Ranking Opinions with Few States in Population Protocols

Joint work with Tom-Lukas Breitkopf, Julien Dallot, and Stefan Schmid

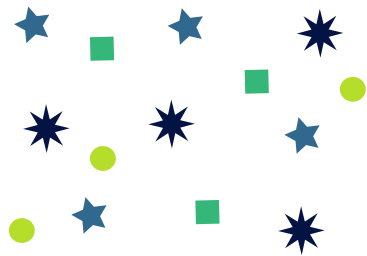
Antoine El-Hayek

PODC 2026

Population Protocols

- n agents, each with an initial state
- At each time step, two agents chosen by a scheduler meet
- They see each other's state, and update theirs according to some rule

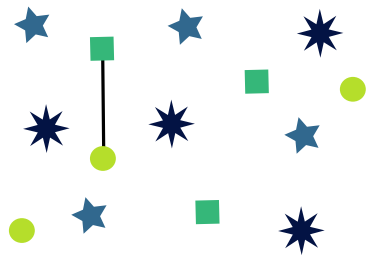
Example: Agent with more angles gets duplicated



Population Protocols

- n agents, each with an initial state
- At each time step, two agents chosen by a scheduler meet
- They see each other's state, and update theirs according to some rule

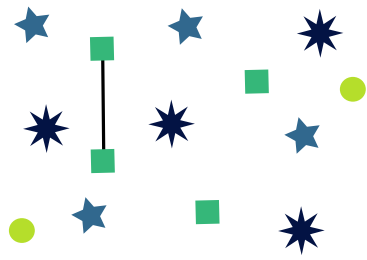
Example: Agent with more angles gets duplicated



Population Protocols

- n agents, each with an initial state
- At each time step, two agents chosen by a scheduler meet
- They see each other's state, and update theirs according to some rule

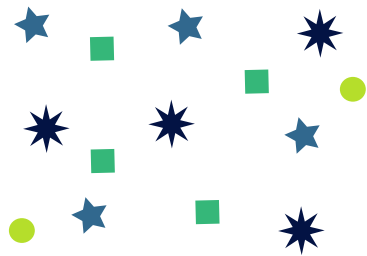
Example: Agent with more angles gets duplicated



Population Protocols

- n agents, each with an initial state
- At each time step, two agents chosen by a scheduler meet
- They see each other's state, and update theirs according to some rule

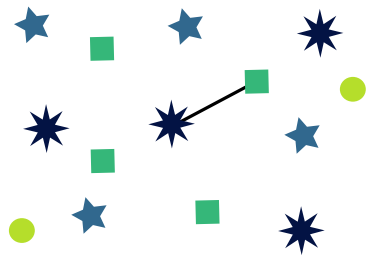
Example: Agent with more angles gets duplicated



Population Protocols

- n agents, each with an initial state
- At each time step, two agents chosen by a scheduler meet
- They see each other's state, and update theirs according to some rule

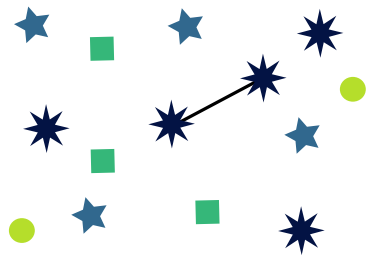
Example: Agent with more angles gets duplicated



Population Protocols

- n agents, each with an initial state
- At each time step, two agents chosen by a scheduler meet
- They see each other's state, and update theirs according to some rule

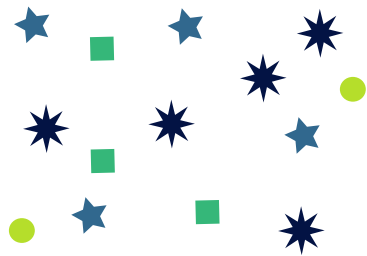
Example: Agent with more angles gets duplicated



Population Protocols

- n agents, each with an initial state
- At each time step, two agents chosen by a scheduler meet
- They see each other's state, and update theirs according to some rule

Example: Agent with more angles gets duplicated



Full definition

- We have n agents

Full definition

- We have n agents
- Each agent has an input

Full definition

- We have n agents
- Each agent has an input
- Each agent is assigned an initial state according to a function $\mathcal{I}$ on the input

Full definition

- We have n agents
- Each agent has an input
- Each agent is assigned an initial state according to a function $\mathcal{I}$ on the input
- In each round, two agents are selected for interaction according to a a schedule S

Full definition

- We have n agents
- Each agent has an input
- Each agent is assigned an initial state according to a function $\mathcal{I}$ on the input
- In each round, two agents are selected for interaction according to a a schedule S
- When two agents interact, they update their states according to a *transition function* $\mathcal{T}$

Full definition

- We have n agents
- Each agent has an input
- Each agent is assigned an initial state according to a function $\mathcal{I}$ on the input
- In each round, two agents are selected for interaction according to a a schedule S
- When two agents interact, they update their states according to a *transition function* $\mathcal{T}$
- Each state corresponds to an output according to a function $\mathcal{O}$

Full definition

- We have n agents
- Each agent has an input
- Each agent is assigned an initial state according to a function $\mathcal{I}$ on the input
- In each round, two agents are selected for interaction according to a a schedule S
- When two agents interact, they update their states according to a *transition function* $\mathcal{T}$
- Each state corresponds to an output according to a function $\mathcal{O}$
- There must exist a time T after which all agents output the correct answer

Full definition

- We have n agents
- Each agent has an input
- Each agent is assigned an initial state according to a function $\mathcal{I}$ on the input
- In each round, two agents are selected for interaction according to a a schedule S
- When two agents interact, they update their states according to a *transition function* $\mathcal{T}$
- Each state corresponds to an output according to a function $\mathcal{O}$
- There must exist a time T after which all agents output the correct answer

- The protocol is given by $\mathcal{I}, \mathcal{T}, \mathcal{O}$

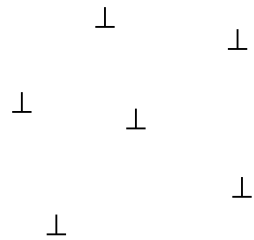
Full definition

- We have n agents
- Each agent has an input
- Each agent is assigned an initial state according to a function $\mathcal{I}$ on the input
- In each round, two agents are selected for interaction according to a a schedule S
- When two agents interact, they update their states according to a *transition function* $\mathcal{T}$
- Each state corresponds to an output according to a function $\mathcal{O}$
- There must exist a time T after which all agents output the correct answer

- The protocol is given by $\mathcal{I}, \mathcal{T}, \mathcal{O}$
- The schedule S is given by a *scheduler* (random or adversarial)

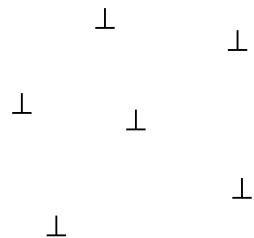
Example: Leader Election

- Input: nothing (symbolized by $\perp$)



Example: Leader Election

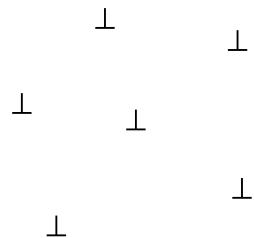
- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)



Example: Leader Election

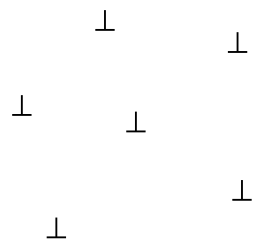
- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times



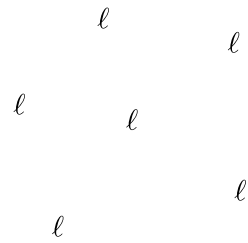
Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)



Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)

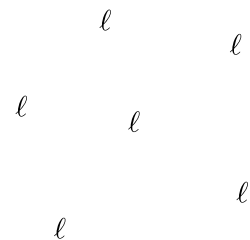


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

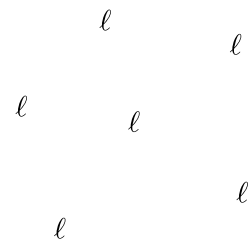


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

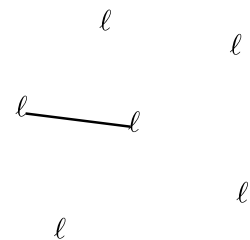


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

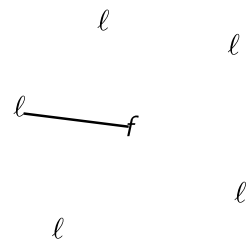


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

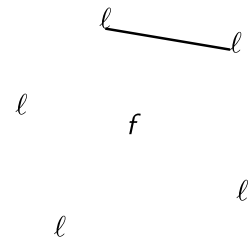


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

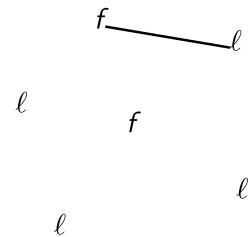


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

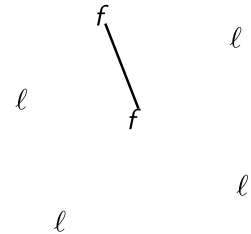


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

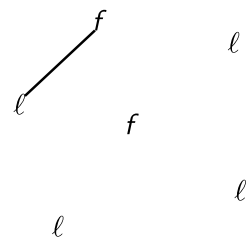


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

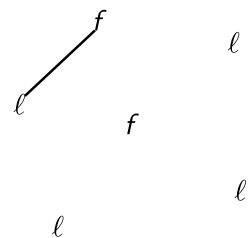


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

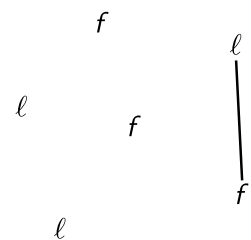


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

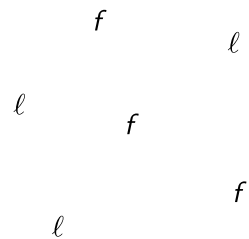


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$

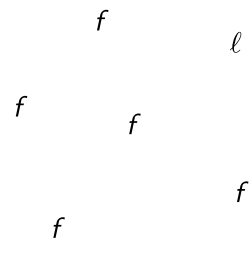


Example: Leader Election

- Input: nothing (symbolized by $\perp$)
- Output: one agent should output L (for leader), all others should output F (for follower)

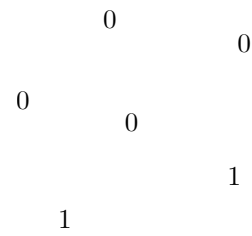
- Scheduler: Adversarial, each pair should interact infinitely many times

- Input function: $\mathcal{I}(\perp) = \ell$ (ℓ for Leader)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\ell, f) & \text{if } x = y = \ell \\ (x, y) & \text{otherwise} \end{cases}$
- Output function: $\mathcal{O}(x) = \begin{cases} L & \text{if } x = \ell \\ F & \text{if } x = f \end{cases}$



Example: Majority Consensus, [AADFP'2004][1]

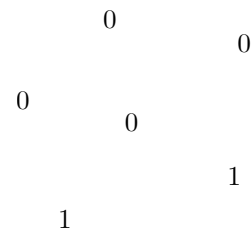
- Input: 0 or 1



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

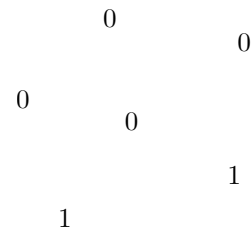
- Input: 0 or 1
- Output: The majority input, either 0 or 1



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

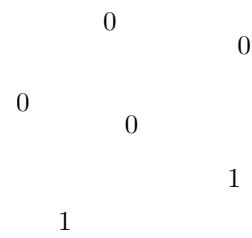
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

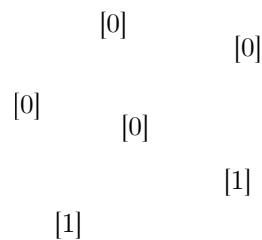
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$

$[0]$ $[0]$ $[0]$ $[0]$ $[1]$ $[1]$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

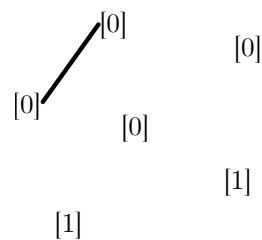
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$

$[0]$ $[0]$ $[0]$ $[0]$ $[1]$ $[1]$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

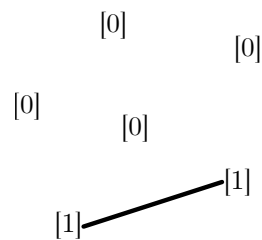
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$

$[0]$ $[0]$ $[0]$ $[0]$ $[1]$ $[1]$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

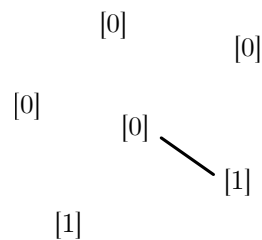
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$

$[0]$ $[0]$ $[0]$ $[0]$ $[1]$ $[1]$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

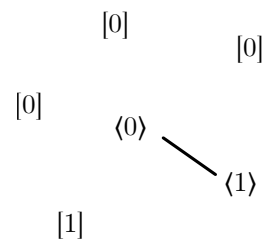
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

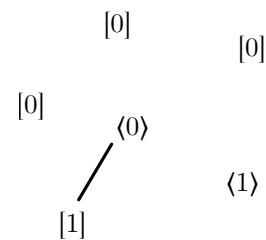
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$

$[0]$ $[0]$ $[0]$ $\langle 0\rangle$ $\langle 1\rangle$ $[1]$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

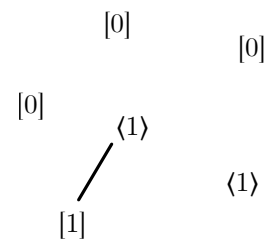
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

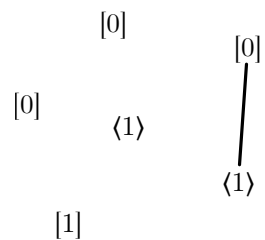
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$

$[0]$ $[0]$ $[0]$ $\langle 1\rangle$ $\langle 1\rangle$ $[1]$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

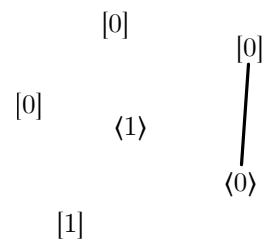
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

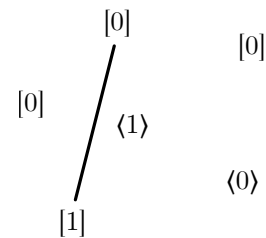
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$

$[0]$ $[0]$ $[0]$ $\langle 1\rangle$ $\langle 0\rangle$ $[1]$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

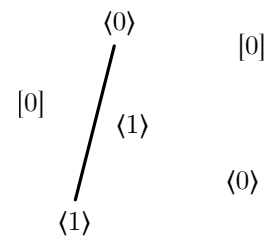
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

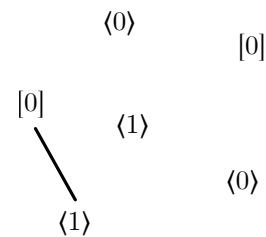
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$

$\langle 0\rangle$ $[0]$ $[0]$ $\langle 1\rangle$ $\langle 0\rangle$ $\langle 1\rangle$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

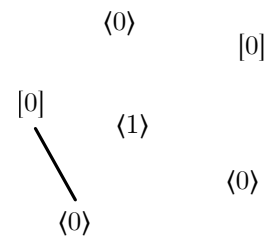
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

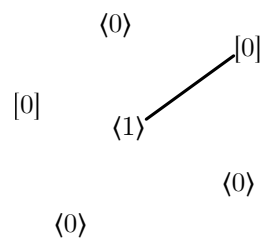
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$

$\langle 0\rangle$ $[0]$ $[0]$ $\langle 1\rangle$ $\langle 0\rangle$ $\langle 0\rangle$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

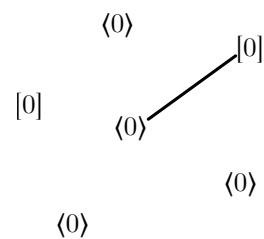
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$



[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

Example: Majority Consensus, [AADFP'2004][1]

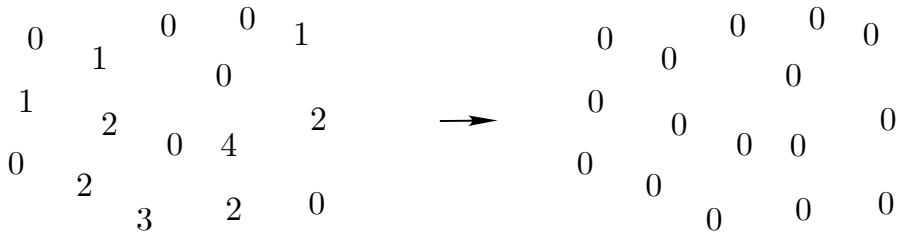
- Input: 0 or 1
- Output: The majority input, either 0 or 1
- Scheduler: Adversarial, each pair should interact infinitely many times
- Input function: $\mathcal{I}(\mathrm{i}) = [\mathrm{i}]$ ([ ] represent *strong* opinions, $\langle\ \rangle$ represent *weak* opinions)
- Transition Function: $\mathcal{T}(x, y) = \begin{cases} (\langle 0\rangle, \langle 1\rangle) & \text{if } x = [0], y = [1] \\ ([\mathrm{i}], \langle \mathrm{i}\rangle) & \text{if } x = [\mathrm{i}], y = \langle \mathrm{j}\rangle \\ (x, y) & \text{otherwise} \end{cases}$
- Output function: $\mathcal{O}([\mathrm{i}]) = \mathcal{O}(\langle \mathrm{i}\rangle) = i$

$\langle 0\rangle$ $[0]$ $[0]$ $\langle 0\rangle$ $\langle 0\rangle$ $\langle 0\rangle$

[1] Angluin, Aspnes, Diamadi, Fischer, and Peralta. Computation in networks of passively mobile finite-state sensors. PODC 2004

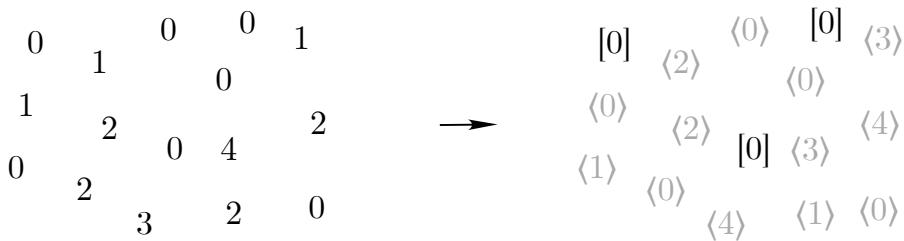
The Plurality Consensus Problem

- Input: an element in $\{0, 1, \ldots, k-1\}$
- Output: an element in $\{0, 1, \ldots, k-1\}$, whichever was the most common in the input



The Plurality Consensus Problem

- Input: an element in $\{0, 1, \ldots, k-1\}$
- Output: an element in $\{0, 1, \ldots, k-1\}$, whichever was the most common in the input
- It suffices for agents with strong opinions to be correct

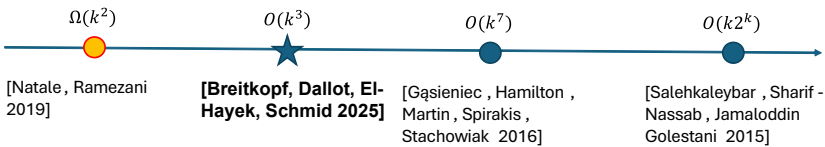


The CIRCLES protocol: Low-Space with a weakly fair scheduler

- *Weakly fair* scheduler: adversarial, but all pairs of agents must interact infinitely many times
- Focus on *space*: the *number* of states needed to solve the problem

The CIRCLES protocol: Low-Space with a weakly fair scheduler

- *Weakly fair* scheduler: adversarial, but all pairs of agents must interact infinitely many times
- Focus on *space*: the *number* of states needed to solve the problem



Comparison with the Binary Consensus Problem

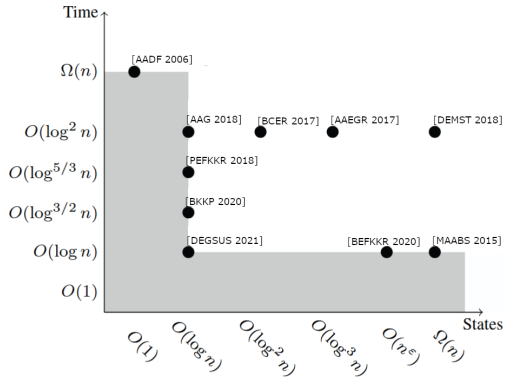
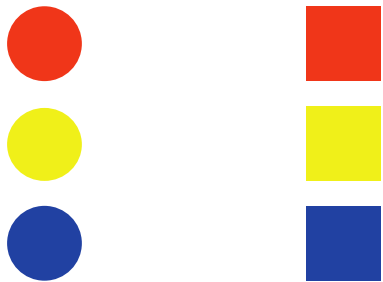


Figure 2: State of the art of the **Binary** consensus problem. Figure by [DEGSUS, FOCS'21]

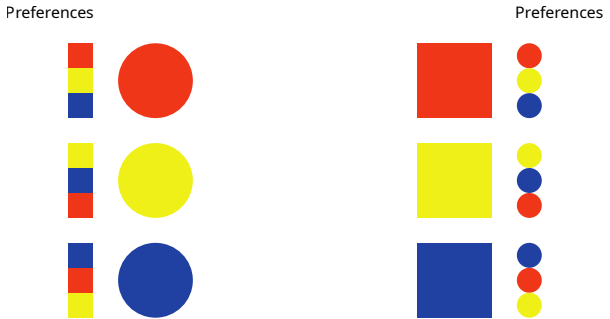
Stable Matchings/Hetero Marriages?

- Complete bipartite graph



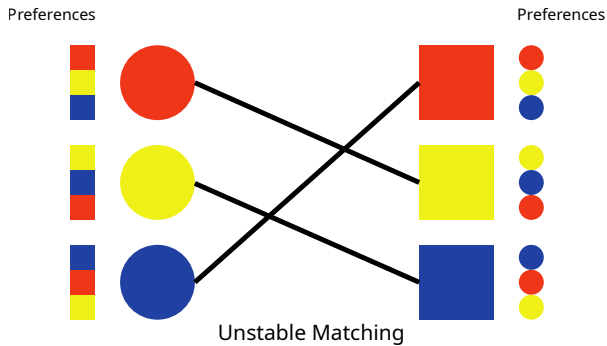
Stable Matchings/Hetero Marriages?

- Complete bipartite graph
- Each node has a preference list of the opposite nodes



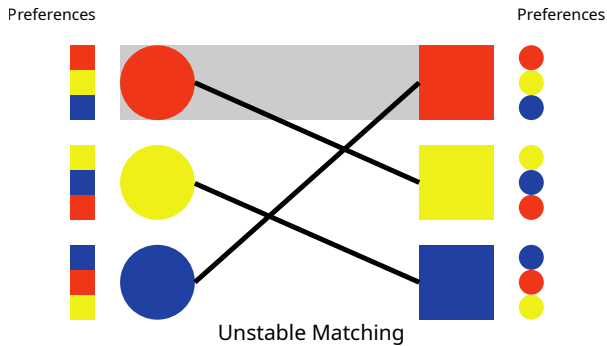
Stable Matchings/Hetero Marriages?

- Complete bipartite graph
- Each node has a preference list of the opposite nodes
- A matching is *unstable* if two opposite nodes prefer each other to the ones they are matched to



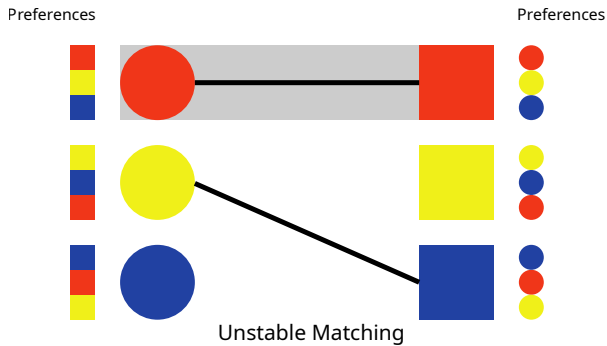
Stable Matchings/Hetero Marriages?

- Complete bipartite graph
- Each node has a preference list of the opposite nodes
- A matching is *unstable* if two opposite nodes prefer each other to the ones they are matched to



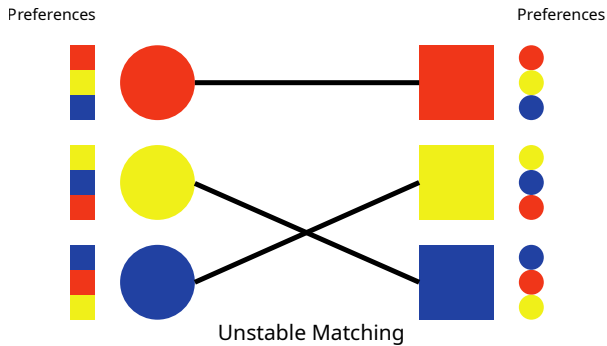
Stable Matchings/Hetero Marriages?

- Complete bipartite graph
- Each node has a preference list of the opposite nodes
- A matching is *unstable* if two opposite nodes prefer each other to the ones they are matched to



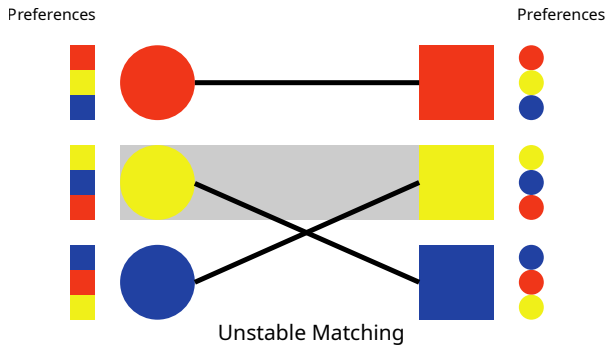
Stable Matchings/Hetero Marriages?

- Complete bipartite graph
- Each node has a preference list of the opposite nodes
- A matching is *unstable* if two opposite nodes prefer each other to the ones they are matched to



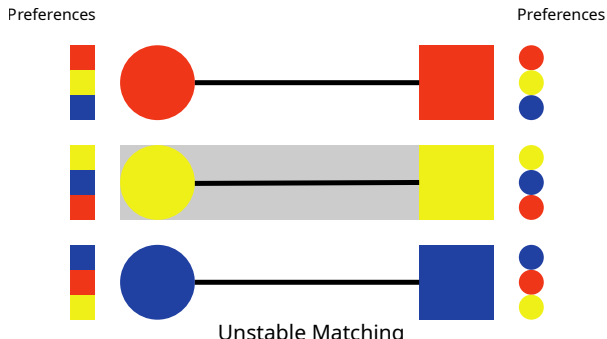
Stable Matchings/Hetero Marriages?

- Complete bipartite graph
- Each node has a preference list of the opposite nodes
- A matching is *unstable* if two opposite nodes prefer each other to the ones they are matched to



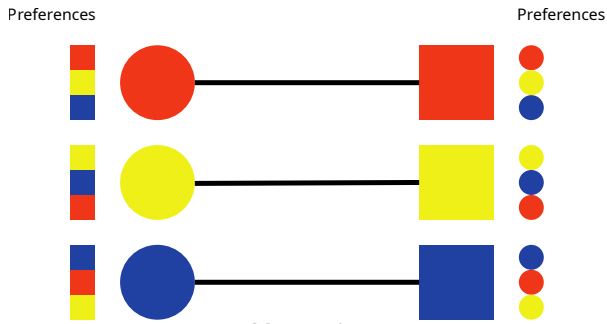
Stable Matchings/Hetero Marriages?

- Complete bipartite graph
- Each node has a preference list of the opposite nodes
- A matching is *unstable* if two opposite nodes prefer each other to the ones they are matched to



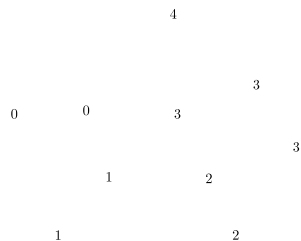
Stable Matchings/Hetero Marriages?

- Complete bipartite graph
- Each node has a preference list of the opposite nodes
- A matching is *unstable* if two opposite nodes prefer each other to the ones they are matched to
- A matching is otherwise a *stable* matching



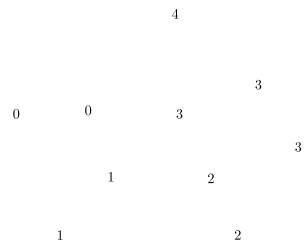
The CIRCLES protocol

- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i| , |j\rangle\}$



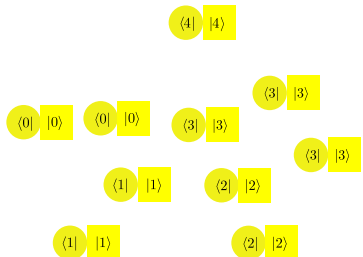
The CIRCLES protocol

- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i| , |j\rangle\}$
- The agents encode a *matching*: each agent is an edge of the matching.



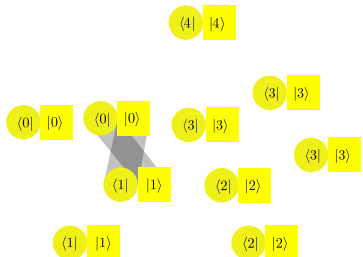
The CIRCLES protocol

- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i|, |j\rangle\}$
- The agents encode a *matching*: each agent is an edge of the matching.



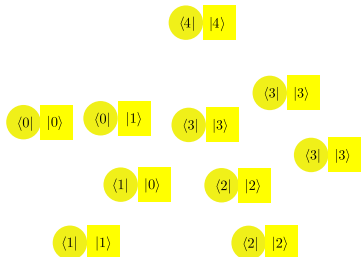
The CIRCLES protocol

- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i|, |j\rangle\}$
- The agents encode a *matching*: each agent is an edge of the matching.



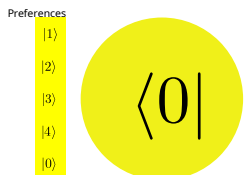
The CIRCLES protocol

- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i|, |j\rangle\}$
- The agents encode a *matching*: each agent is an edge of the matching.



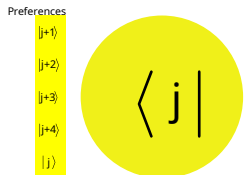
The CIRCLES protocol

- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i|, |j\rangle\}$
- The agents encode a *matching*: each agent is an edge of the matching.
- $\langle j|$ order of preference: $|j+1\rangle, \ldots, |k-1\rangle, |0\rangle, \ldots, |j\rangle$



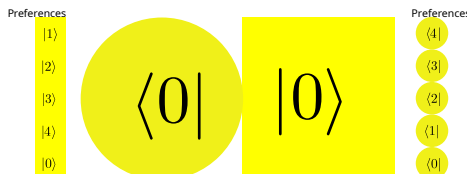
The CIRCLES protocol

- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i|, |j\rangle\}$
- The agents encode a *matching*: each agent is an edge of the matching.
- $\langle j|$ order of preference: $|j+1\rangle, \ldots, |k-1\rangle, |0\rangle, \ldots, |j\rangle$



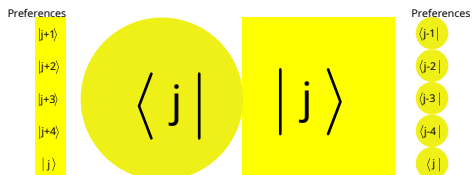
The CIRCLES protocol

- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i|, |j\rangle\}$
- The agents encode a *matching*: each agent is an edge of the matching.
- $\langle j|$ order of preference: $|j+1\rangle, \ldots, |k-1\rangle, |0\rangle, \ldots, |j\rangle$
- $|j\rangle$ order of preference: $|j-1\rangle, \ldots, |0\rangle, |k-1\rangle, \ldots, |j\rangle$



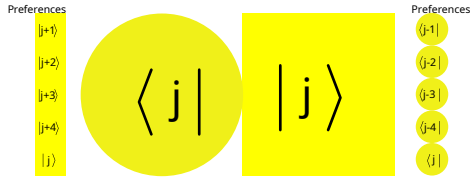
The CIRCLES protocol

- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i|, |j\rangle\}$
- The agents encode a *matching*: each agent is an edge of the matching.
- $\langle j|$ order of preference: $|j+1\rangle, \ldots, |k-1\rangle, |0\rangle, \ldots, |j\rangle$
- $|j\rangle$ order of preference: $|j-1\rangle, \ldots, |0\rangle, |k-1\rangle, \ldots, |j\rangle$



The CIRCLES protocol

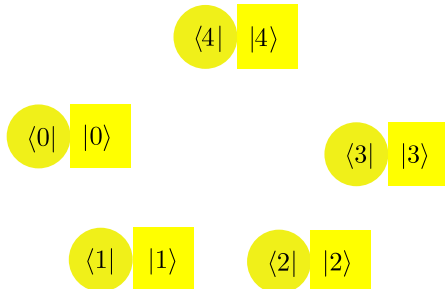
- Input function: Each agent gets two nodes of the bipartite graph: $\mathcal{I}(i) = \{\langle i| , |j\rangle\}$
- The agents encode a *matching*: each agent is an edge of the matching.
- $\langle j|$ order of preference: $|j+1\rangle, \ldots, |k-1\rangle, |0\rangle, \ldots, |j\rangle$
- $|j\rangle$ order of preference: $|j-1\rangle, \ldots, |0\rangle, |k-1\rangle, \ldots, |j\rangle$



- Output function: $\mathcal{O}(\langle i|j\rangle) = \begin{cases} i & \text{if } i = j \\ \perp & \text{otherwise} \end{cases}$

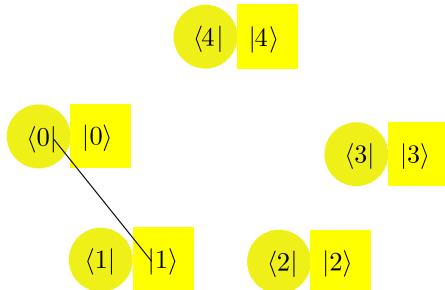
A small example

- Let us start with a simple case



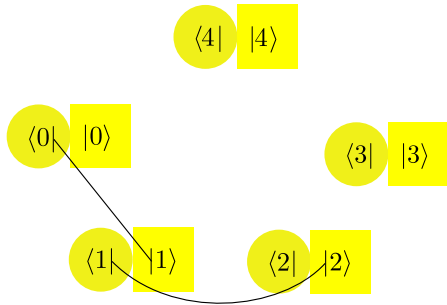
A small example

- Let us start with a simple case



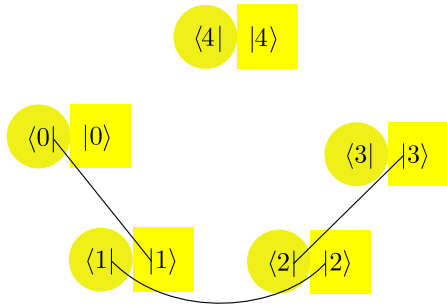
A small example

- Let us start with a simple case



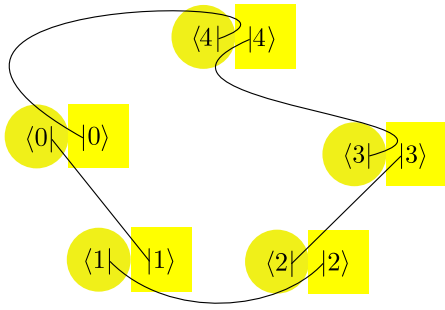
A small example

- Let us start with a simple case



A small example

- Let us start with a simple case



Stable Matchings

- There exists a stable marriage

Stable Matchings

- There exists a stable marriage $\Rightarrow$ There exists a stable configuration

Stable Matchings

- There exists a stable marriage $\Rightarrow$ There exists a stable configuration
- Strategy:
  1. Find a stable configuration C

Stable Matchings

- There exists a stable marriage $\Rightarrow$ There exists a stable configuration
- Strategy:
  1. Find a stable configuration C
  2. Give a potential function

Stable Matchings

- There exists a stable marriage $\Rightarrow$ There exists a stable configuration
- Strategy:
  1. Find a stable configuration C
  2. Give a potential function
  3. Show that the transition function strictly decreases the potential when there is an exchange

Stable Matchings

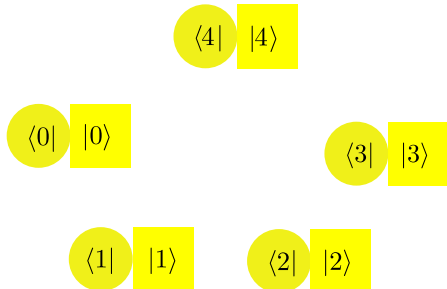
- There exists a stable marriage $\Rightarrow$ There exists a stable configuration
- Strategy:
  1. Find a stable configuration C
  2. Give a potential function
  3. Show that the transition function strictly decreases the potential when there is an exchange
  4. Show that any configuration different from C has a larger potential than C, and that its potential can be reduced by choosing an interaction

Stable Matchings

- There exists a stable marriage $\Rightarrow$ There exists a stable configuration
- Strategy:
  1. Find a stable configuration C
  2. Give a potential function
  3. Show that the transition function strictly decreases the potential when there is an exchange
  4. Show that any configuration different from C has a larger potential than C, and that its potential can be reduced by choosing an interaction
- We show that there is only one stable marriage.

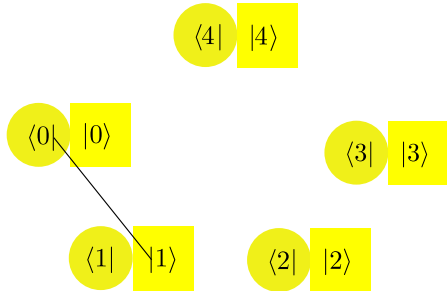
Finding *a* stable configuration

- Let us start with a simple case



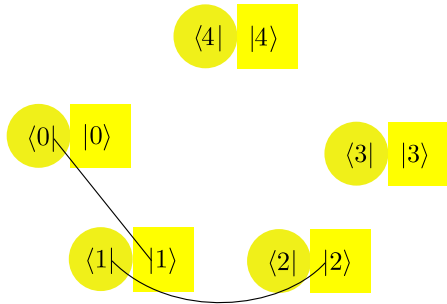
Finding *a* stable configuration

- Let us start with a simple case



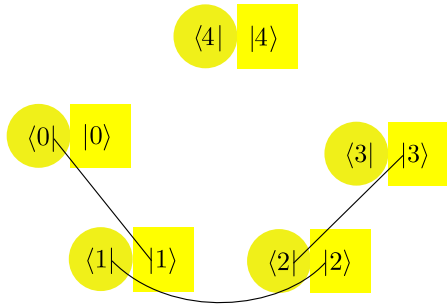
Finding *a* stable configuration

- Let us start with a simple case



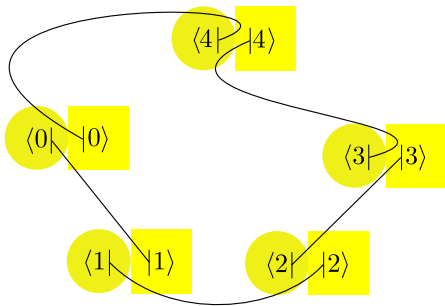
Finding *a* stable configuration

- Let us start with a simple case



Finding *a* stable configuration

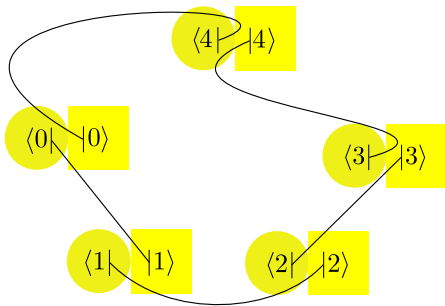
- Let us start with a simple case



- This is a *Circle*!

Finding *a* stable configuration

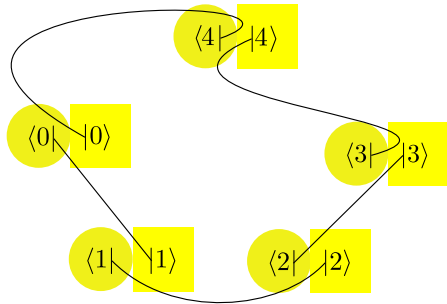
- Let us start with a simple case



- This is a *Circle*!

Finding *a* stable configuration

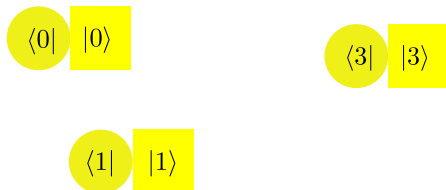
- Let us start with a simple case



- This is a *Circle*!
- This is now set in stone: no interaction will change any of these combinations.

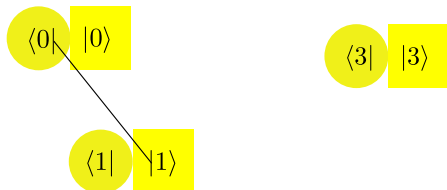
Finding *a* stable configuration

- What if the circle is incomplete?



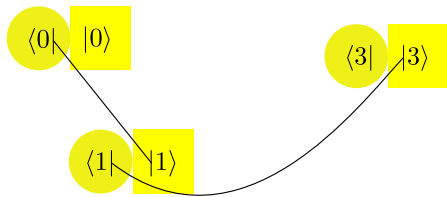
Finding *a* stable configuration

- What if the circle is incomplete?



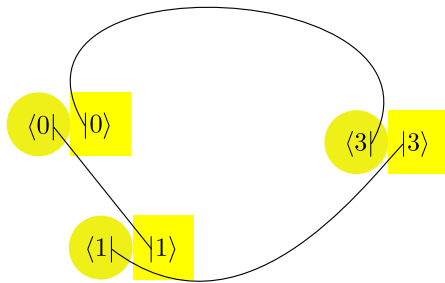
Finding *a* stable configuration

- What if the circle is incomplete?



Finding *a* stable configuration

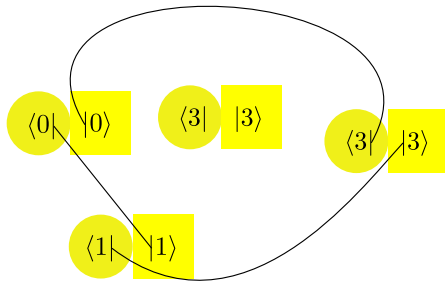
- What if the circle is incomplete?



- This is *also* a Circle

Finding *a* stable configuration

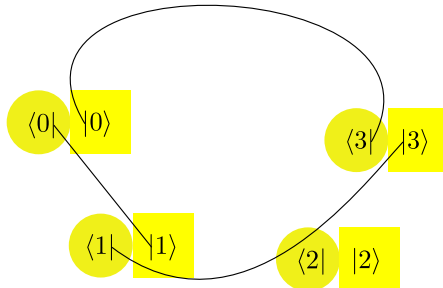
- What if the circle is incomplete?



- This is *also* a Circle
- Adding an element that is already in the Circle does not affect it

Finding *a* stable configuration

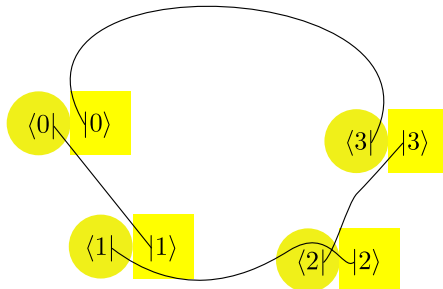
- What if the circle is incomplete?



- This is *also* a Circle
- Adding an element that is already in the Circle does not affect it
- An element not in the circle gets added

Finding *a* stable configuration

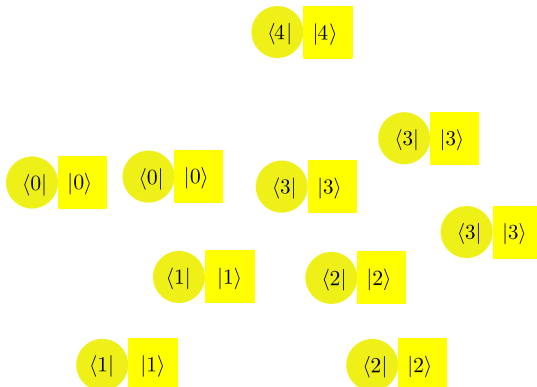
- What if the circle is incomplete?



- This is *also* a Circle
- Adding an element that is already in the Circle does not affect it
- An element not in the circle gets added

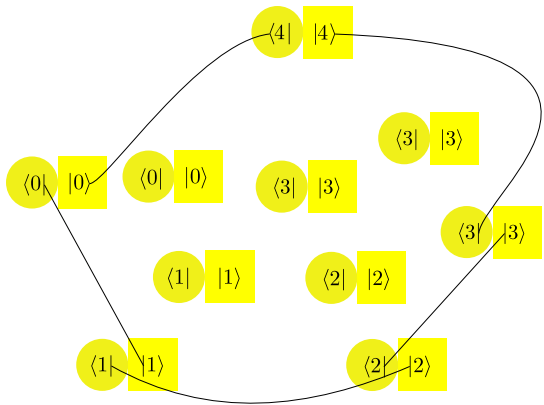
Finding *a* stable configuration

- Let us add some complexity



Finding *a* stable configuration

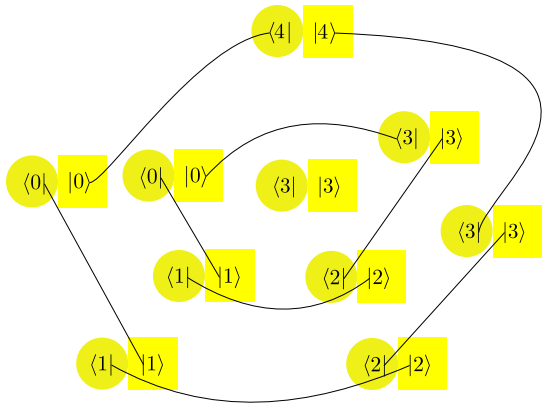
- Let us add some complexity



- We start with the biggest possible circle

Finding *a* stable configuration

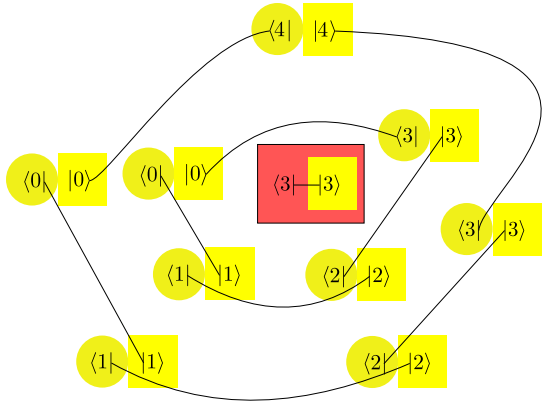
- Let us add some complexity



- We start with the biggest possible circle, then the next

Finding *a* stable configuration

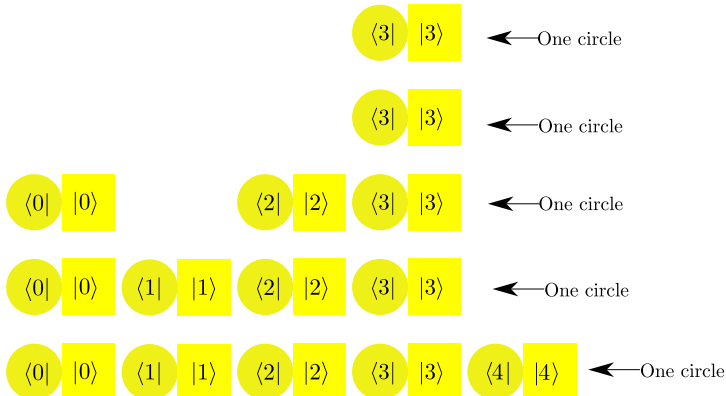
- Let us add some complexity



- We start with the biggest possible circle, then the next
- The final circle has one element in it: the majority element

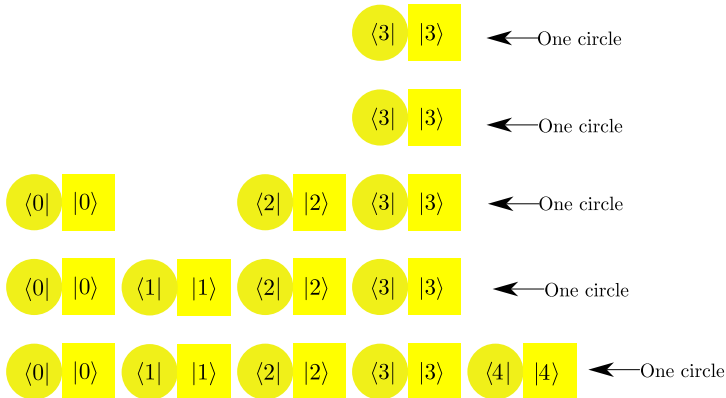
Finding *a* stable configuration

- Switching to a bar chart view



Finding *a* stable configuration

- Switching to a bar chart view



- We have as many circles as the majority element

Is this configuration unique?

- *weight* function: $w(\langle i| \, , |j\rangle) = \begin{cases} k & \text{if } i = j \\ (j - i) \bmod k & \text{otherwise (a number beween } 1 \text{ and } k - 1) \end{cases}$

Is this configuration unique?

- *weight* function: $w(\langle i| \, , |j\rangle) = \begin{cases} k & \text{if } i = j \\ (j - i) \bmod k & \text{otherwise (a number beween } 1 \text{ and } k - 1) \end{cases}$
- We define the potential function: Given a configuration C, let $w_1(C), w_2(C) \ldots w_n(C)$ be the weights of each agent sorted in increasing order. The potential function is:

$$g(C) = \omega^{n-1} \cdot w_1(C) + \omega^{n-2} \cdot w_2(C) + \cdots + \omega \cdot w_{n-1} + 1 \cdot w_n$$

Where ω the smallest ordinal number greater than all the integers.

Is this configuration unique?

- *weight* function: $w(\langle i| \, , |j\rangle) = \begin{cases} k & \text{if } i = j \\ (j - i) \bmod k & \text{otherwise (a number beween } 1 \text{ and } k - 1) \end{cases}$
- We define the potential function: Given a configuration C, let $w_1(C), w_2(C) \ldots w_n(C)$ be the weights of each agent sorted in increasing order. The potential function is:

$$g(C) = \omega^{n-1} \cdot w_1(C) + \omega^{n-2} \cdot w_2(C) + \cdots + \omega \cdot w_{n-1} + 1 \cdot w_n$$

  Where ω the smallest ordinal number greater than all the integers.
- We show that two agents exchange "squares" if and only if the potential strictly decreases.

Is this configuration unique?

- *weight* function: $w(\langle i| \, , |j\rangle) = \begin{cases} k & \text{if } i = j \\ (j - i) \bmod k & \text{otherwise (a number beween } 1 \text{ and } k - 1) \end{cases}$
- We define the potential function: Given a configuration C, let $w_1(C), w_2(C) \ldots w_n(C)$ be the weights of each agent sorted in increasing order. The potential function is:

$$g(C) = \omega^{n-1} \cdot w_1(C) + \omega^{n-2} \cdot w_2(C) + \cdots + \omega \cdot w_{n-1} + 1 \cdot w_n$$

  Where ω the smallest ordinal number greater than all the integers.
- We show that two agents exchange "squares" if and only if the potential strictly decreases.
- We then show that if the configuration does not correspond to the Circles, there exist two agents that can exchange "squares"

Is this configuration unique?

- *weight* function: $w(\langle i| \, , |j\rangle) = \begin{cases} k & \text{if } i = j \\ (j - i) \bmod k & \text{otherwise (a number beween } 1 \text{ and } k - 1) \end{cases}$
- We define the potential function: Given a configuration C, let $w_1(C), w_2(C) \ldots w_n(C)$ be the weights of each agent sorted in increasing order. The potential function is:

$$g(C) = \omega^{n-1} \cdot w_1(C) + \omega^{n-2} \cdot w_2(C) + \cdots + \omega \cdot w_{n-1} + 1 \cdot w_n$$

  Where ω the smallest ordinal number greater than all the integers.
- We show that two agents exchange "squares" if and only if the potential strictly decreases.
- We then show that if the configuration does not correspond to the Circles, there exist two agents that can exchange "squares"
  - By looking at the first circle that is incomplete, starting from the base of the bar chart

Is this configuration unique?

- *weight* function: $w(\langle i| \, , |j\rangle) = \begin{cases} k & \text{if } i = j \\ (j - i) \bmod k & \text{otherwise (a number beween } 1 \text{ and } k - 1) \end{cases}$
- We define the potential function: Given a configuration C, let $w_1(C), w_2(C) \ldots w_n(C)$ be the weights of each agent sorted in increasing order. The potential function is:

$$g(C) = \omega^{n-1} \cdot w_1(C) + \omega^{n-2} \cdot w_2(C) + \cdots + \omega \cdot w_{n-1} + 1 \cdot w_n$$

  Where ω the smallest ordinal number greater than all the integers.
- We show that two agents exchange "squares" if and only if the potential strictly decreases.
- We then show that if the configuration does not correspond to the Circles, there exist two agents that can exchange "squares"
  - By looking at the first circle that is incomplete, starting from the base of the bar chart
- This shows that every other configuration has potential larger than than the circles configuration, and is unstable

Is this configuration unique?

- *weight* function: $w(\langle i| \, , |j\rangle) = \begin{cases} k & \text{if } i = j \\ (j - i) \bmod k & \text{otherwise (a number beween } 1 \text{ and } k - 1) \end{cases}$
- We define the potential function: Given a configuration C, let $w_1(C), w_2(C) \ldots w_n(C)$ be the weights of each agent sorted in increasing order. The potential function is:

$$g(C) = \omega^{n-1} \cdot w_1(C) + \omega^{n-2} \cdot w_2(C) + \cdots + \omega \cdot w_{n-1} + 1 \cdot w_n$$

  Where ω the smallest ordinal number greater than all the integers.
- We show that two agents exchange "squares" if and only if the potential strictly decreases.
- We then show that if the configuration does not correspond to the Circles, there exist two agents that can exchange "squares"
  - By looking at the first circle that is incomplete, starting from the base of the bar chart
- This shows that every other configuration has potential larger than than the circles configuration, and is unstable
- The potential function argument also ensures we do not end up in an infinite loop.

Generalisations and future work

Generalisations and future work

- This gives a $O(k^3)$-states algorithms: k possible values for $\langle i|$, k for $|j\rangle$, and k for the output. Can we close the gap to the $\Omega(k^2)$ lower bound?

Generalisations and future work

- This gives a $O(k^3)$-states algorithms: k possible values for $\langle i|$, k for $|j\rangle$, and k for the output. Can we close the gap to the $\Omega(k^2)$ lower bound?
- We can also deal with ties, and we can compute the full ranking!

Generalisations and future work

- This gives a $O(k^3)$-states algorithms: k possible values for $\langle i|$, k for $|j\rangle$, and k for the output. Can we close the gap to the $\Omega(k^2)$ lower bound?
- We can also deal with ties, and we can compute the full ranking!
- Can we use stable marriage with different preferences to compute different functions in population protocols?

Generalisations and future work

- This gives a $O(k^3)$-states algorithms: k possible values for $\langle i|$, k for $|j\rangle$, and k for the output. Can we close the gap to the $\Omega(k^2)$ lower bound?
- We can also deal with ties, and we can compute the full ranking!
- Can we use stable marriage with different preferences to compute different functions in population protocols?
- Interesting result emerge from simple algorithm: is there a relationship to swarm intelligence?

Generalisations and future work

- This gives a $O(k^3)$-states algorithms: k possible values for $\langle i|$, k for $|j\rangle$, and k for the output. Can we close the gap to the $\Omega(k^2)$ lower bound?
- We can also deal with ties, and we can compute the full ranking!
- Can we use stable marriage with different preferences to compute different functions in population protocols?
- Interesting result emerge from simple algorithm: is there a relationship to swarm intelligence?
- Explore other problems like partition or counting

Generalisations and future work

- This gives a $O(k^3)$-states algorithms: k possible values for $\langle i|$, k for $|j\rangle$, and k for the output. Can we close the gap to the $\Omega(k^2)$ lower bound?
- We can also deal with ties, and we can compute the full ranking!
- Can we use stable marriage with different preferences to compute different functions in population protocols?
- Interesting result emerge from simple algorithm: is there a relationship to swarm intelligence?
- Explore other problems like partition or counting
- Explore population protocols *on graphs*, that is, when interactions between agents are restricted to an underlying graph, where agents are nodes and possible interactions are edges.