DAG-width and Parity Games

Paul Hunter
Humboldt University, Berlin

Joint work with
Dietmar Berwanger, Anuj Dawar and Stephan Kreutzer

STACS, February 2006

Motivation

Tree-width introduced by Robertson and Seymour

- Tree decompositions provide for recursive algorithms
- Bounding tree-width gives polynomial time execution

©Madlantern Art

Problem

Tree-width ignores direction

Motivation

Tree-width introduced by Robertson and Seymour

- Tree decompositions provide for recursive algorithms
- Bounding tree-width gives polynomial time execution

©Madlantern Art

Problem

Tree-width ignores direction

Motivation

Tree-width introduced by Robertson and Seymour

- Tree decompositions provide for recursive algorithms
- Bounding tree-width gives polynomial time execution

©Madlantern Art

Problem

Tree-width ignores direction

Motivation

Directed tree-width by Johnson, Robertson, Seymour and Thomas

- Not an obvious extension of tree-width
- Complicated definition does not lend itself to algorithms

Aim

Find a natural extension of tree-width to directed graphs that is algorithmically useful.

Motivation

Directed tree-width by Johnson, Robertson, Seymour and Thomas

- Not an obvious extension of tree-width
- Complicated definition does not lend itself to algorithms

Aim

Find a natural extension of tree-width to directed graphs that is algorithmically useful.

Motivation

Directed tree-width by Johnson, Robertson, Seymour and Thomas

- Not an obvious extension of tree-width
- Complicated definition does not lend itself to algorithms

Aim

Find a natural extension of tree-width to directed graphs that is algorithmically useful.

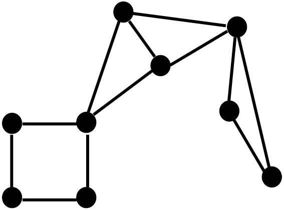
Overview

- Review tree-width
- Cops and robber game
- DAG-decompositions and DAG-width
- An algorithm for parity games
- Further work

Tree-width

The tree-width of a graph measures its similarity to a tree.

A graph has tree-width $\leq k$ if it can be covered by sub-graphs of size $\leq (k+1)$ in a tree-like fashion.

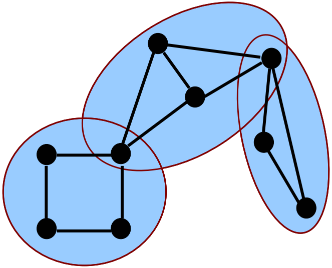


Tree-width can be characterised by a cops and robber game.

Tree-width

The tree-width of a graph measures its similarity to a tree.

A graph has tree-width $\leq k$ if it can be covered by sub-graphs of size $\leq (k+1)$ in a tree-like fashion.

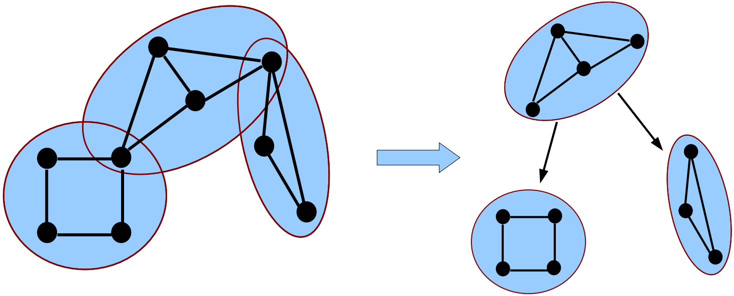


Tree-width can be characterised by a cops and robber game.

Tree-width

The tree-width of a graph measures its similarity to a tree.

A graph has tree-width $\leq k$ if it can be covered by sub-graphs of size $\leq (k+1)$ in a tree-like fashion.

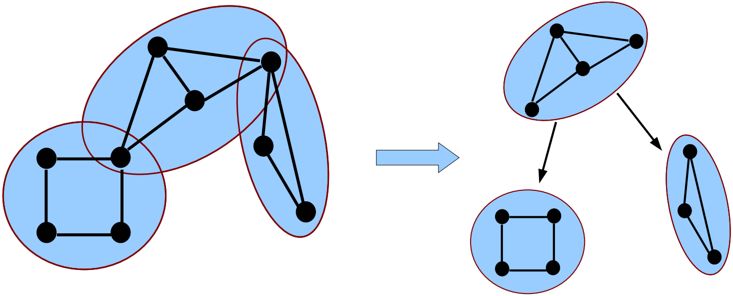


Tree-width can be characterised by a cops and robber game.

Tree-width

The tree-width of a graph measures its similarity to a tree.

A graph has tree-width $\leq k$ if it can be covered by sub-graphs of size $\leq (k+1)$ in a tree-like fashion.



Tree-width can be characterised by a cops and robber game.

Cops and robber game

Cops and robber game

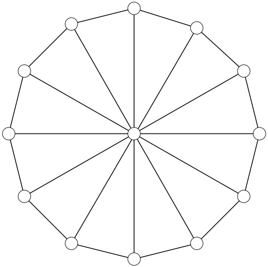
k Cops

Cops and robber game

k Cops

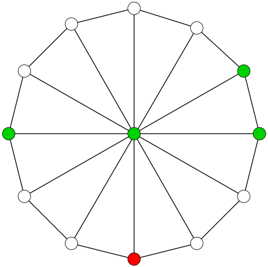
Robber

Cops and robber game



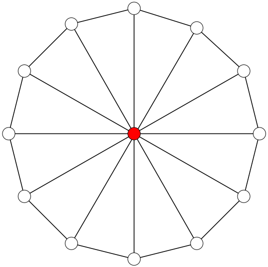
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \rightarrow (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



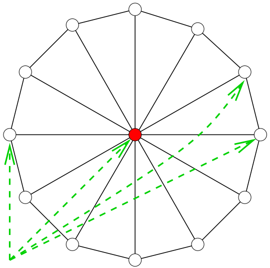
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \to (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



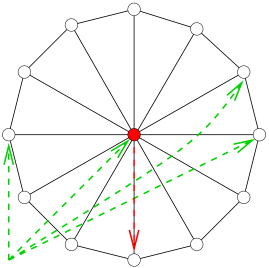
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \to (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



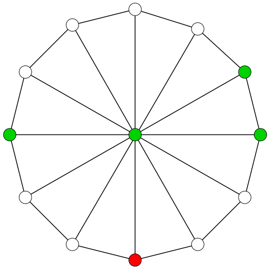
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \to (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



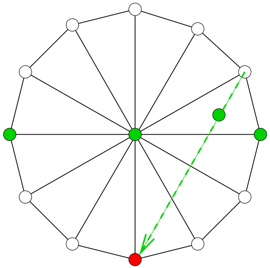
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \rightarrow (r', C')$
  Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



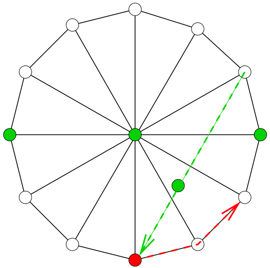
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \rightarrow (r', C')$
  Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



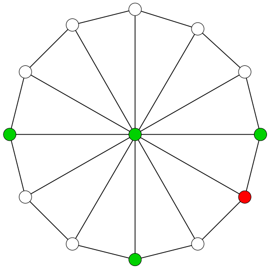
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \to (r', C')$
  Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



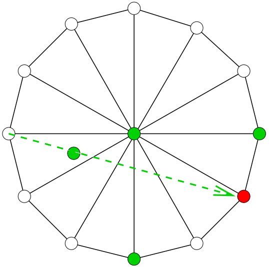
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \rightarrow (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



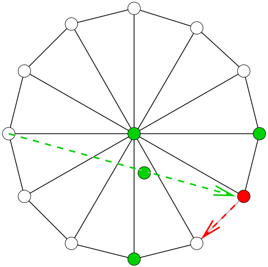
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \to (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



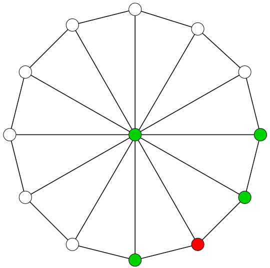
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \to (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



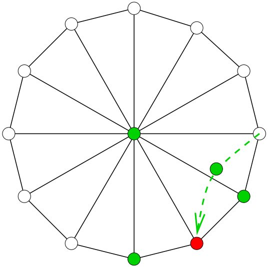
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \to (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



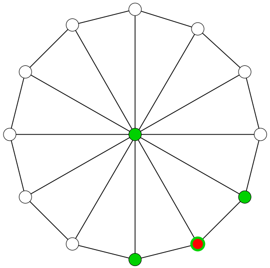
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \to (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



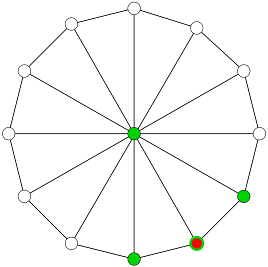
- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \to (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \rightarrow (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops and robber game



- **Players:** Cop and robber
- **Positions:** (r, C), where $r \in V$ and $C \subseteq V$ with $|C| \leq k$
- **Initial position:** $(r_0, \emptyset)$, where $r_0 \in V$ is chosen by the robber
- **Round of a play:** $(r, C) \rightarrow (r', C')$ Cops choose C', then robber chooses r' such that there is a path from r to r' in $G \setminus (C \cap C')$.
- **Winning Conditions:** Cops win if position (r, C) with $r \in C$ is reached; otherwise the robber wins.

Cops, robbers and tree-width

Theorem (Seymour and Thomas 1993)

$\mathcal{G}$ has tree-width $\leq k$ if, and only if $k+1$ cops have a winning strategy

Question

What about directed graphs?

Cops, robbers and tree-width

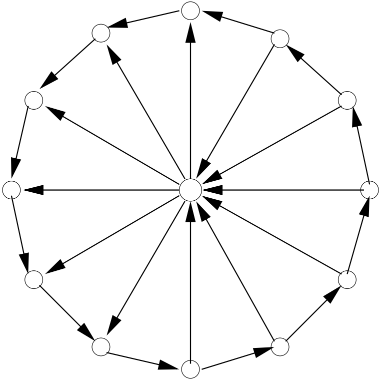
Theorem (Seymour and Thomas 1993)

$\mathcal{G}$ has tree-width $\leq k$ if, and only if $k+1$ cops have a winning strategy

Question

What about directed graphs?

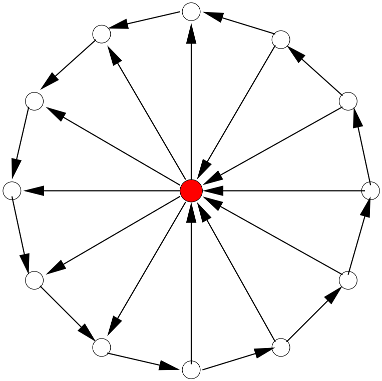
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

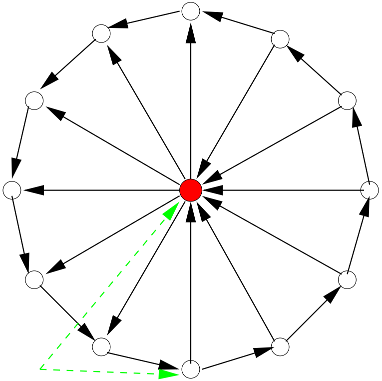
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

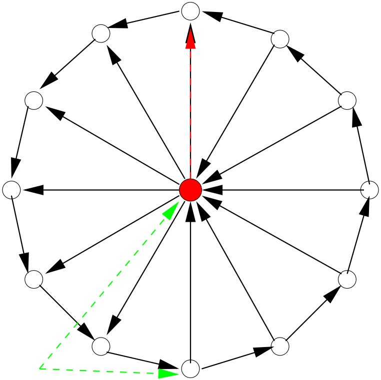
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

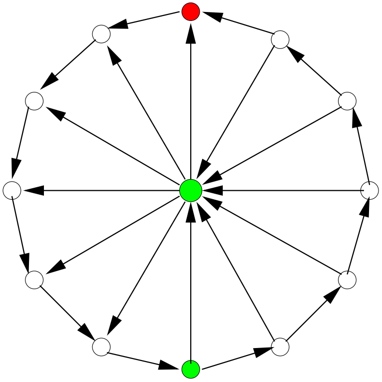
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

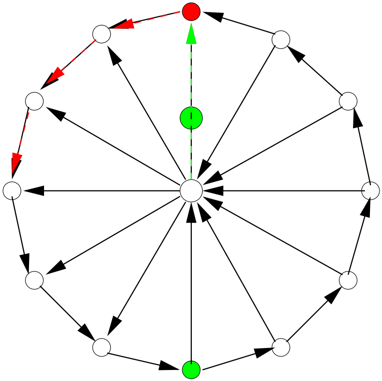
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

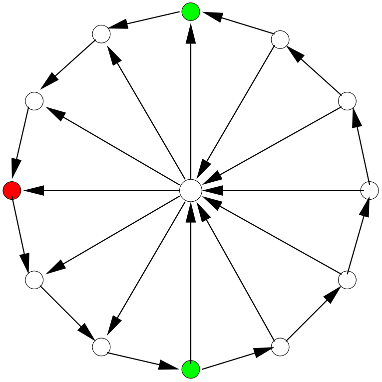
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

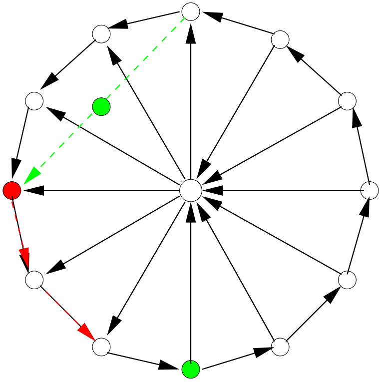
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

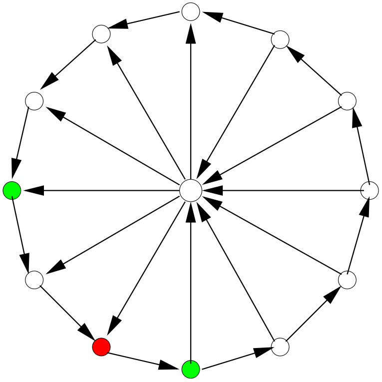
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

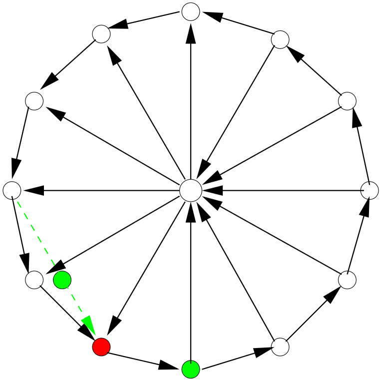
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

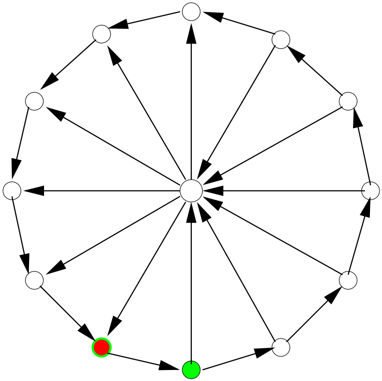
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

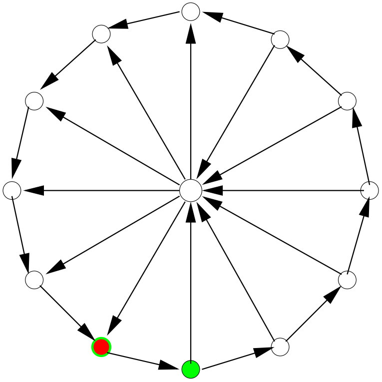
Directed cops and robbers



Problem

Find a decomposition that corresponds to this game

Directed cops and robbers



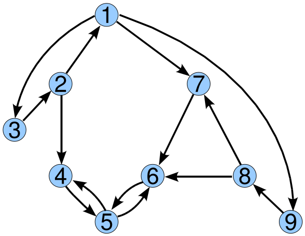
Problem

Find a decomposition that corresponds to this game

DAG-width

The DAG-width of a directed graph measures its similarity to a DAG.

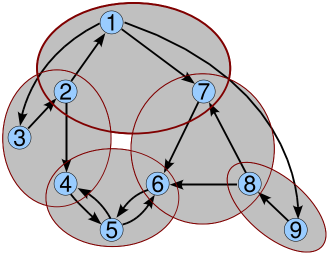
A graph has DAG-width $\leq k$ if it can be covered by **subsets** of size $\leq k$ in a DAG-like fashion such that an edge only leaves a sub-DAG through its (root's) connection with the rest of the DAG



DAG-width

The DAG-width of a directed graph measures its similarity to a DAG.

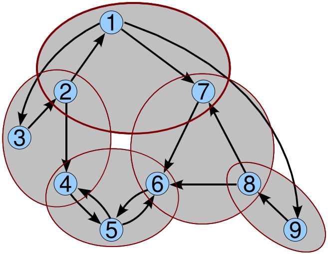
A graph has DAG-width $\leq k$ if it can be covered by **subsets** of size $\leq k$ in a DAG-like fashion such that an edge only leaves a sub-DAG through its (root's) connection with the rest of the DAG

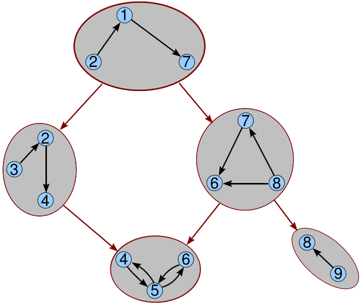


DAG-width

The DAG-width of a directed graph measures its similarity to a DAG.

A graph has DAG-width $\leq k$ if it can be covered by **subsets** of size $\leq k$ in a DAG-like fashion such that an edge only leaves a sub-DAG through its (root's) connection with the rest of the DAG

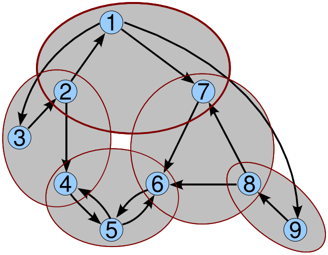


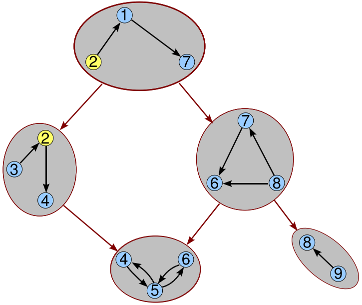


DAG-width

The DAG-width of a directed graph measures its similarity to a DAG.

A graph has DAG-width $\leq k$ if it can be covered by **subsets** of size $\leq k$ in a DAG-like fashion such that an edge only leaves a sub-DAG through its (root's) connection with the rest of the DAG





DAG-decompositions and DAG-width

A DAG-decomposition of a directed graph $\mathcal{G}$ is a tuple $(\mathcal{D}, (X_d)_{d \in V(\mathcal{D})})$ such that:

- $\mathcal{D}$ is a DAG
- X_d cover $V(\mathcal{G})$
- For every d' on the path from d to d'' ($d \preceq_{\mathcal{D}} d' \preceq_{\mathcal{D}} d''$), $X_d \cap X_{d''} \subseteq X_{d'}$
- For every $(c, d) \in E(\mathcal{D})$, $X_c \cap X_d$ guards $\left(\bigcup_{d \preceq_{\mathcal{D}} d'} X_{d'}\right) \setminus X_c$. If d is a root of $\mathcal{D}$, we replace X_c with $\emptyset$.

The width of a DAG-decomposition is $\max_{d \in V(\mathcal{D})} |X_d|$.
The DAG-width of a directed graph is the minimal width of all its DAG-decompositions.

Results

Theorem

$\mathcal{G}$ has DAG-width k if and only if k cops have a monotone winning strategy on $\mathcal{G}$

A monotone strategy is one where every vertex is visited by a cop at most once.

Theorem (Complexity Issues)

- *For fixed k, deciding if $\mathcal{G}$ has DAG-width $\leq k$ is in* $\text{P}\small{\text{TIME}}$
- *Given $\mathcal{G}$ and k, deciding if $\mathcal{G}$ has DAG-width $\leq k$ is* NP*-hard*

Results

Theorem

$\mathcal{G}$ has DAG-width k if and only if k cops have a monotone winning strategy on $\mathcal{G}$

A monotone strategy is one where every vertex is visited by a cop at most once.

Theorem (Complexity Issues)

- *For fixed k, deciding if $\mathcal{G}$ has DAG-width $\leq k$ is in* $\textsc{Ptime}$
- *Given $\mathcal{G}$ and k, deciding if $\mathcal{G}$ has DAG-width $\leq k$ is* NP*-hard*

More results...

- $\mathrm{dtw}(\mathcal{G}) \leq \text{DAG-width}(\mathcal{G}) \leq \mathrm{tw}(\mathcal{G})$
- $\text{DAG-width}(\mathcal{G}) = 1$ iff $\mathcal{G}$ is acyclic
- DAG-width is **not** preserved under edge reversal

Theorem

Parity games on graphs of bounded DAG-width can be decided in polynomial time

More results...

- $\mathrm{dtw}(\mathcal{G}) \leq \text{DAG-width}(\mathcal{G}) \leq \mathrm{tw}(\mathcal{G})$
- $\text{DAG-width}(\mathcal{G}) = 1$ iff $\mathcal{G}$ is acyclic
- DAG-width is **not** preserved under edge reversal

Theorem

Parity games on graphs of bounded DAG-width can be decided in polynomial time

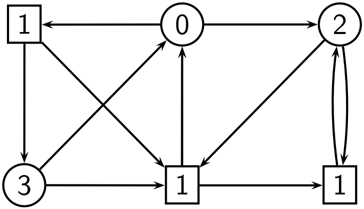
Parity games

Players: *Player 0* and *Player 1*
Arena: (V, E, V_0, V_1, Ω), where

- (V, E) is a directed graph
- V_0 and V_1 partition V
- $\Omega : V \to \mathbb{N}$ priority function

Players move a token around the graph for possibly infinitely many moves

Winner is determined by minimum priority seen infinitely often



Circles: nodes for Player 0
Boxes: nodes for Player 1

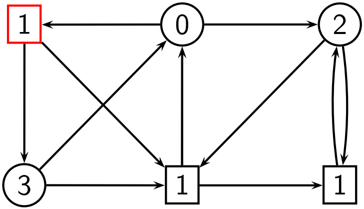
Parity games

Players: *Player 0* and *Player 1*
Arena: (V, E, V_0, V_1, Ω), where

- (V, E) is a directed graph
- V_0 and V_1 partition V
- $\Omega : V \to \mathbb{N}$ priority function

Players move a token around the graph for possibly infinitely many moves

Winner is determined by minimum priority seen infinitely often



Circles: nodes for Player 0
Boxes: nodes for Player 1

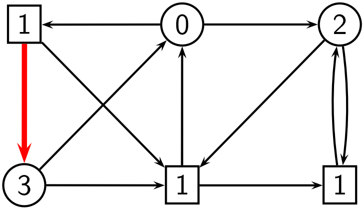
Parity games

Players: *Player 0* and *Player 1*
Arena: (V, E, V_0, V_1, Ω), where

- (V, E) is a directed graph
- V_0 and V_1 partition V
- $\Omega : V \to \mathbb{N}$ priority function

Players move a token around the graph for possibly infinitely many moves

Winner is determined by minimum priority seen infinitely often



Circles: nodes for Player 0
Boxes: nodes for Player 1

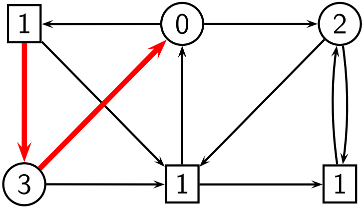
Parity games

Players: *Player 0* and *Player 1*
Arena: (V, E, V_0, V_1, Ω), where

- (V, E) is a directed graph
- V_0 and V_1 partition V
- $\Omega : V \to \mathbb{N}$ priority function

Players move a token around the graph for possibly infinitely many moves

Winner is determined by minimum priority seen infinitely often



Circles: nodes for Player 0
Boxes: nodes for Player 1

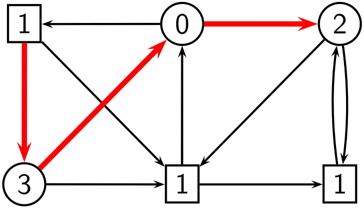
Parity games

Players: *Player 0* and *Player 1*
Arena: (V, E, V_0, V_1, Ω), where

- (V, E) is a directed graph
- V_0 and V_1 partition V
- $\Omega : V \to \mathbb{N}$ priority function

Players move a token around the graph for possibly infinitely many moves

Winner is determined by minimum priority seen infinitely often



Circles: nodes for Player 0
Boxes: nodes for Player 1

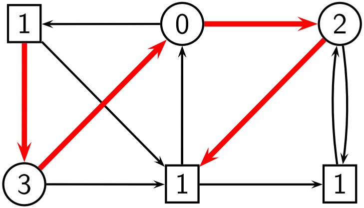
Parity games

Players: *Player 0* and *Player 1*
Arena: (V, E, V_0, V_1, Ω), where

- (V, E) is a directed graph
- V_0 and V_1 partition V
- $\Omega : V \to \mathbb{N}$ priority function

Players move a token around the graph for possibly infinitely many moves

Winner is determined by minimum priority seen infinitely often



Circles: nodes for Player 0
Boxes: nodes for Player 1

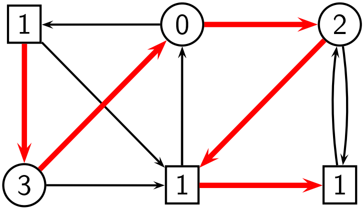
Parity games

Players: *Player 0* and *Player 1*
Arena: (V, E, V_0, V_1, Ω), where

- (V, E) is a directed graph
- V_0 and V_1 partition V
- $\Omega : V \to \mathbb{N}$ priority function

Players move a token around the graph for possibly infinitely many moves

Winner is determined by minimum priority seen infinitely often



Circles: nodes for Player 0
Boxes: nodes for Player 1

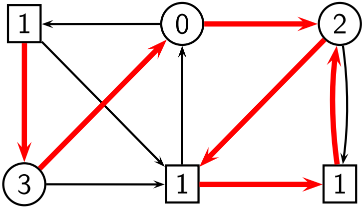
Parity games

Players: *Player 0* and *Player 1*
Arena: (V, E, V_0, V_1, Ω), where

- (V, E) is a directed graph
- V_0 and V_1 partition V
- $\Omega : V \to \mathbb{N}$ priority function

Players move a token around the graph for possibly infinitely many moves

Winner is determined by minimum priority seen infinitely often



Circles: nodes for Player 0
Boxes: nodes for Player 1

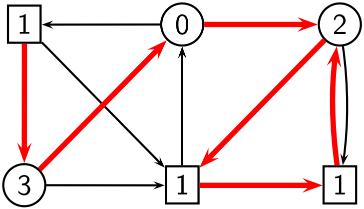
Parity games

Players: *Player 0* and *Player 1*
Arena: (V, E, V_0, V_1, Ω), where

- (V, E) is a directed graph
- V_0 and V_1 partition V
- $\Omega : V \to \mathbb{N}$ priority function

Players move a token around the graph for possibly infinitely many moves

Winner is determined by minimum priority seen infinitely often



Circles: nodes for Player 0
Boxes: nodes for Player 1

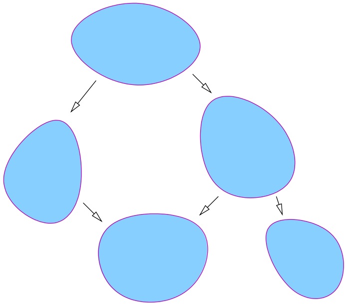
Parity game results

Polynomial-time equivalent to μ-calculus model checking

Decidable in $\mathrm{NP} \cap \textsc{co}\text{-}\mathrm{NP}$

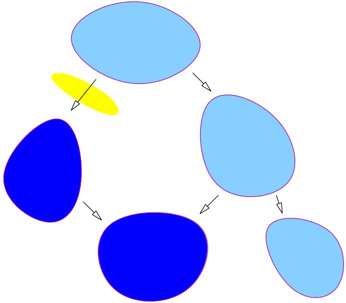
Decidability in $\textsc{Ptime}$ an open problem

Parity games algorithm



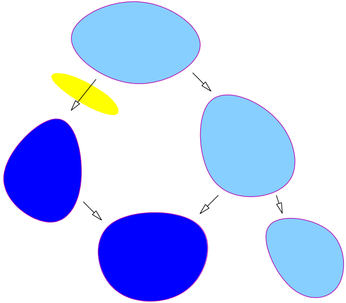
1. **Compute DAG-Decomposition**
2. **Use structure to succinctly represent all plays in subgraphs**
   - $\text{result}_f(U, v)$ is all possible outcomes when Player 0 plays f from v in U
3. **Compute $\text{result}_f(U, v)$ bottom-up**

Parity games algorithm



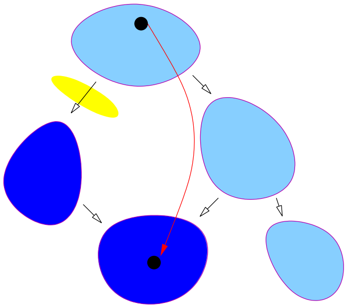
1. **Compute DAG-Decomposition**
2. **Use structure to succinctly represent all plays in subgraphs**
   - $\text{result}_f(U, v)$ is all possible outcomes when Player 0 plays f from v in U
3. **Compute $\text{result}_f(U, v)$ bottom-up**

Parity games algorithm



1. **Compute DAG-Decomposition**
2. **Use structure to succinctly represent all plays in subgraphs**
   - $\text{result}_f(U, v)$ is all possible outcomes when Player 0 plays f from v in U
3. **Compute $\text{result}_f(U, v)$ bottom-up**

Parity games algorithm



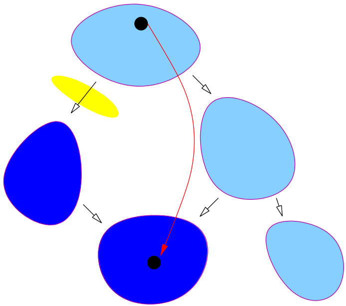
Major problem: Edges entering the sub-DAG

- Cannot "forget" vertices
- Exponential number of strategies generated

Solutions:

- Consider functions from sub-DAG to border
- Compute feasible outcomes

Parity games algorithm



Major problem: Edges entering the sub-DAG

- Cannot "forget" vertices
- Exponential number of strategies generated

Solutions:

- Consider functions from sub-DAG to border
- Compute feasible outcomes

Conclusions and further work

- Introduced a natural extension of tree-width to directed graphs.
- Provided a polynomial-time algorithm for parity games on graphs of bounded DAG-width – subsuming other results such as bounded tree-width.

- Are monotone strategies sufficient?
- Generalisation of havens, brambles, minors, separators?
- Generalisation of Courcelle's theorem?

Conclusions and further work

- Introduced a natural extension of tree-width to directed graphs.
- Provided a polynomial-time algorithm for parity games on graphs of bounded DAG-width – subsuming other results such as bounded tree-width.

- Are monotone strategies sufficient?
- Generalisation of havens, brambles, minors, separators?
- Generalisation of Courcelle's theorem?

Conclusions and further work

- Introduced a natural extension of tree-width to directed graphs.
- Provided a polynomial-time algorithm for parity games on graphs of bounded DAG-width – subsuming other results such as bounded tree-width.

- Are monotone strategies sufficient?
- Generalisation of havens, brambles, minors, separators?
- Generalisation of Courcelle's theorem?

Conclusions and further work

- Introduced a natural extension of tree-width to directed graphs.
- Provided a polynomial-time algorithm for parity games on graphs of bounded DAG-width – subsuming other results such as bounded tree-width.

- Are monotone strategies sufficient?
- Generalisation of havens, brambles, minors, separators?
- Generalisation of Courcelle's theorem?