105.81. URL: https://doi.org/10.1137/1.9781611973105.81.

[91] David R Karger. "Minimum cuts in near-linear time". In: *Journal of the ACM (JACM)* 47.1 (2000), pp. 46–76.

[92] David R Karger and Clifford Stein. "A new approach to the minimum cut problem". In: *Journal of the ACM (JACM)* 43.4 (1996), pp. 601–640.

[93] David R. Karger. "Using Randomized Sparsification to Approximate Minimum Cuts". In: *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*. Ed. by Daniel Dominic Sleator. ACM/SIAM, 1994, pp. 424–432. URL: http://dl.acm.org/citation.cfm?id=314464.314582.

[94] Ken-ichi Kawarabayashi and Mikkel Thorup. "Deterministic Edge Connectivity in Near-Linear Time". In: *J. ACM* 66.1 (2019), 4:1–4:50. DOI: 10.1145/3274663. URL: https://doi.org/10.1145/3274663.

[95] Ken-ichi Kawarabayashi and Mikkel Thorup. "Deterministic global minimum cut of a simple graph in near-linear time". In: *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*. 2015, pp. 665–674.

[96] Yotam Kenneth-Mordoch and Robert Krauthgamer. "Simple Algorithms for Fully Dynamic Edge Connectivity". In: *2026 Symposium on Simplicity in Algorithms, SOSA 2026, Vancouver, BC, Canada, January 12-14, 2026*. Ed. by Sepehr Assadi and Eva Rotenberg. SIAM, 2026, pp. 394–403. DOI: 10.1137/1.9781611978964.31. URL: https://doi.org/10.1137/1.9781611978964.31.

[97] Adrian Kosowski and Przemyslaw Uznanski. "Brief Announcement: Population Protocols Are Fast". In: *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing, PODC'18*. Full version available at arXiv:1802.06872. 2018, pp. 475–477. DOI: 10.1145/3212734.3212788. URL: http://doi.acm.org/10.1145/3212734.3212788.

[98] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. "Distributed computation in dynamic networks". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. Ed. by Leonard J. Schulman. ACM, 2010, pp. 513–522. DOI: 10.1145/1806689.1806760. URL: https://doi.org/10.1145/1806689.1806760.

[99] Fabian Kuhn and Rotem Oshman. "Dynamic networks: models and algorithms". In: *ACM SIGACT News* 42.1 (2011), pp. 82–96.

[100] Johannes Lengler. "Drift Analysis". In: *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*. Ed. by Benjamin Doerr and Frank Neumann. Natural Computing Series. Springer, 2020, pp. 89–131. DOI: 10.1007/978-3-030-29414-4_2. URL: https://doi.org/10.1007/978-3-030-29414-4%5C_2.

[101] Jason Li. "Deterministic mincut in almost-linear time". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 384–395.

[102] Jason Li and Debmalya Panigrahi. "Deterministic min-cut in poly-logarithmic max-flows". In: *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2020, pp. 85–92.

[103] Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. "A parameterized approximation scheme for min k-cut". In: *SIAM Journal on Computing* 0 (2022), FOCS20–205.

[104] Linyuan Lu, Austin Mohr, and László Székely. "Quest for negative dependency graphs". In: *Recent Advances in Harmonic Analysis and Applications*. Springer, 2012, pp. 243–258.

[105] Simon Meierhans and Maximilian Probst Gutenberg. "Dynamic Connectivity with Expected Polylogarithmic Worst-Case Update Time". In: *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*. Ed. by Kasper Green Larsen and Barna Saha. SIAM, 2026, pp. 6184–6198. DOI: 10.1137/1.9781611978971.220. URL: https://doi.org/10.1137/1.9781611978971.220.

[106] Uri Meir, Ami Paz, and Gregory Schwartzman. "Models of Smoothing in Dynamic Networks". In: *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*. Ed. by Hagit Attiya. Vol. 179. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 36:1–36:16. DOI: 10.4230/LIPICS.DISC.2020.36. URL: https://doi.org/10.4230/LIPIcs.DISC.2020.36.

[107] George B. Mertzios, Sotiris E. Nikoletseas, Christoforos L. Raptopoulos, and Paul G. Spirakis. "Determining Majority in Networks with Local Interactions and Very Small Local Memory". English. In: *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, ICALP'14*. Springer Berlin Heidelberg, 2014, pp. 871–882. ISBN: 978-3-662-43947-0. DOI: 10.1007/978-3-662-43948-7_72. URL: http://dx.doi.org/10.1007/978-3-662-43948-7_72.

[108] Othon Michail, George Skretas, and Paul G Spirakis. "Distributed computation and reconfiguration in actively dynamic networks". In: *Proceedings of the 39th Symposium on Principles of Distributed Computing*. 2020, pp. 448–457.

[109] Ron Mosenzon. "Almost-optimal approximation algorithms for global minimum cut in directed graphs". In: *Proceedings of the 58th Annual ACM Symposium on Theory of Computing*. 2026, pp. 106–117.

[110] James D Murray et al. *Mathematical biology I: an introduction*. 2002.

[111] Hiroshi Nagamochi and Toshihide Ibaraki. "A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph". In: *Algorithmica* 7.5&6 (1992), pp. 583–596. DOI: 10.1007/BF01758778. URL: https://doi.org/10.1007/BF01758778.

[112] Chaitanya Nalam and Thatchaphol Saranurak. "Maximal k-Edge-Connected Subgraphs in Weighted Graphs via Local Random Contraction". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*. Ed. by Nikhil Bansal and Viswanath Nagarajan. SIAM, 2023, pp. 183–211. DOI: 10.1137/1.9781611977554.CH8. URL: https://doi.org/10.1137/1.9781611977554.ch8.

[113] C St JA Nash-Williams. "Edge-disjoint spanning trees of finite graphs". In: *Journal of the London Mathematical Society* 1.1 (1961), pp. 445–450.

[114] Emanuele Natale and Iliad Ramezani. "On the Necessary Memory to Compute the Plurality in Multi-agent Systems". In: *Algorithms and Complexity - 11th International Conference, CIAC 2019*. Ed. by Pinar Heggernes. Vol. 11485. Lecture Notes in Computer Science. Springer, 2019, pp. 323–338. DOI: 10.1007/978-3-030-17402-6_27. URL: https://doi.org/10.1007/978-3-030-17402-6_27.

[115] Pietro S. Oliveto and Carsten Witt. "Improved time complexity analysis of the Simple Genetic Algorithm". In: *Theor. Comput. Sci.* 605 (2015), pp. 21–41. DOI: 10.1016/J.TCS.2015.01.002. URL: https://doi.org/10.1016/j.tcs.2015.01.002.

[116] Jim Pitman. "Coalescent random forests". In: *Journal of Combinatorial Theory, Series A* 85.2 (1999), pp. 165–193.

[117] Nicola Santoro and Peter Widmayer. "Time is Not a Healer". In: *STACS 89, 6th Annual Symposium on Theoretical Aspects of Computer Science, Paderborn, FRG, February 16-18, 1989, Proceedings*. Ed. by Burkhard Monien and Robert Cori. Vol. 349. Lecture Notes in Computer Science. Springer, 1989, pp. 304–313. DOI: 10.1007/BFb0028994. URL: https://doi.org/10.1007/BFb0028994.

[118] Thatchaphol Saranurak and Di Wang. "Expander Decomposition and Pruning: Faster, Stronger, and Simpler". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 2616–2635. DOI: 10.1137/1.9781611975482.162. URL: https://doi.org/10.1137/1.9781611975482.162.

[119] Jonathan DH Smith. *Introduction to abstract algebra*. Vol. 31. CRC Press, 2015.

[120] Shay Solomon. "Fully Dynamic Maximal Matching in Constant Update Time". In: *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, Hyatt Regency, New Brunswick, New Jersey, USA, October 9-11, 2016*. Ed. by Irit Dinur. IEEE Computer Society, 2016, pp. 325–334. DOI: 10.1109/FOCS.2016.43. URL: https://doi.org/10.1109/FOCS.2016.43.

[121] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Brook. "Computation with finite stochastic chemical reaction networks". In: *Natural Computing* 4.7 (2008), pp. 615–633.

[122] Aurelio L. Sulser and Maximilian Probst Gutenberg. "Near-Optimal Algorithm for Directed Expander Decompositions". In: *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, Aarhus, Denmark, July 8-11, 2025*. Ed. by Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis. Vol. 334. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, 132:1–132:20. DOI: 10.4230/LIPICS.ICALP.2025.132. URL: https://doi.org/10.4230/LIPIcs.ICALP.2025.132.

[123] Mikkel Thorup. "Fully-dynamic min-cut". In: *Combinatorica* 27.1 (2007), pp. 91–127.

[124] Mikkel Thorup. "Near-optimal fully-dynamic graph connectivity". In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*. Ed. by F. Frances Yao and Eugene M. Luks. ACM, 2000, pp. 343–350. DOI: 10.1145/335305.335345. URL: https://doi.org/10.1145/335305.335345.

[125] Mikkel Thorup and David R Karger. "Dynamic graph algorithms with applications". In: *Scandinavian Workshop on Algorithm Theory*. Springer. 2000, pp. 1–9.

[126] Tijn de Vos and Aleksander B. G. Christiansen. "Tree-Packing Revisited: Faster Fully Dynamic Min-Cut and Arboricity". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*. Ed. by Yossi Azar and Debmalya Panigrahi. SIAM, 2025, pp. 700–749. DOI: 10.1137/1.9781611978322.21. URL: https://doi.org/10.1137/1.9781611978322.21.

[127] Kyrill Winkler, Hugo Rincon Galeana, Ami Paz, Stefan Schmid, and Ulrich Schmid. "The Time Complexity of Consensus Under Oblivious Message Adversaries". In: *14th Innovations in Theoretical Computer Science (ITCS)*. 2023.

[128] Christian Wulff-Nilsen. "Fully-dynamic minimum spanning forest with improved worst-case update time". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. Ed. by Hamed Hatami, Pierre McKenzie, and Valerie King. ACM, 2017, pp. 1130–1143. DOI: 10.1145/3055399.3055415. URL: https://doi.org/10.1145/3055399.3055415.

[129] Martin Zeiner, Manfred Schwarz, and Ulrich Schmid. "On linear-time data dissemination in dynamic rooted trees". In: *Discret. Appl. Math.* 255 (2019), pp. 307–319. DOI: 10.1016/j.dam.2018.08.015. URL: https://doi.org/10.1016/j.dam.2018.08.015.