

Clique-Width and Parity Games

Jan Obdržálek

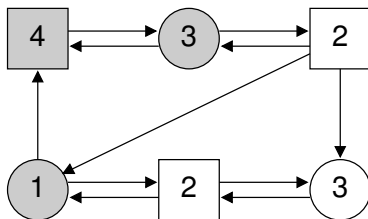
Masaryk University, Brno, Czech Republic

CSL, September 11, 2007

Parity games – the definition

$$\mathcal{G} = (V_0, V_1, E, \lambda)$$

- players: P_0 (Even) and P_1 (Odd)
- game graph $G = (V, E)$, where $V = V_0 \cup V_1$
- parity function $\lambda: V \rightarrow \mathbb{N}$ ($0 \notin \mathbb{N}$)
- Notation: $\circ \in V_0, \square \in V_1$



- Play: $\pi: \pi_1 \pi_2 \dots$
- The *highest* priority appearing infinitely often is **winning**

Parity games – the facts

- Equivalent to the modal μ -calculus model checking
- in $NP \cap co-NP$ (even $UP \cap co-UP$)
- sub-exponential *deterministic* algorithm [JPZ06]
- no polynomial algorithm known
- the naughty *strategy improvement* algorithm
- relationship to *mean payoff games* and *stochastic games*

Parity games – the facts

- Equivalent to the modal μ -calculus model checking
- in $\text{NP} \cap \text{co-NP}$ (even $\text{UP} \cap \text{co-UP}$)
- sub-exponential *deterministic* algorithm [JPZ06]
- no polynomial algorithm known
- the naughty *strategy improvement* algorithm
- relationship to *mean payoff games* and *stochastic games*

Parity games – the facts

- Equivalent to the modal μ -calculus model checking
- in $NP \cap co-NP$ (even $UP \cap co-UP$)
- sub-exponential *deterministic* algorithm [JPZ06]
- no polynomial algorithm known
- the naughty *strategy improvement* algorithm
- relationship to *mean payoff games* and *stochastic games*

Parity games – the facts

- Equivalent to the modal μ -calculus model checking
- in $NP \cap co-NP$ (even $UP \cap co-UP$)
- sub-exponential *deterministic* algorithm [JPZ06]
- no polynomial algorithm known
- the naughty *strategy improvement* algorithm
- relationship to *mean payoff games* and *stochastic games*

Parity games – the facts

- Equivalent to the modal μ -calculus model checking
- in $NP \cap co-NP$ (even $UP \cap co-UP$)
- sub-exponential *deterministic* algorithm [JPZ06]
- no polynomial algorithm known
- the naughty *strategy improvement* algorithm
- relationship to *mean payoff games* and *stochastic games*

Parity games – the facts

- Equivalent to the modal μ -calculus model checking
- in $NP \cap co-NP$ (even $UP \cap co-UP$)
- sub-exponential *deterministic* algorithm [JPZ06]
- no polynomial algorithm known
- the naughty *strategy improvement* algorithm
- relationship to *mean payoff games* and *stochastic games*

What if we restrict ourselves to special classes of graphs?

- *Bounded tree-width* in P [O03]
- *Bounded entanglement* in P [BG04]
- *Bounded DAG-width* in P [BDHK06]

Tree-width

- connectivity measure for undirected graphs
- measures how close is a graph to a tree
- many problems which are NP(even PSPACE) hard can be solved in linear/polynomial time on graphs of bounded tree-width

What if we restrict ourselves to special classes of graphs?

- *Bounded tree-width* in P [O03]
- *Bounded entanglement* in P [BG04]
- *Bounded DAG-width* in P [BDHK06]

Tree-width

- connectivity measure for undirected graphs
- measures how close is a graph to a tree
- many problems which are NP(even PSPACE) hard can be solved in linear/polynomial time on graphs of bounded tree-width

What if we restrict ourselves to special classes of graphs?

- *Bounded tree-width* in P [O03]
- *Bounded entanglement* in P [BG04]
- *Bounded DAG-width* in P [BDHK06]

Tree-width

- connectivity measure for undirected graphs
- measures how close is a graph to a tree
- many problems which are NP(even PSPACE) hard can be solved in linear/polynomial time on graphs of bounded tree-width

What if we restrict ourselves to special classes of graphs?

- *Bounded tree-width* in P [O03]
- *Bounded entanglement* in P [BG04]
- *Bounded DAG-width* in P [BDHK06]

Tree-width

- connectivity measure for undirected graphs
- measures how close is a graph to a tree
- many problems which are NP(even PSPACE) hard can be solved in linear/polynomial time on graphs of bounded tree-width

What if we restrict ourselves to special classes of graphs?

- *Bounded tree-width* in P [O03]
- *Bounded entanglement* in P [BG04]
- *Bounded DAG-width* in P [BDHK06]

Tree-width

- connectivity measure for undirected graphs
- measures how close is a graph to a tree
- many problems which are NP(even PSPACE) hard can be solved in linear/polynomial time on graphs of bounded tree-width

What about “*dense*” graphs?

- Can we do cliques in P?
- Can we do “complete” bipartite graphs in P?
- Can we ...?

What about “*dense*” graphs?

- Can we do cliques in P?
- Can we do “complete” bipartite graphs in P?
- Can we ...?

What about “*dense*” graphs?

- Can we do cliques in P?
- Can we do “complete” bipartite graphs in P?
- Can we ...?

What about “*dense*” graphs?

- Can we do cliques in P?
- Can we do “complete” bipartite graphs in P?
- Can we ...?

- Introduced by Courcelle and Olariu
- Measures how close is a graph to a complete bipartite graph
- Complete bipartite graphs and cliques have clique-width 2
- Naturally defined for both directed and undirected graphs
- Bounded tree-width implies bounded clique-width
- Many NP-hard problems can be solved in linear/polynomial time on graphs of bounded clique-width

c.w. is a both extension and a complementary measure to t.w.

- Introduced by Courcelle and Olariu
- Measures how close is a graph to a complete bipartite graph
- Complete bipartite graphs and cliques have clique-width 2
- Naturally defined for both directed and undirected graphs
- Bounded tree-width implies bounded clique-width
- Many NP-hard problems can be solved in linear/polynomial time on graphs of bounded clique-width

c.w. is a both extension and a complementary measure to t.w.

Clique-width – the definition

Let us have k colours and the following four operations:

- $[i]$ – creates a vertex of colour i
- $\alpha_{i,j}$ – adds edges between all pairs of vertices with colours i and j
- $\rho_{i \rightarrow j}$ – changes the colour i to j
- \oplus – disjoint union

Graph G is of *clique-width* k if it can be created by the operations above using k colours (and no less).

Clique-width – an example

2 ●

1 ●

$$A = [1] \oplus [2]$$

2 ●

2 ●

$$B = \rho_{1 \rightarrow 2}(\alpha_{1,2}(A))$$

2 ●

2 ●

● 1

$$C = B \oplus [1]$$

2 ●

2 ●

$$D = \rho_{1 \rightarrow 2}(\alpha_{1,2}(C))$$

2 ●

2 ●

$$E = D \oplus [1]$$

● 1

2 ●

2 ●

$$F = \alpha_{1,2}(E)$$

● 1

2 ●

Isn't it easy?

Solving parity games on graphs of bounded clique-width:

- We would use dynamic programming.
- Use the same idea as for tree width – compute all possible results using the *best replies* of P_1 to *all choices of strategies* of P_0 .
- Boils down to providing algorithms for the four different node types.

Definition (Reward ordering)

For two priorities $p, q \in \mathbb{N}$ we write $p \sqsubset q$ if p is odd and q is even, or $p > q$ and p, q are odd, or $p < q$ and p, q are even. We write $p \sqsubseteq q$ if $p \sqsubset q$ or $p = q$.

Isn't it easy?

Solving parity games on graphs of bounded clique-width:

- We would use dynamic programming.
- Use the same idea as for tree width – compute all possible results using the *best replies* of P_1 to *all choices of strategies* of P_0 .
- Boils down to providing algorithms for the four different node types.

Definition (Reward ordering)

For two priorities $p, q \in \mathbb{N}$ we write $p \sqsubset q$ if p is odd and q is even, or $p > q$ and p, q are odd, or $p < q$ and p, q are even. We write $p \sqsubseteq q$ if $p \sqsubset q$ or $p = q$.

Isn't it easy?

Solving parity games on graphs of bounded clique-width:

- We would use dynamic programming.
- Use the same idea as for tree width – compute all possible results using the *best replies* of P_1 to *all choices of strategies* of P_0 .
- Boils down to providing algorithms for the four different node types.

Definition (Reward ordering)

For two priorities $p, q \in \mathbb{N}$ we write $p \sqsubset q$ if p is odd and q is even, or $p > q$ and p, q are odd, or $p < q$ and p, q are even. We write $p \sqsubseteq q$ if $p \sqsubset q$ or $p = q$.

The result space

Fix a strategy σ of P_0 .

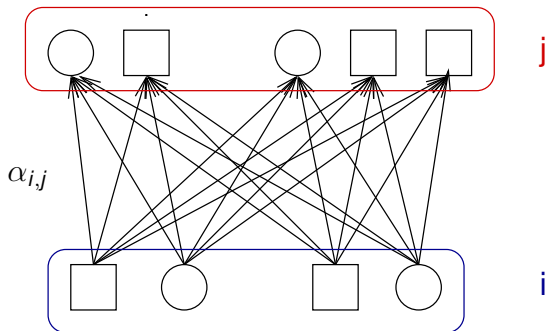
Let a , b and c be the colours.

For a node of n of the decomposition and σ the table would look something like this:

	a	b	c
a	4	6	5
b	-	5	4
c	\perp	\perp	\perp

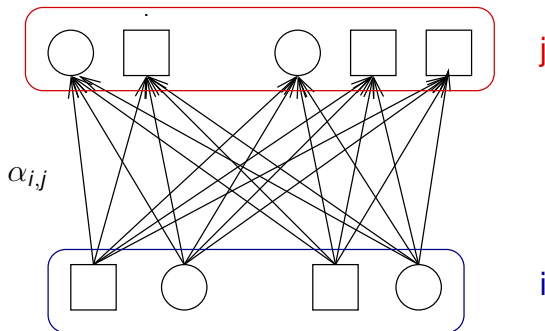
But each colour represents several vertices!

Main problems



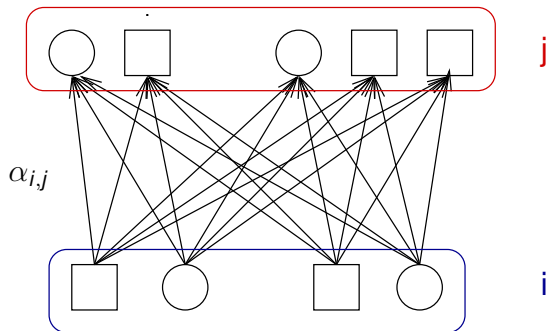
- 1 We have to choose a strategy for each of the vertices of colour i .
- 2 Vertices of both players have the same colour.
- 3 Graph is not bipartite.

Main problems



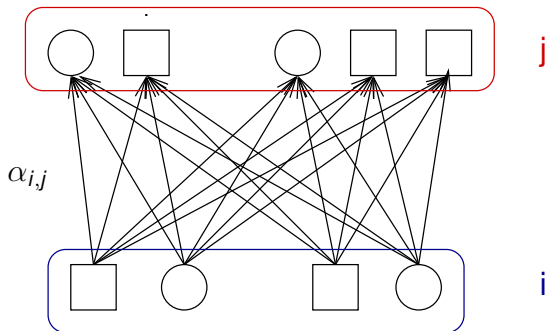
- 1 We have to choose a strategy for each of the vertices of colour i .
- 2 Vertices of both players have the same colour.
- 3 Graph is not bipartite.

Main problems



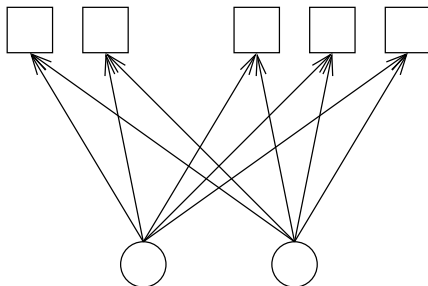
- 1 We have to choose a strategy for each of the vertices of colour i .
- 2 Vertices of both players have the same colour.
- 3 Graph is not bipartite.

Main problems

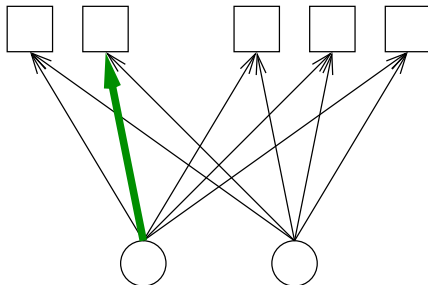


- 1 We have to choose a strategy for each of the vertices of colour i .
- 2 Vertices of both players have the same colour.
- 3 Graph is not bipartite.

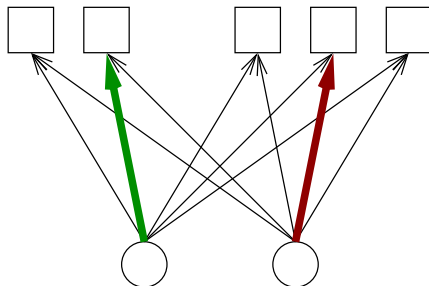
Can we choose a common successor?



Can we choose a common successor?

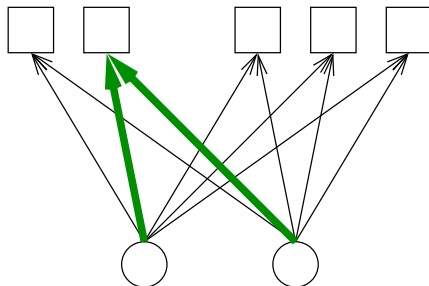


Can we choose a common successor?



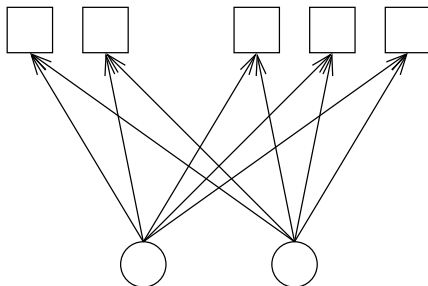
That's easy!

Can we choose a common successor?



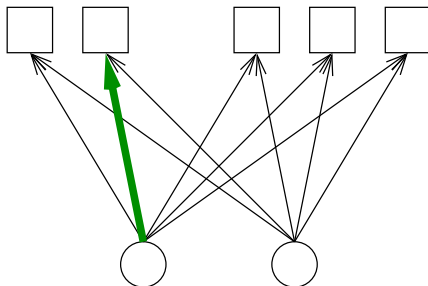
That's easy!

Can we choose a common successor?

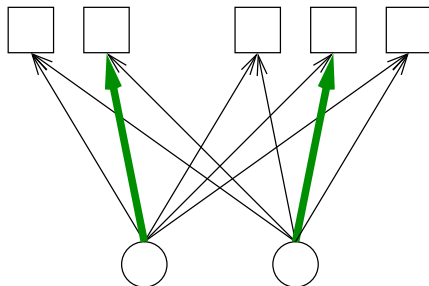


This is harder

Can we choose a common successor?

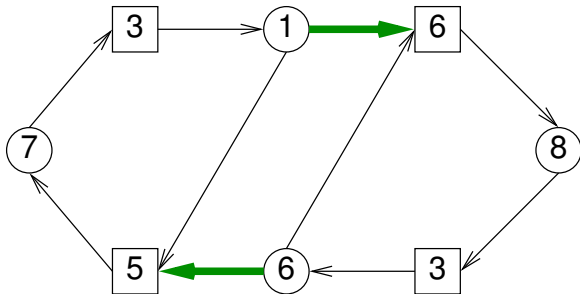


Can we choose a common successor?

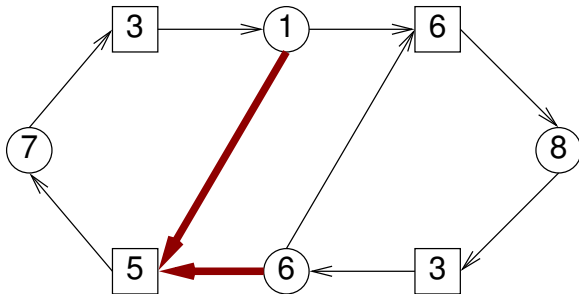


And now?

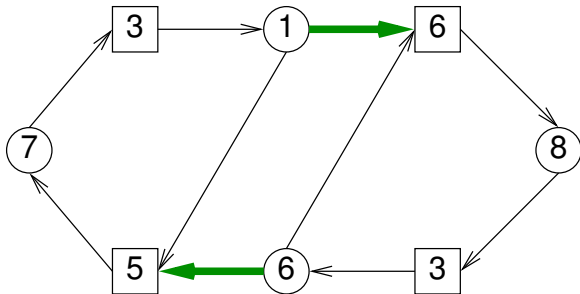
The important case



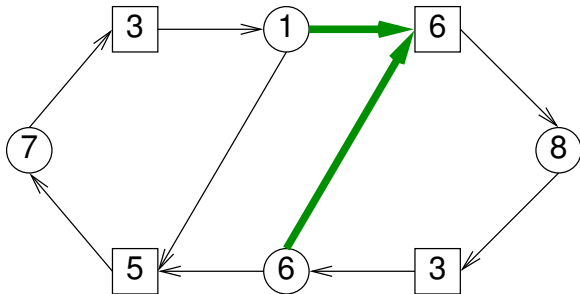
The important case



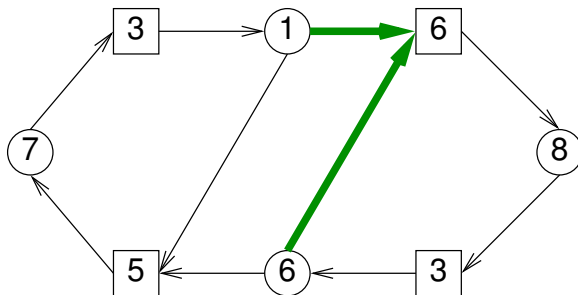
The important case



The important case



The important case



- There is always a way of choosing a common successor.
- For more source vertices we use induction.

Theorem (joint choice property)

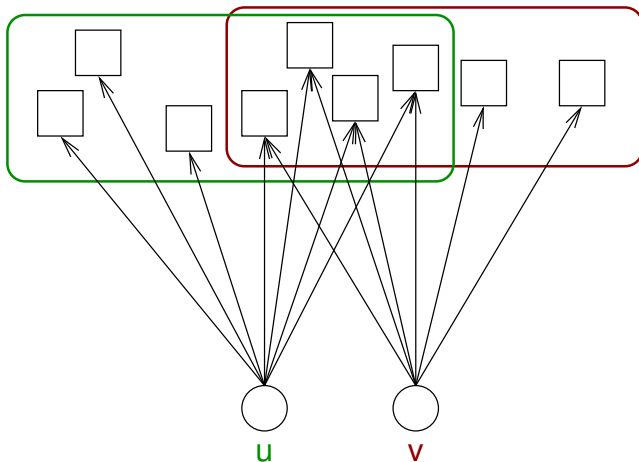
Let $\mathcal{G} = (V, E, \lambda)$ be a parity game, $U \subseteq V_0$ a set of vertices of P_0 and $\sigma \in \Sigma_0$ a strategy of P_0 . Also denote $\sigma(U) = \bigcup_{u \in U} \sigma(u)$ and require that $\forall u \in U \forall v \in \sigma(U). (u, v) \in E$. If σ is a winning strategy for some vertex $w \in U$ and P_0 , then there exists a vertex \bar{u} in $\sigma(U)$ such that the strategy σ' defined as

$$\sigma'(v) = \begin{cases} \bar{u} & \text{if } v \in U \\ \sigma(v) & \text{otherwise} \end{cases}$$

is a winning strategy for P_0 from all vertices of U . Moreover for all $v \in V(G)$ if σ is winning for v and P_0 , so is σ' .

In general

This is how the situation can look in general:



Application to clique-width

- At each point a vertex of P_0 is either connected to a successor, or not yet.
- Once two vertices have the same colour, they will also have the same successors *from that point onwards*.
- (The other successors were already processed.)
- All vertices of the same colour are assigned a successor at once.

That allows us to treat a colour as a single vertex (almost).

Application to clique-width

- At each point a vertex of P_0 is either connected to a successor, or not yet.
- Once two vertices have the same colour, they will also have the same successors *from that point onwards*.
- (The other successors were already processed.)
- All vertices of the same colour are assigned a successor at once.

That allows us to treat a colour as a single vertex (almost).

Application to clique-width

- At each point a vertex of P_0 is either connected to a successor, or not yet.
- Once two vertices have the same colour, they will also have the same successors *from that point onwards*.
- (The other successors were already processed.)
- All vertices of the same colour are assigned a successor at once.

That allows us to treat a colour as a single vertex (almost).

Application to clique-width

- At each point a vertex of P_0 is either connected to a successor, or not yet.
- Once two vertices have the same colour, they will also have the same successors *from that point onwards*.
- (The other successors were already processed.)
- All vertices of the same colour are assigned a successor at once.

That allows us to treat a colour as a single vertex (almost).

Theorem

Parity games can be solved in *polynomial time* on graphs of bounded clique-width.

- Similar argument works for *rank-width*.
- Can *Joint Choice Property* be useful in other contexts?
- *Are parity games in P?*

Theorem

Parity games can be solved in *polynomial time* on graphs of bounded clique-width.

- Similar argument works for *rank-width*.
- Can *Joint Choice Property* be useful in other contexts?
- *Are parity games in P?*

Theorem

Parity games can be solved in *polynomial time* on graphs of bounded clique-width.

- Similar argument works for *rank-width*.
- Can *Joint Choice Property* be useful in other contexts?
- *Are parity games in P?*

Theorem

Parity games can be solved in *polynomial time* on graphs of bounded clique-width.

- Similar argument works for *rank-width*.
- Can *Joint Choice Property* be useful in other contexts?
- *Are parity games in P?*