

A subexponential lower bound for the Random Facet algorithm for Parity Games

Oliver Friedmann¹ Thomas Dueholm Hansen² Uri Zwick³

¹ Department of Computer Science,
University of Munich, Germany.

² Center for Algorithmic Game Theory,
Department of Computer Science,
Aarhus University, Denmark.

³ School of Computer Science,
Tel Aviv University, Israel.

January 23, 2011



The RANDOMFACET algorithm

- RANDOMFACET: Randomized pivoting rule for the simplex algorithm for linear programming (*LP-type* problems).

The RANDOMFACET algorithm

- RANDOMFACET: Randomized pivoting rule for the simplex algorithm for linear programming (*LP-type* problems).
- Introduced by Matoušek, Sharir and Welzl (1992). A different variant was introduced by Kalai (1992).

The RANDOMFACET algorithm

- RANDOMFACET: Randomized pivoting rule for the simplex algorithm for linear programming (*LP-type* problems).
- Introduced by Matoušek, Sharir and Welzl (1992). A different variant was introduced by Kalai (1992).
- Terminates in a subexponential number of steps: $2^{O(\sqrt{n \log m})}$, (n variables, m constraints).

The RANDOMFACET algorithm

- RANDOMFACET: Randomized pivoting rule for the simplex algorithm for linear programming (*LP-type* problems).
- Introduced by Matoušek, Sharir and Welzl (1992). A different variant was introduced by Kalai (1992).
- Terminates in a subexponential number of steps: $2^{O(\sqrt{n \log m})}$, (n variables, m constraints).
- This is the best known upper bound in n and m for any algorithm for the problem.

The RANDOMFACET algorithm

- RANDOMFACET: Randomized pivoting rule for the simplex algorithm for linear programming (*LP-type* problems).
- Introduced by Matoušek, Sharir and Welzl (1992). A different variant was introduced by Kalai (1992).
- Terminates in a subexponential number of steps: $2^{O(\sqrt{n \log m})}$, (n variables, m constraints).
- This is the best known upper bound in n and m for any algorithm for the problem.
- Matoušek (1994): $2^{\Omega(\sqrt{n})}$ lower bound in abstract setting.

The RANDOMFACET algorithm

- RANDOMFACET: Randomized pivoting rule for the simplex algorithm for linear programming (*LP-type* problems).
- Introduced by Matoušek, Sharir and Welzl (1992). A different variant was introduced by Kalai (1992).
- Terminates in a subexponential number of steps: $2^{O(\sqrt{n \log m})}$, (n variables, m constraints).
- This is the best known upper bound in n and m for any algorithm for the problem.
- Matoušek (1994): $2^{\Omega(\sqrt{n})}$ lower bound in abstract setting.
- Until recently a candidate for solving linear programs in strongly polynomial time: Disproved by the continuation of the work presented here.

- Stochastic games were introduced by Shapley (1953).

- Stochastic games were introduced by Shapley (1953).
- Turn-based stochastic games are played on directed graphs with vertices controlled by **maximizer**, **minimizer** and **nature**.

- Stochastic games were introduced by Shapley (1953).
- Turn-based stochastic games are played on directed graphs with vertices controlled by **maximizer**, **minimizer** and **nature**.
- Major open problem, first stated by Condon (1992): The problem of solving turn-based stochastic games is in **NP** \cap **coNP**, but no polynomial time algorithm is known.

- Stochastic games were introduced by Shapley (1953).
- Turn-based stochastic games are played on directed graphs with vertices controlled by **maximizer**, **minimizer** and **nature**.
- Major open problem, first stated by Condon (1992): The problem of solving turn-based stochastic games is in **NP** \cap **coNP**, but no polynomial time algorithm is known.
- Halman (2007): Turn-based stochastic games are of LP-type.
 - Vertices correspond to variables and edges correspond to constraints.

- Stochastic games were introduced by Shapley (1953).
- Turn-based stochastic games are played on directed graphs with vertices controlled by **maximizer**, **minimizer** and **nature**.
- Major open problem, first stated by Condon (1992): The problem of solving turn-based stochastic games is in **$\text{NP} \cap \text{coNP}$** , but no polynomial time algorithm is known.
- Halman (2007): Turn-based stochastic games are of LP-type.
 - Vertices correspond to variables and edges correspond to constraints.
- The **RANDOMFACET** algorithm is the fastest known algorithm for solving turn-based stochastic games.

1 player:

- Markov decision processes (MDPs): **maximizer** and **nature**.
Introduced by Bellman (1957).
 - Manne (1960): MDPs can be solved by linear programming.
 - No known strongly polynomial time algorithm for MDPs.

1 player:

- Markov decision processes (MDPs): **maximizer** and **nature**.
Introduced by Bellman (1957).
 - Manne (1960): MDPs can be solved by linear programming.
 - No known strongly polynomial time algorithm for MDPs.
- Deterministic MDPs: **maximizer**.
 - Solves the minimum mean cost cycle problem.
 - Karp (1978): $O(nm)$ time algorithm, n vertices and m edges.

1 player:

- Markov decision processes (MDPs): **maximizer** and **nature**.
Introduced by Bellman (1957).
 - Manne (1960): MDPs can be solved by linear programming.
 - No known strongly polynomial time algorithm for MDPs.
- Deterministic MDPs: **maximizer**.
 - Solves the minimum mean cost cycle problem.
 - Karp (1978): $O(nm)$ time algorithm, n vertices and m edges.

No vertices controlled by nature:

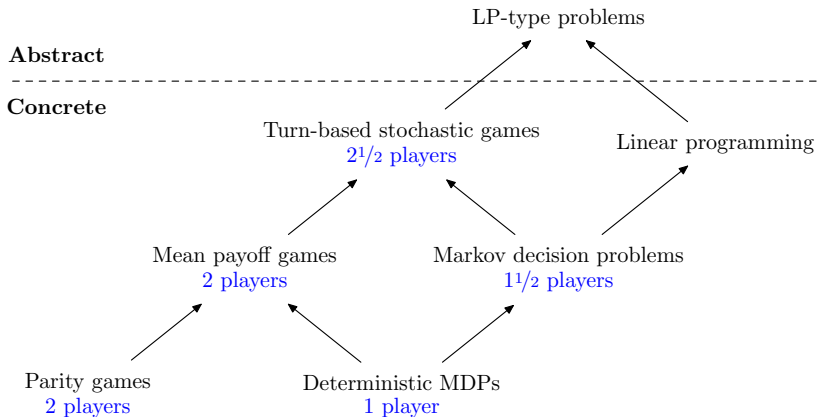
- Mean payoff games: **maximizer** and **minimizer**.

1 player:

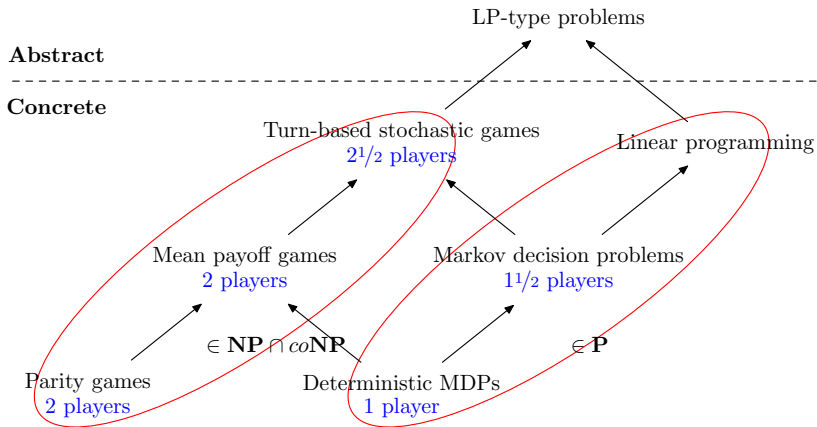
- Markov decision processes (MDPs): **maximizer** and **nature**. Introduced by Bellman (1957).
 - Manne (1960): MDPs can be solved by linear programming.
 - No known strongly polynomial time algorithm for MDPs.
- Deterministic MDPs: **maximizer**.
 - Solves the minimum mean cost cycle problem.
 - Karp (1978): $O(nm)$ time algorithm, n vertices and m edges.

No vertices controlled by nature:

- Mean payoff games: **maximizer** and **minimizer**.
- Parity games: **maximizer** and **minimizer**, special structure.
 - Equivalent to the problem of μ -calculus model checking.



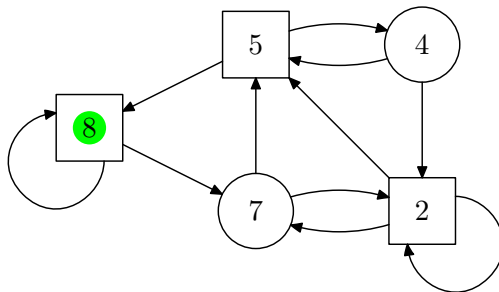
Overview



Theorem

The RANDOMFACET algorithm may require $2^{\Omega(\sqrt{n}/\log n)}$ expected steps to solve n -state parity games, mean payoff games and turn-based stochastic games.

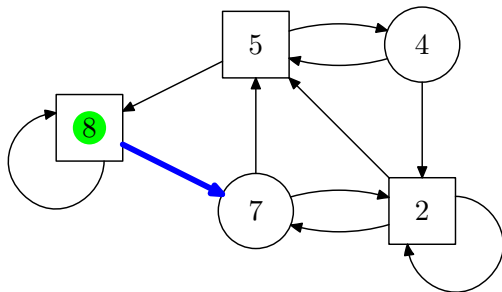
Parity games



- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8

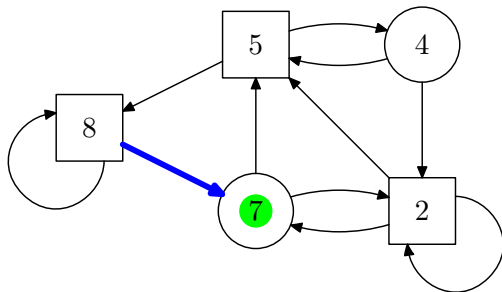
Parity games



- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8

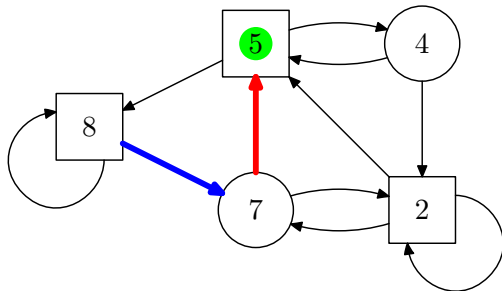
Parity games



- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8 , 7

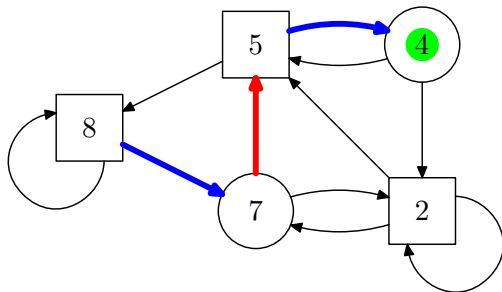
Parity games



- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8 , 7 , 5

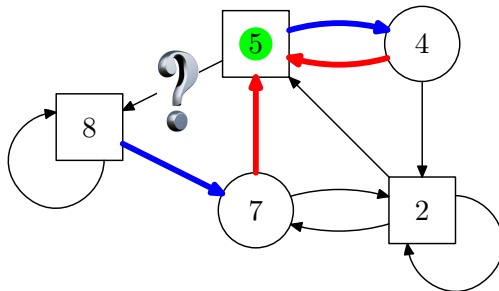
Parity games



- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8 , 7 , 5 , 4

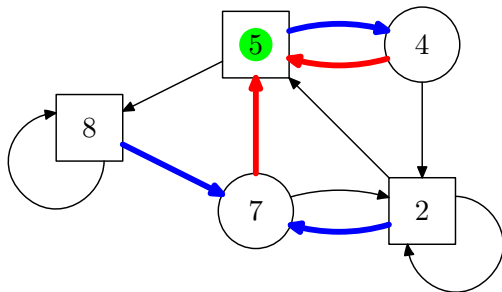
Parity games



- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8 , 7 , 5 , 4 , 5

Parity games

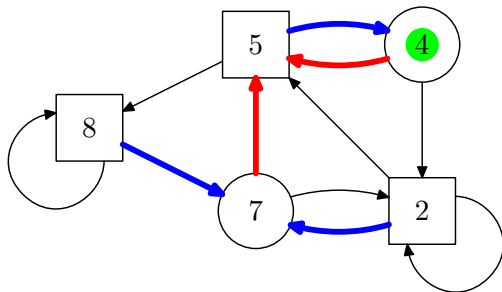


- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8 , 7 , 5 , 4 , 5

- A (positional) strategy, σ or τ , is an outgoing edge from each vertex controlled by the corresponding player.

Parity games

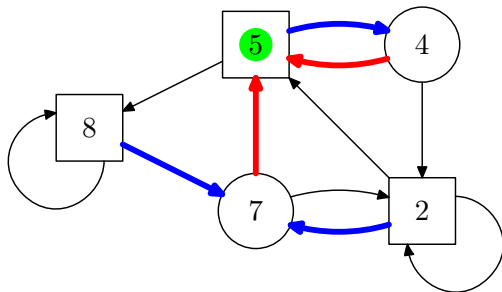


- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8 , 7 , 5 , 4 , 5 , 4

- A (positional) strategy, σ or τ , is an outgoing edge from each vertex controlled by the corresponding player.

Parity games

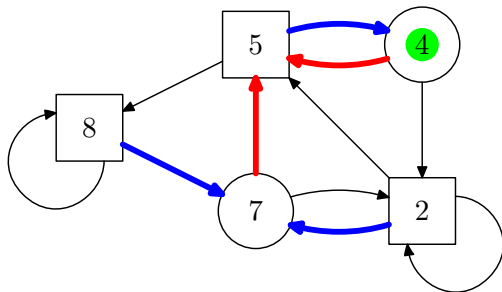


- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8 , 7 , 5 , 4 , 5 , 4 , 5

- A (positional) strategy, σ or τ , is an outgoing edge from each vertex controlled by the corresponding player.

Parity games



- **EVEN** (circle) wins if the largest priority seen infinitely often is even, **ODD** (square) wins otherwise:

Observed priorities: 8 , 7 , 5 , 4 , 5 , 4 , 5 , 4 , ...

- A (positional) strategy, σ or τ , is an outgoing edge from each vertex controlled by the corresponding player.

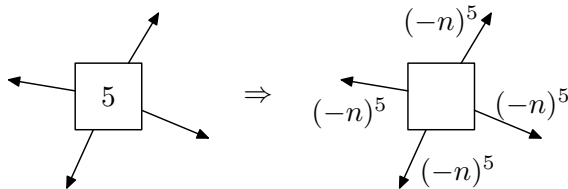
- Priorities are replaced by rewards on edges. Players take the roles of **maximizer** and **minimizer**.

- Priorities are replaced by rewards on edges. Players take the roles of **maximizer** and **minimizer**.
- The value $\text{VAL}_{\sigma, \tau}(v)$ of a vertex v is the average reward of the cycle reached from v when moving according to σ and τ .

- Priorities are replaced by rewards on edges. Players take the roles of **maximizer** and **minimizer**.
- The value $\text{VAL}_{\sigma, \tau}(v)$ of a vertex v is the average reward of the cycle reached from v when moving according to σ and τ .
- The path leading to the cycle is of secondary importance.

Mean payoff games

- Priorities are replaced by rewards on edges. Players take the roles of **maximizer** and **minimizer**.
- The value $\text{VAL}_{\sigma, \tau}(v)$ of a vertex v is the average reward of the cycle reached from v when moving according to σ and τ .
- The path leading to the cycle is of secondary importance.
- Reduction from parity games:



A cycle has positive value iff its largest priority is even.

- In our lower bound examples we always reach a cycle with value zero (essentially).

- In our lower bound examples we always reach a cycle with value zero (essentially).
- The path leading to the cycle is then the primary focus.

- In our lower bound examples we always reach a cycle with value zero (essentially).
- The path leading to the cycle is then the primary focus.
- For simplicity I will use $\text{VAL}_{\sigma, \tau}(v)$ to denote the total sum of rewards on the path to the cycle.

- We generally assume that the **minimizer** plays an optimal counter-strategy:

$$\text{VAL}_{\sigma}(v) = \min_{\tau} \text{VAL}_{\sigma, \tau}(v)$$

- We generally assume that the **minimizer** plays an optimal counter-strategy:

$$\text{VAL}_\sigma(v) = \min_{\tau} \text{VAL}_{\sigma,\tau}(v)$$

- σ is optimal from v if for all σ' , $\text{VAL}_\sigma(v) \geq \text{VAL}_{\sigma'}(v)$.

- We generally assume that the **minimizer** plays an optimal counter-strategy:

$$\text{VAL}_\sigma(v) = \min_{\tau} \text{VAL}_{\sigma,\tau}(v)$$

- σ is optimal from v if for all σ' , $\text{VAL}_\sigma(v) \geq \text{VAL}_{\sigma'}(v)$.
- σ^* is optimal if it is optimal from all v .

- We generally assume that the **minimizer** plays an optimal counter-strategy:

$$\text{VAL}_\sigma(v) = \min_{\tau} \text{VAL}_{\sigma,\tau}(v)$$

- σ is optimal from v if for all σ' , $\text{VAL}_\sigma(v) \geq \text{VAL}_{\sigma'}(v)$.
- σ^* is optimal if it is optimal from all v .
- Shapley (1957): Optimal positional strategies are guaranteed to exist.

- We generally assume that the **minimizer** plays an optimal counter-strategy:

$$\text{VAL}_\sigma(v) = \min_{\tau} \text{VAL}_{\sigma,\tau}(v)$$

- σ is optimal from v if for all σ' , $\text{VAL}_\sigma(v) \geq \text{VAL}_{\sigma'}(v)$.
- σ^* is optimal if it is optimal from all v .
- Shapley (1957): Optimal positional strategies are guaranteed to exist.
- An edge (u, v) is an improving switch w.r.t. σ if the value of u is improved by switching to (u, v) :

$$\text{VAL}_{\sigma[(u,v)]}(u) \geq \text{VAL}_\sigma(u)$$

- We generally assume that the **minimizer** plays an optimal counter-strategy:

$$\text{VAL}_\sigma(v) = \min_{\tau} \text{VAL}_{\sigma,\tau}(v)$$

- σ is optimal from v if for all σ' , $\text{VAL}_\sigma(v) \geq \text{VAL}_{\sigma'}(v)$.
- σ^* is optimal if it is optimal from all v .
- Shapley (1957): Optimal positional strategies are guaranteed to exist.
- An edge (u, v) is an improving switch w.r.t. σ if the value of u is improved by switching to (u, v) :

$$\text{VAL}_{\sigma[(u,v)]}(u) \geq \text{VAL}_\sigma(u)$$

- σ^* is optimal iff there are no improving switches.

The RANDOMFACET algorithm

Function RANDOMFACET(G, σ)

if $E_0 = \sigma$ **then**

 | **return** σ

else

 | Choose $e \in E_0 \setminus \sigma$ uniformly at random

 | $\sigma' \leftarrow \text{RANDOMFACET}(G \setminus \{e\}, \sigma)$

 | **if** e is improving switch w.r.t. σ' **then**

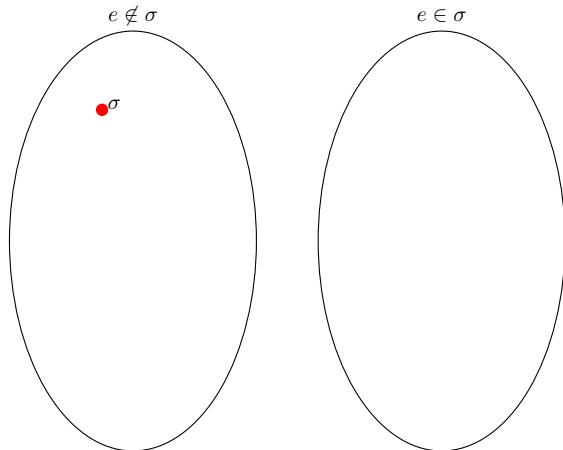
 | $\sigma'' \leftarrow \sigma'[e]$

 | **return** RANDOMFACET(G, σ'')

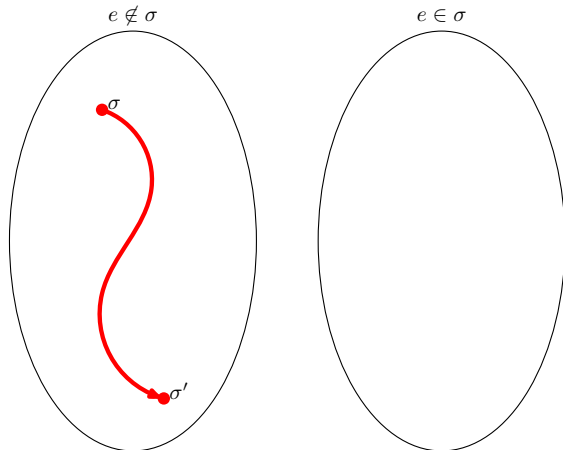
 | **else**

 | **return** σ'

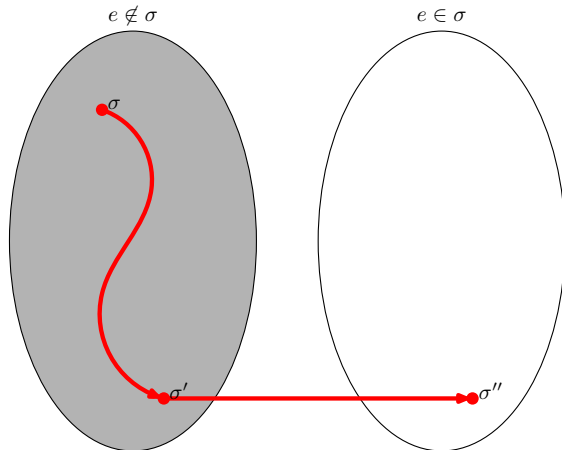
Example: Binary choices



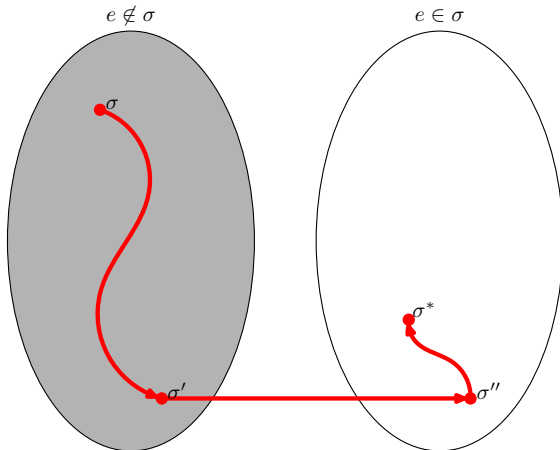
Example: Binary choices



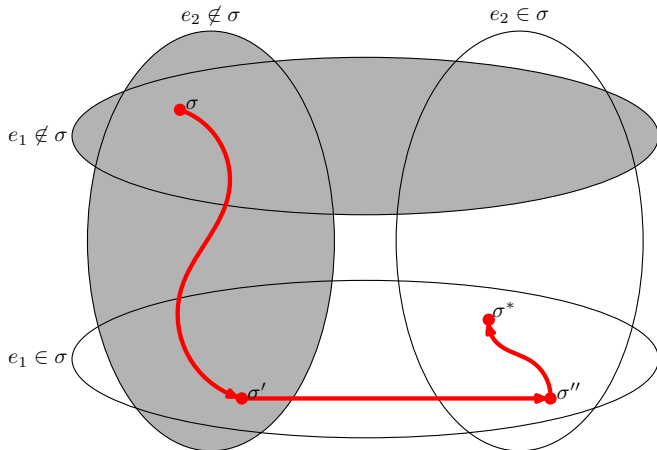
Example: Binary choices



Example: Binary choices



Example: Binary choices



$$\text{VAL}(G \setminus \{e_1\}) \leq \text{VAL}(G \setminus \{e_2\}) \leq \dots \leq \text{VAL}(G \setminus \{e_n\})$$

Example: Binary choices

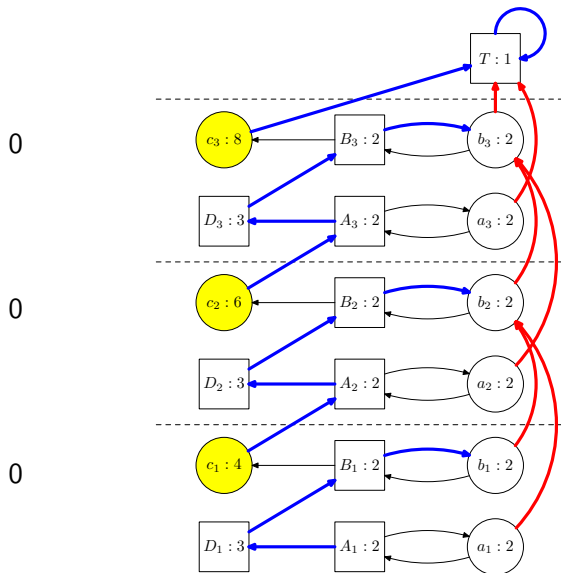
- The number of steps is upper bounded by:

$$f(0) = 1$$

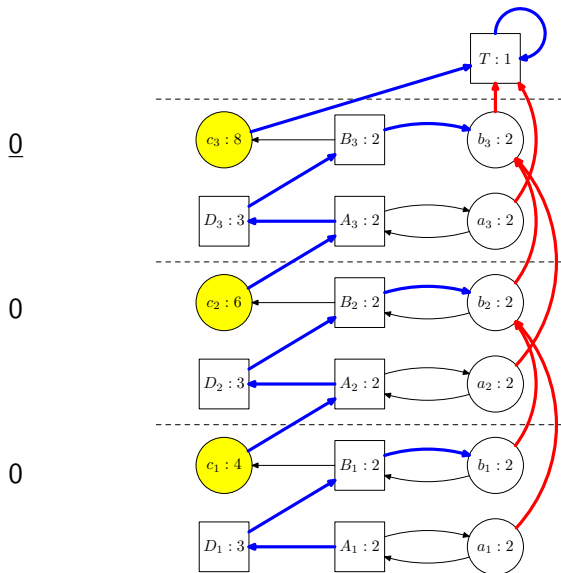
$$f(n) = f(n-1) + \frac{1}{n} \sum_{i=0}^{n-1} f(i) \quad \text{for } n > 0$$

- The recurrence is bounded by: $f(n) = 2^{\Theta(\sqrt{n})}$

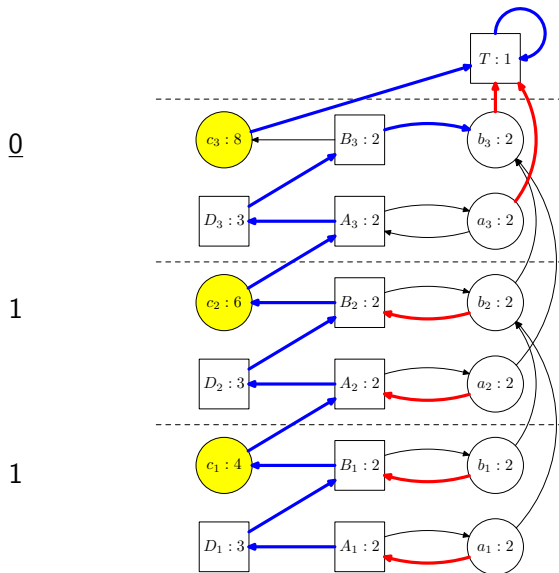
The construction



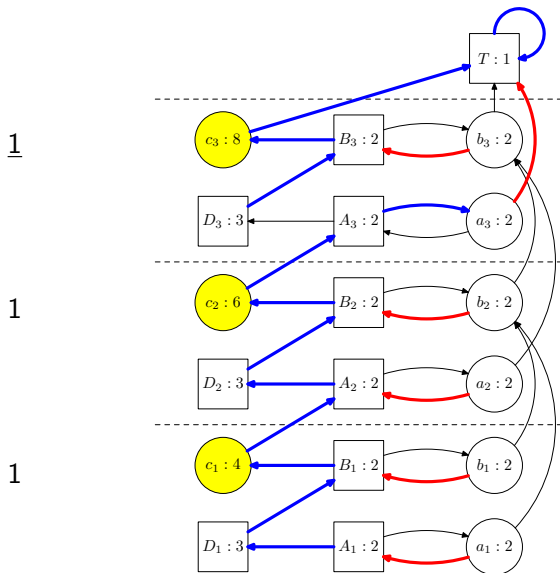
The construction



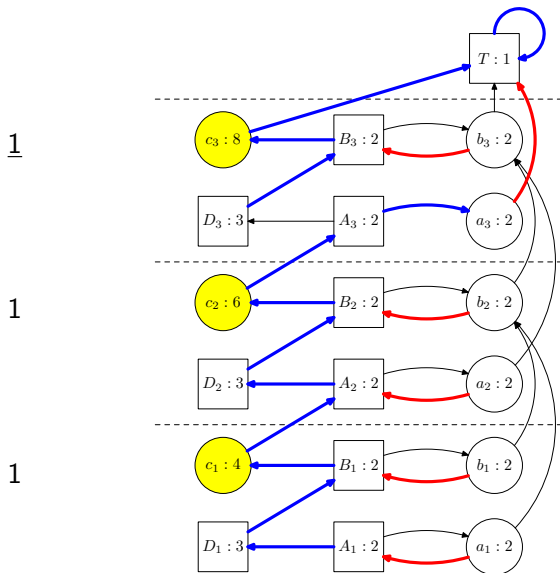
The construction



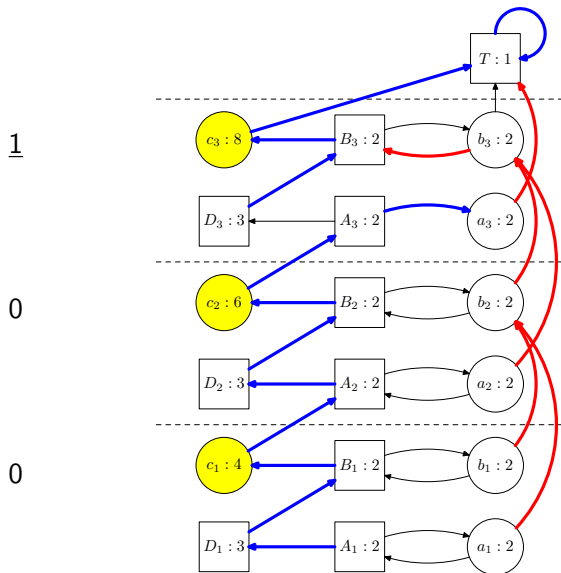
The construction



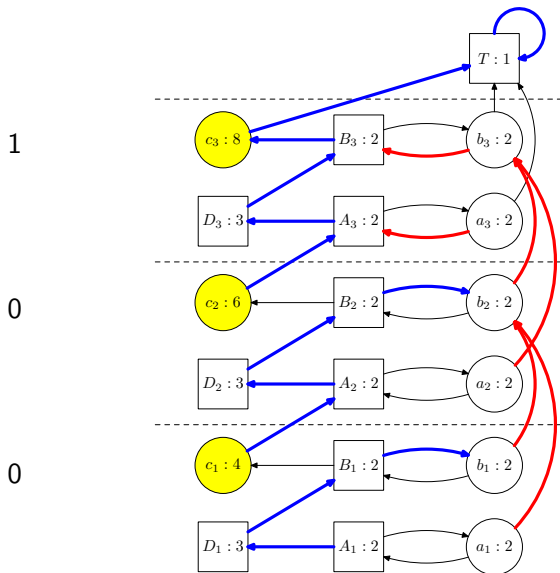
The construction



The construction



The construction



“Randomized bitcounter”

Start with n bits with value 0:

00000

“Randomized bitcounter”

Start with n bits with value 0:	00000
Pick a random bit i and fix it:	00 <u>0</u> 00

“Randomized bitcounter”

Start with n bits with value 0:	00000
Pick a random bit i and fix it:	00 <u>0</u> 00
Count recursively with the remaining $n - 1$ bits:	11 <u>0</u> 11



“Randomized bitcounter”

Start with n bits with value 0:	00000
Pick a random bit i and fix it:	00 <u>0</u> 00
Count recursively with the remaining $n - 1$ bits:	11 <u>0</u> 11
Increment the i 'th bit:	11 <u>1</u> 11



“Randomized bitcounter”

Start with n bits with value 0:	00000
Pick a random bit i and fix it:	00 <u>0</u> 00
Count recursively with the remaining $n - 1$ bits:	11 <u>0</u> 11
Increment the i 'th bit:	11 <u>1</u> 11
Reset the $i - 1$ lower bits:	11 <u>1</u> 00

“Randomized bitcounter”

Start with n bits with value 0:	00000
Pick a random bit i and fix it:	00 <u>0</u> 00
Count recursively with the remaining $n - 1$ bits:	11 <u>0</u> 11
Increment the i 'th bit:	11 <u>1</u> 11
Reset the $i - 1$ lower bits:	11 <u>1</u> 00
Count recursively with the $i - 1$ lower bits:	11100



“Randomized bitcounter”

Start with n bits with value 0:	00000
Pick a random bit i and fix it:	00 <u>0</u> 00
Count recursively with the remaining $n - 1$ bits:	11 <u>0</u> 11
Increment the i 'th bit:	11 <u>1</u> 11
Reset the $i - 1$ lower bits:	11 <u>1</u> 00
Count recursively with the $i - 1$ lower bits:	11100

- Expected number of steps:

$$f(0) = 1$$

$$f(n) = f(n-1) + \frac{1}{n} \sum_{i=0}^{n-1} f(i) \quad \text{for } n > 0$$

“Randomized bitcounter”

Start with n bits with value 0:	00000
Pick a random bit i and fix it:	00 <u>0</u> 00
Count recursively with the remaining $n - 1$ bits:	11 <u>0</u> 11
Increment the i 'th bit:	11 <u>1</u> 11
Reset the $i - 1$ lower bits:	11 <u>1</u> 00
Count recursively with the $i - 1$ lower bits:	11100

- Expected number of steps:

$$f(0) = 1$$

$$f(n) = f(n-1) + \frac{1}{n} \sum_{i=0}^{n-1} f(i) \quad \text{for } n > 0$$

- Same recurrence as for upper bound: $f(n) = 2^{\Theta(\sqrt{n})}$

The modified RANDOMFACET algorithm

Function RANDOMFACET*(G, σ, φ)

if $E_0 = \sigma$ **then**

 | **return** σ

else

 | $e \leftarrow \operatorname{argmin}_{e' \in E_0 \setminus \sigma} \varphi(e')$

 | $\sigma' \leftarrow \text{RANDOMFACET}^*(G \setminus \{e\}, \sigma, \varphi)$

 | **if** e is improving switch w.r.t. σ' **then**

 | $\sigma'' \leftarrow \sigma'[e]$

 | **return** RANDOMFACET*(G, σ'', φ)

 | **else**

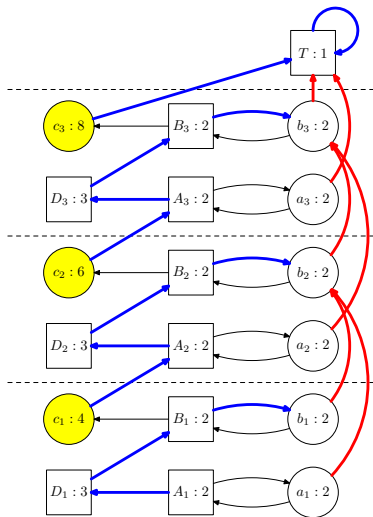
 | **return** σ'

Same expected number of steps when the permutation φ is picked uniformly at random.

Ensuring worst-case behavior

Good permutation:

① $\forall i : \varphi(b_i, B_i) < \varphi(a_i, A_i)$



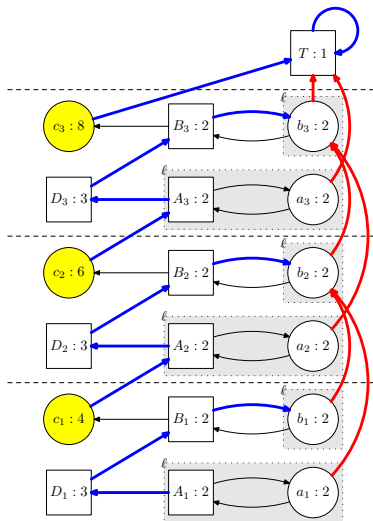
Ensuring worst-case behavior

Good permutation:

$$\textcircled{1} \quad \forall i : \varphi(b_i, B_i) < \varphi(a_i, A_i)$$

Increase probability of picking a good permutation at random by duplication:

$$\Pr[(1) \text{ not satisfied}] \leq n \frac{(\ell!)^2}{(2\ell)!} \leq \frac{n}{2^\ell}$$



Ensuring worst-case behavior

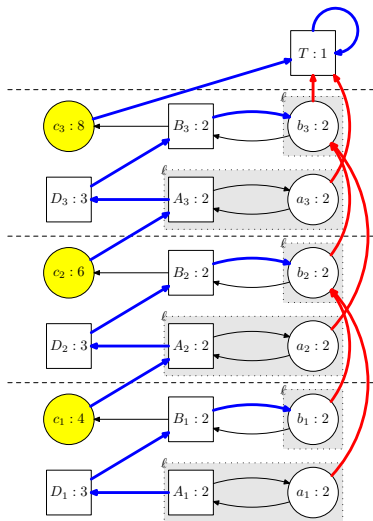
Good permutation:

$$\textcircled{1} \quad \forall i : \varphi(b_i, B_i) < \varphi(a_i, A_i)$$

Increase probability of picking a good permutation at random by duplication:

$$\Pr[(1) \text{ not satisfied}] \leq n \frac{(\ell!)^2}{(2\ell)!} \leq \frac{n}{2^\ell}$$

Here we lose a logarithmic factor in the exponent.



- We constructed parity games where the expected running time of the RANDOMFACET algorithm is $2^{\tilde{\Omega}(\sqrt{n})}$, almost matching the upper bound by Matoušek, Sharir and Welzl (1992).

Concluding remarks

- We constructed parity games where the expected running time of the RANDOMFACET algorithm is $2^{\tilde{\Omega}(\sqrt{n})}$, almost matching the upper bound by Matoušek, Sharir and Welzl (1992).
- By replacing the **minimizer** with vertices controlled by **nature**, we have later managed to prove the same lower bound for MDPs and linear programming.

- We constructed parity games where the expected running time of the RANDOMFACET algorithm is $2^{\tilde{\Omega}(\sqrt{n})}$, almost matching the upper bound by Matoušek, Sharir and Welzl (1992).
- By replacing the **minimizer** with vertices controlled by **nature**, we have later managed to prove the same lower bound for MDPs and linear programming.
- Using similar techniques we have also managed to show that RANDOMEDGE (repeatedly switch a random improving edge), when applied to the same settings, may require $2^{\Omega(n^{1/4})}$ expected steps.

Concluding remarks

- We constructed parity games where the expected running time of the RANDOMFACET algorithm is $2^{\tilde{\Omega}(\sqrt{n})}$, almost matching the upper bound by Matoušek, Sharir and Welzl (1992).
- By replacing the **minimizer** with vertices controlled by **nature**, we have later managed to prove the same lower bound for MDPs and linear programming.
- Using similar techniques we have also managed to show that RANDOMEDGE (repeatedly switch a random improving edge), when applied to the same settings, may require $2^{\Omega(n^{1/4})}$ expected steps.
- Major open problem: Polynomial time algorithm for Parity Games.



Thank you for listening!