

# DAG-width: Cops and Robbers on Directed Graphs

Paul Hunter  
Humboldt-University, Berlin

Joint work with  
Dietmar Berwanger, Anuj Dawar and Stephan Kreutzer

GAMES Workshop, 2005

# Motivation

- Tree-width introduced by Robertson and Seymour
  - ▶ Tree decompositions provide for recursive algorithms
  - ▶ Bounding tree-width gives polynomial time execution
- 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

- Tree-width introduced by Robertson and Seymour
  - ▶ Tree decompositions provide for recursive algorithms
  - ▶ Bounding tree-width gives polynomial time execution
- 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

- Tree-width introduced by Robertson and Seymour
  - ▶ Tree decompositions provide for recursive algorithms
  - ▶ Bounding tree-width gives polynomial time execution
- 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

- Tree-width introduced by Robertson and Seymour
  - ▶ Tree decompositions provide for recursive algorithms
  - ▶ Bounding tree-width gives polynomial time execution
- 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

- Tree-width introduced by Robertson and Seymour
  - ▶ Tree decompositions provide for recursive algorithms
  - ▶ Bounding tree-width gives polynomial time execution
- 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.

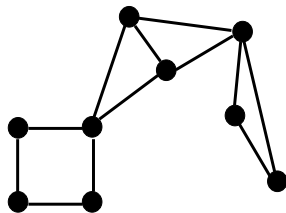
# Overview

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

## Recall...

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

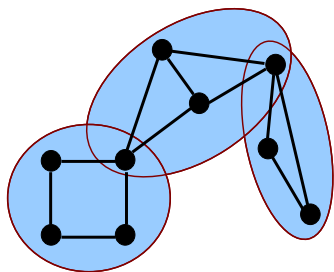
A graph has **tree-width**  $\leq k$  if it can be covered by sub-graphs of size  $\leq (k + 1)$  in a tree-like fashion.



## Recall...

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

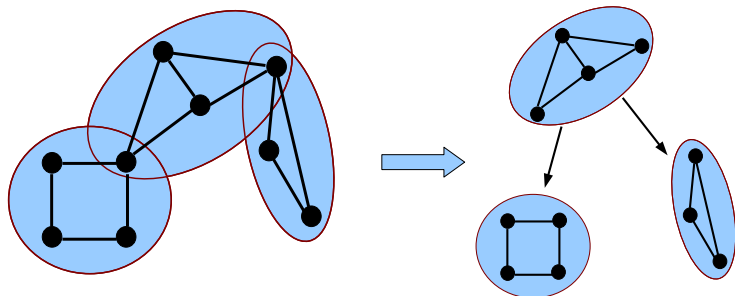
A graph has **tree-width**  $\leq k$  if it can be covered by sub-graphs of size  $\leq (k + 1)$  in a tree-like fashion.



## Recall...

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

A **tree decomposition** of a graph  $\mathcal{G}$  is a tuple  $(\mathcal{T}, (X_t)_{t \in V(\mathcal{T})})$  such that:

- $\mathcal{T}$  is a tree
- $X_t$  cover  $V(\mathcal{G})$
- For every edge  $(u, v) \in E(\mathcal{G})$ , there is a  $t \in V(\mathcal{T})$  with  $\{u, v\} \subseteq X_t$
- For every  $t'$  on the path from  $t$  to  $t''$ ,  $X_t \cap X_{t''} \subseteq X_{t'}$

The width of a tree decomposition is  $\max_{t \in V(\mathcal{T})} |X_t| - 1$ .

The tree-width of a graph is the minimal width of all its tree decompositions.

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

# Tree-width

A tree decomposition of a graph  $\mathcal{G}$  is a tuple  $(\mathcal{T}, (X_t)_{t \in V(\mathcal{T})})$  such that:

- $\mathcal{T}$  is a tree
- $X_t$  cover  $V(\mathcal{G})$
- For every edge  $(u, v) \in E(\mathcal{G})$ , there is a  $t \in V(\mathcal{T})$  with  $\{u, v\} \subseteq X_t$
- For every  $t'$  on the path from  $t$  to  $t''$ ,  $X_t \cap X_{t''} \subseteq X_{t'}$

The **width** of a tree decomposition is  $\max_{t \in V(\mathcal{T})} |X_t| - 1$ .

The tree-width of a graph is the minimal width of all its tree decompositions.

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

# Tree-width

A tree decomposition of a graph  $\mathcal{G}$  is a tuple  $(\mathcal{T}, (X_t)_{t \in V(\mathcal{T})})$  such that:

- $\mathcal{T}$  is a tree
- $X_t$  cover  $V(\mathcal{G})$
- For every edge  $(u, v) \in E(\mathcal{G})$ , there is a  $t \in V(\mathcal{T})$  with  $\{u, v\} \subseteq X_t$
- For every  $t'$  on the path from  $t$  to  $t''$ ,  $X_t \cap X_{t''} \subseteq X_{t'}$

The width of a tree decomposition is  $\max_{t \in V(\mathcal{T})} |X_t| - 1$ .

The **tree-width** of a graph is the minimal width of all its tree decompositions.

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

# Tree-width

A tree decomposition of a graph  $\mathcal{G}$  is a tuple  $(\mathcal{T}, (X_t)_{t \in V(\mathcal{T})})$  such that:

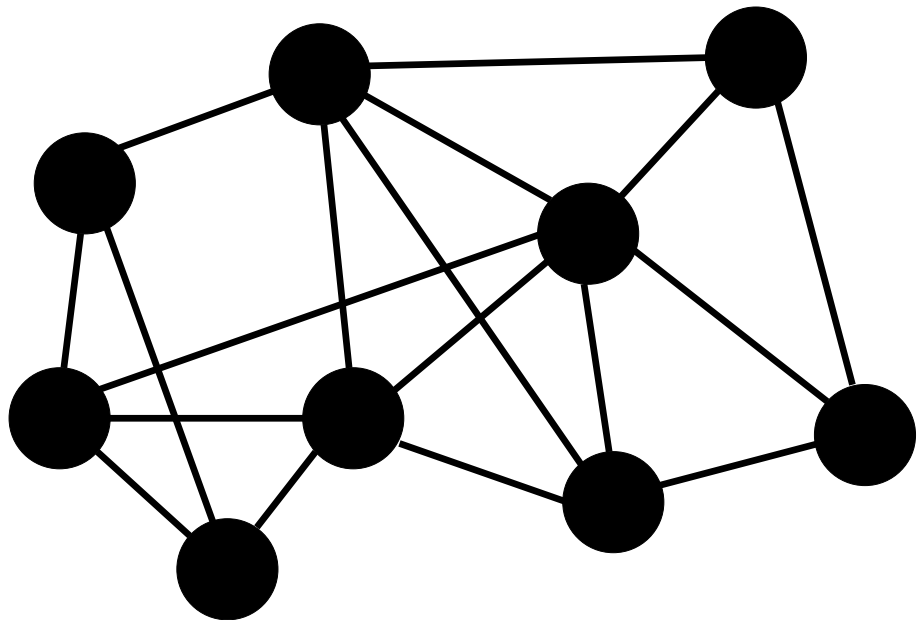
- $\mathcal{T}$  is a tree
- $X_t$  cover  $V(\mathcal{G})$
- For every edge  $(u, v) \in E(\mathcal{G})$ , there is a  $t \in V(\mathcal{T})$  with  $\{u, v\} \subseteq X_t$
- For every  $t'$  on the path from  $t$  to  $t''$ ,  $X_t \cap X_{t''} \subseteq X_{t'}$

The width of a tree decomposition is  $\max_{t \in V(\mathcal{T})} |X_t| - 1$ .

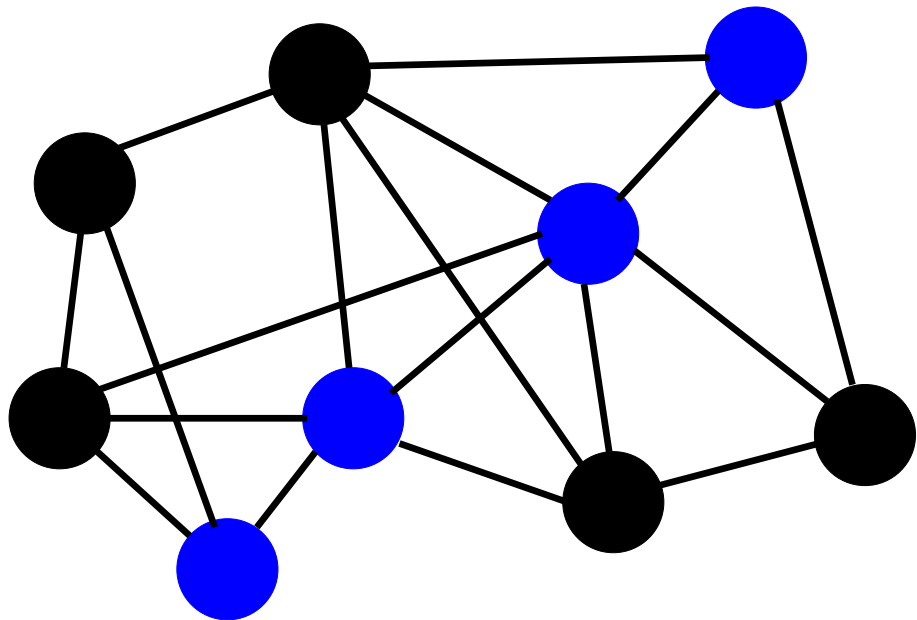
The tree-width of a graph is the minimal width of all its tree decompositions.

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

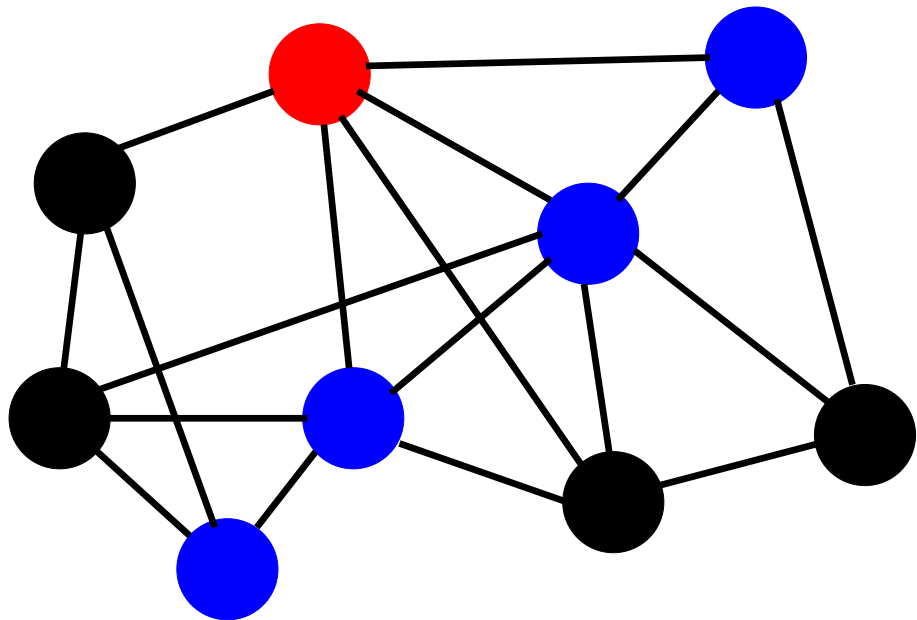
## Cops and robber game



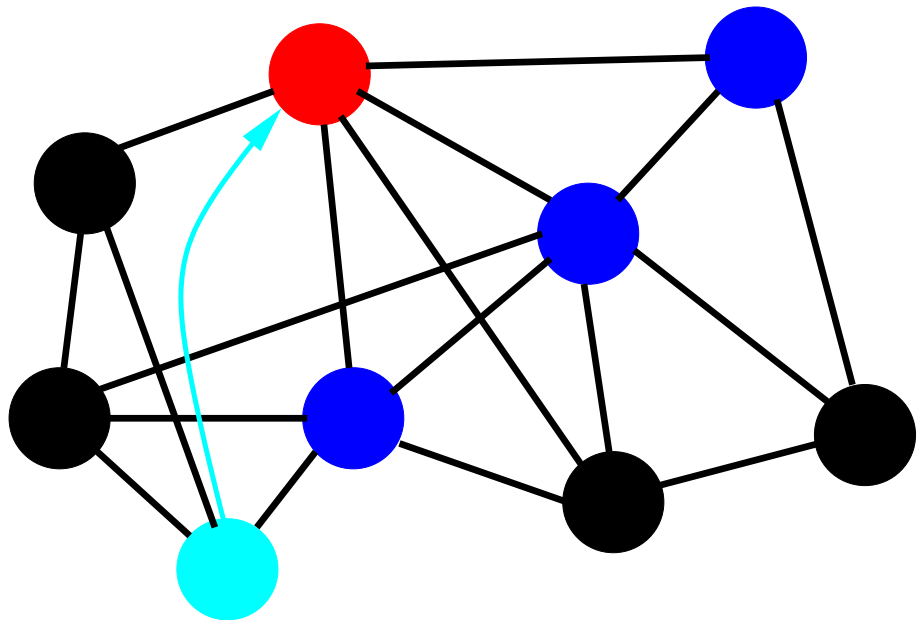
## Cops and robber game



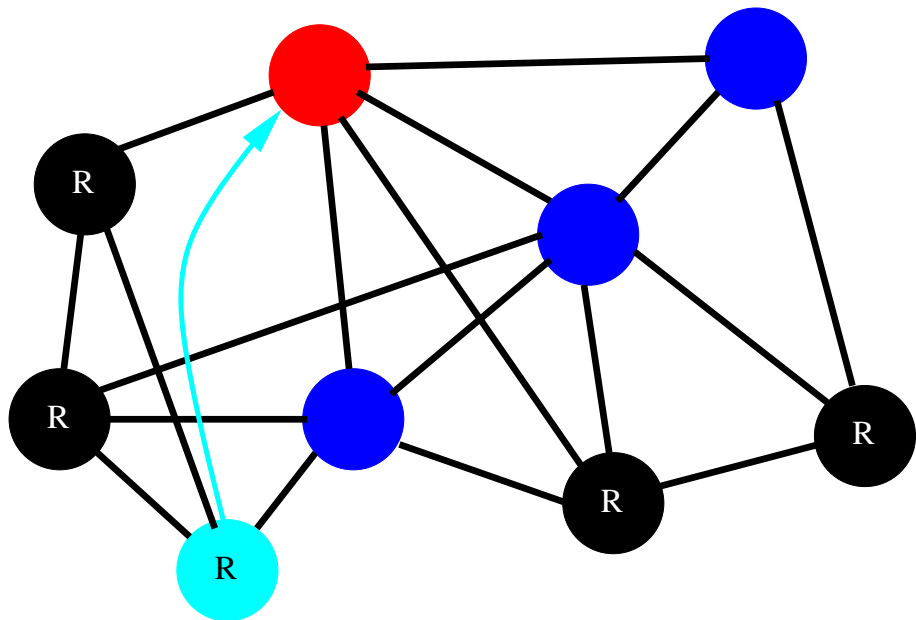
## Cops and robber game



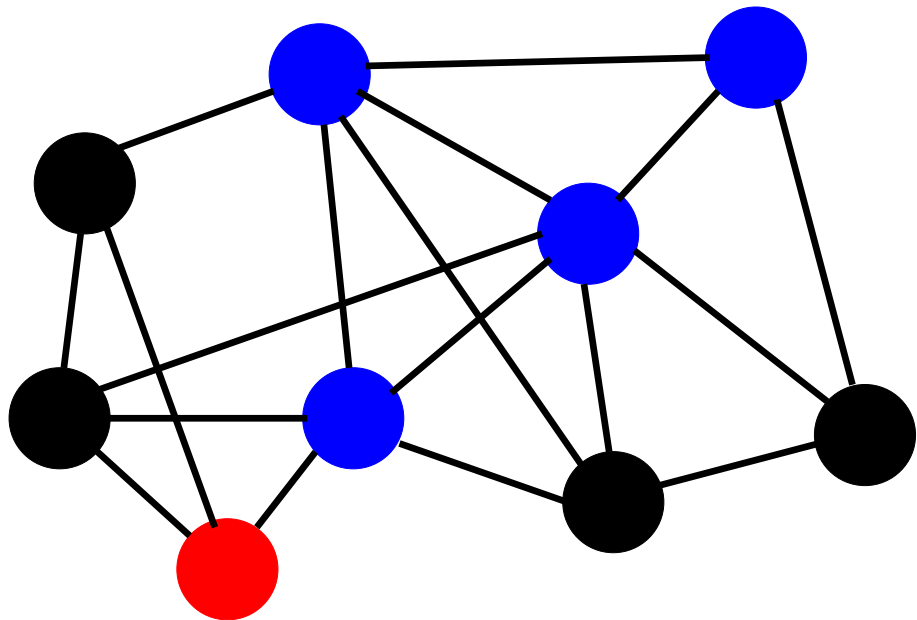
# Cops and robber game



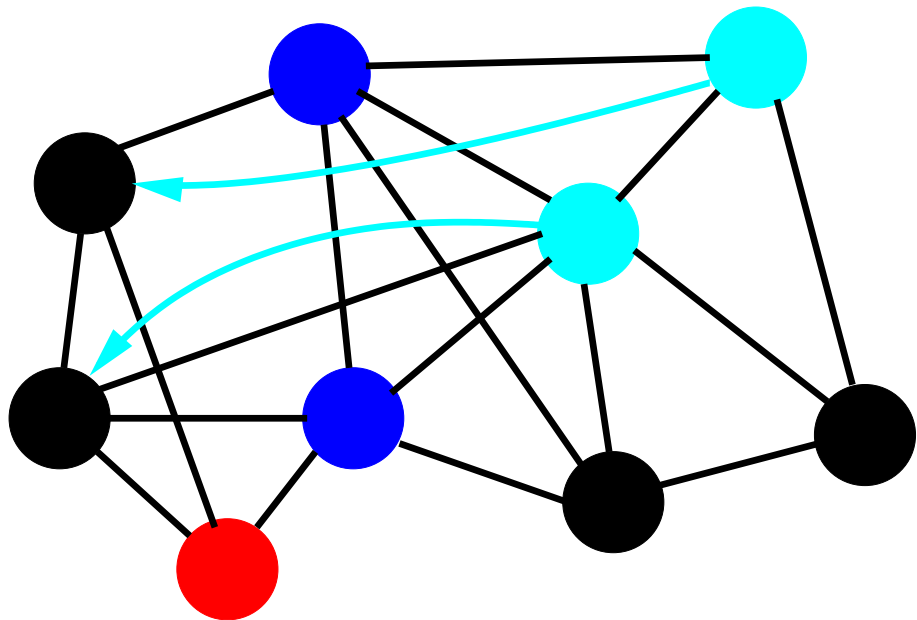
# Cops and robber game



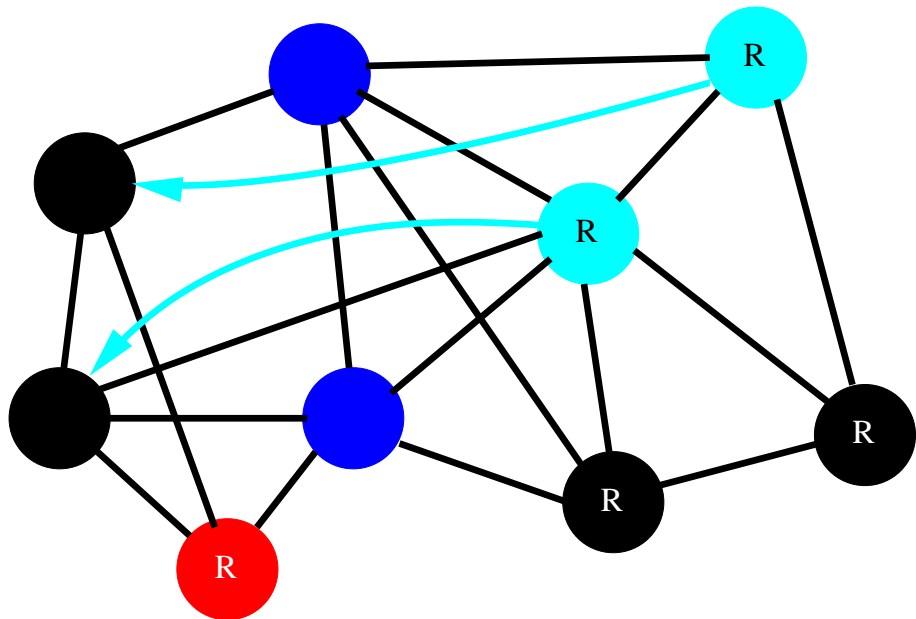
# Cops and robber game



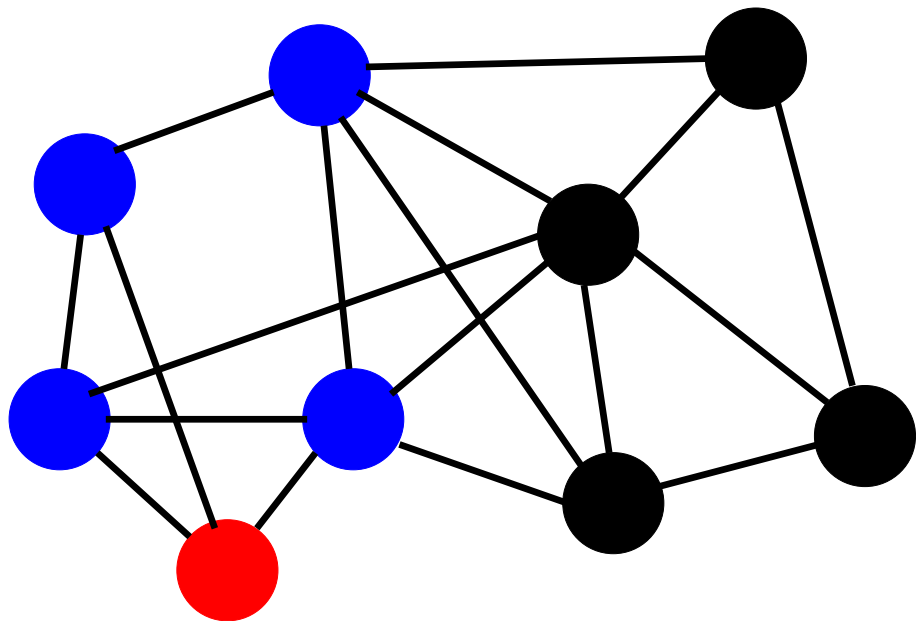
# Cops and robber game



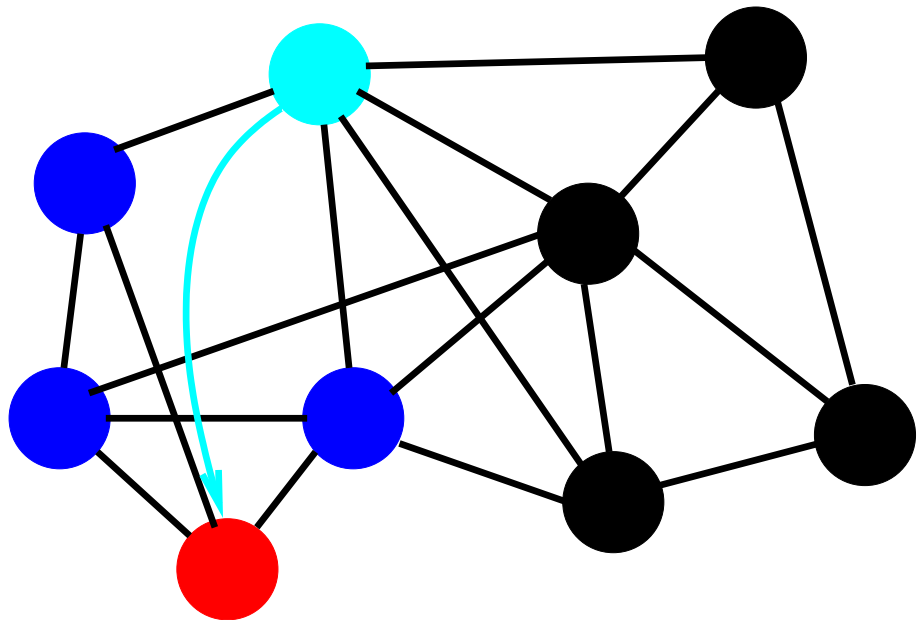
# Cops and robber game



# Cops and robber game



# Cops and robber game



# Cops and robber game

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 and robber game

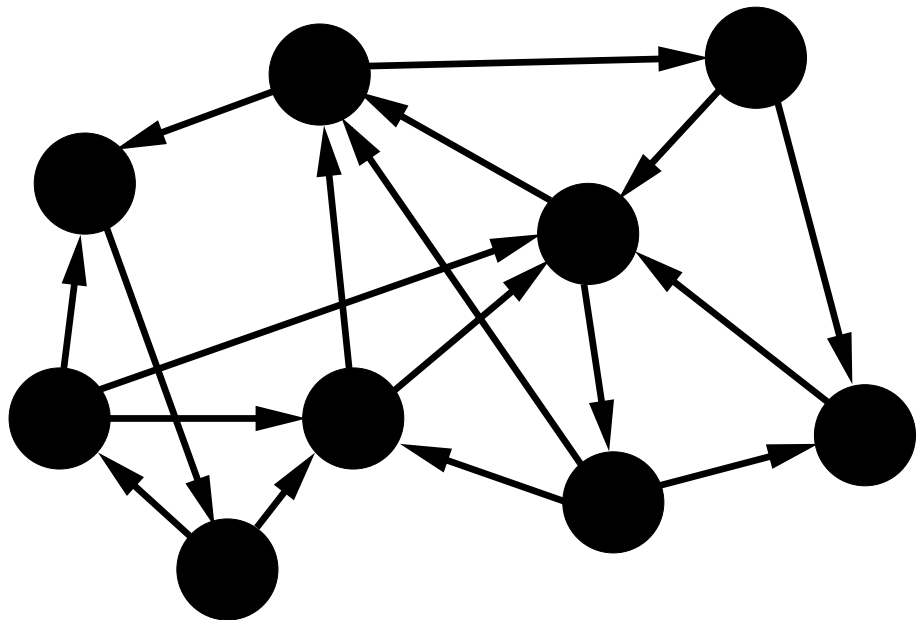
Theorem (Seymour and Thomas 1993)

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

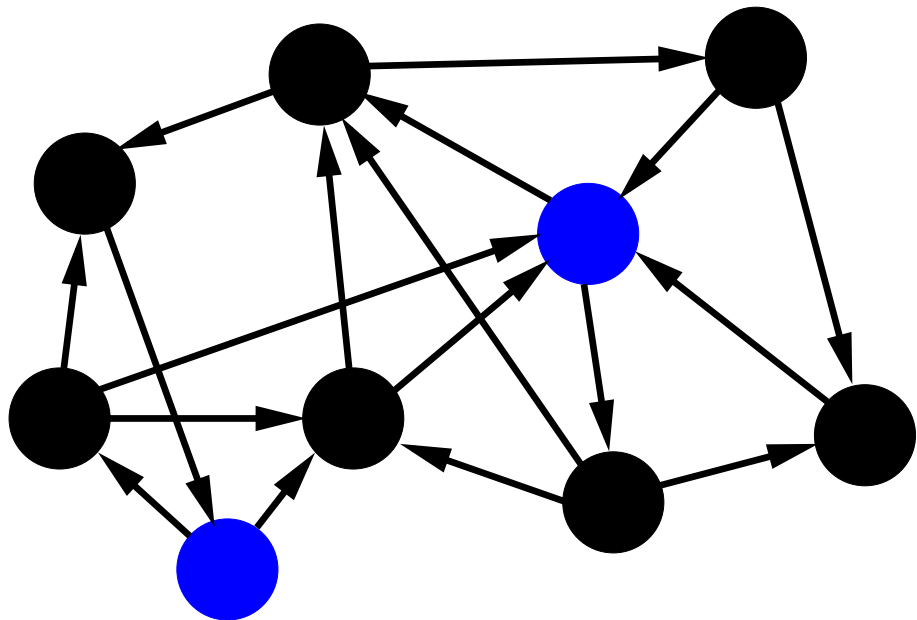
Question

What about directed graphs?

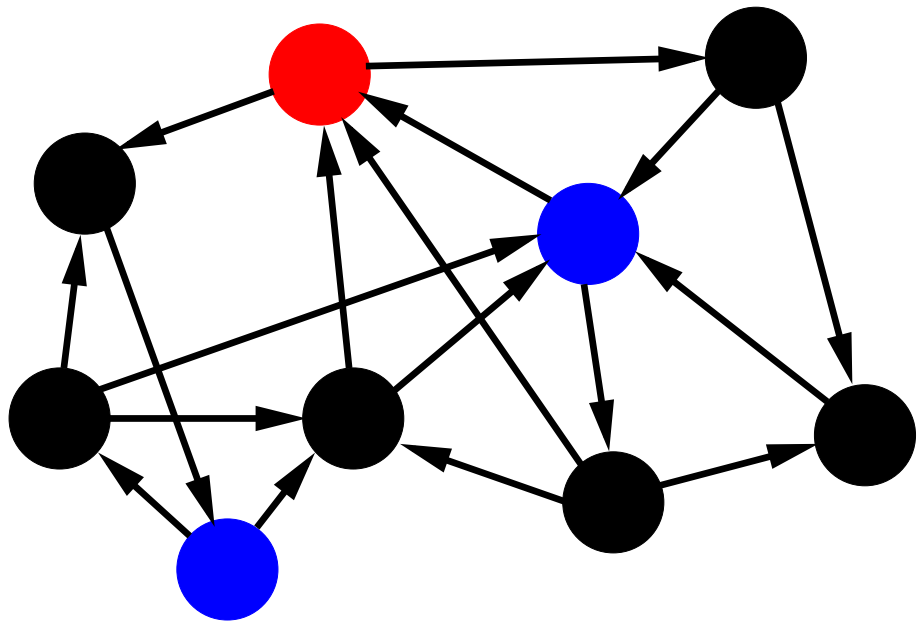
## Directed cops and robbers



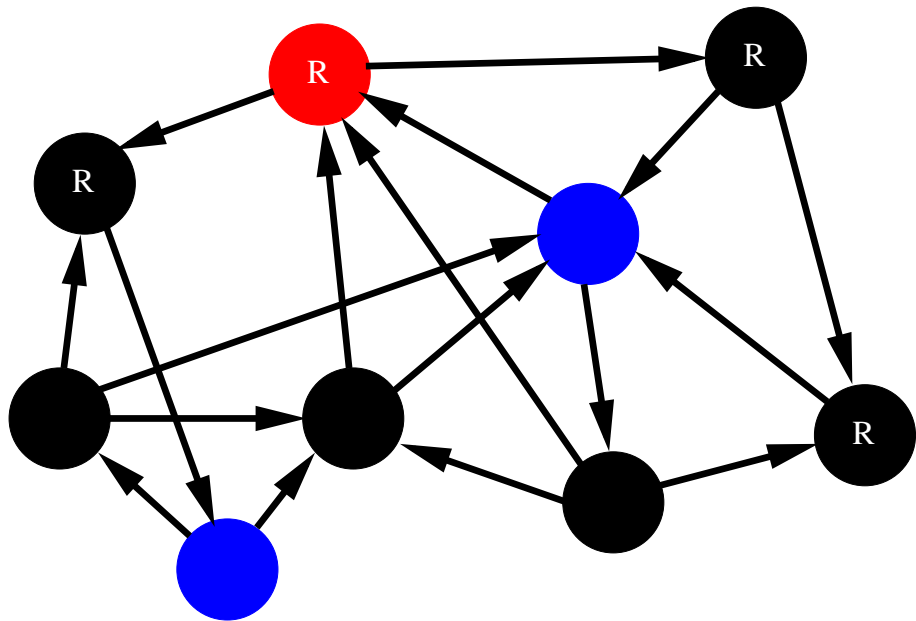
## Directed cops and robbers



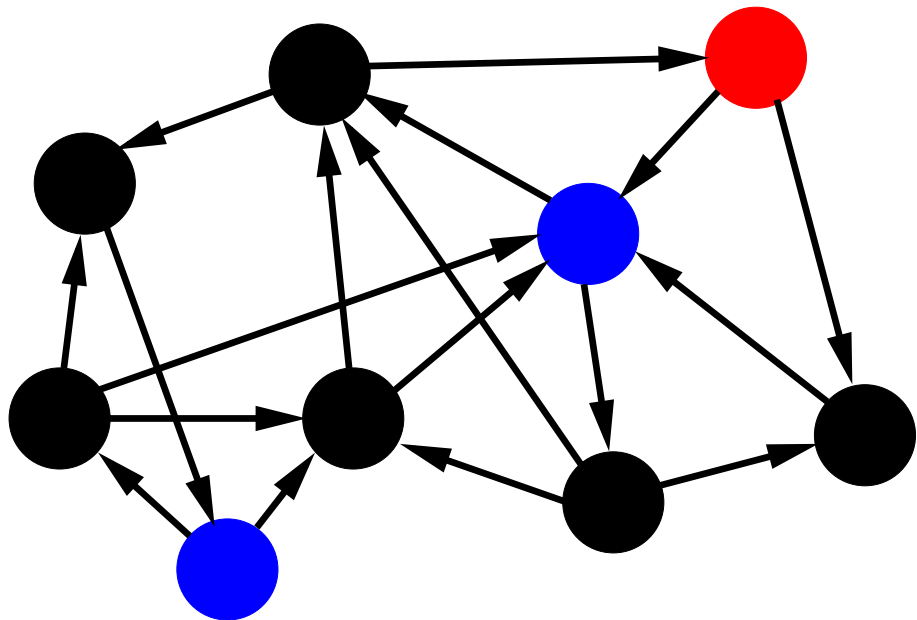
## Directed cops and robbers



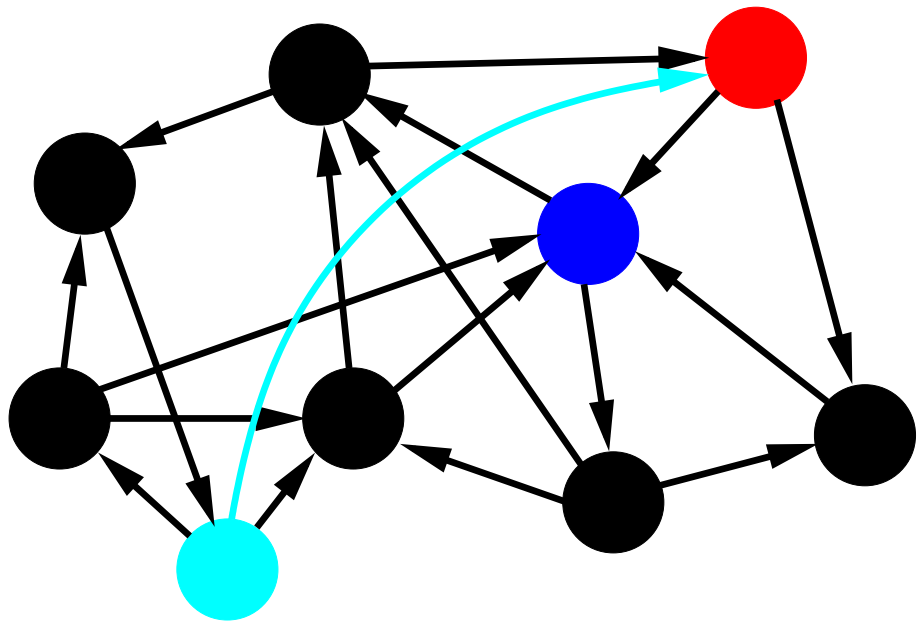
## Directed cops and robbers



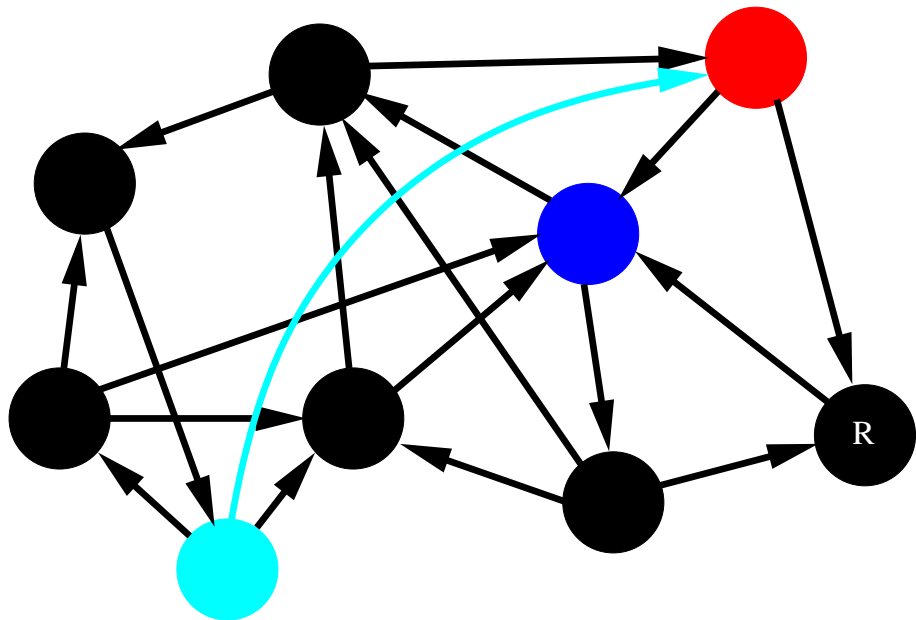
## Directed cops and robbers



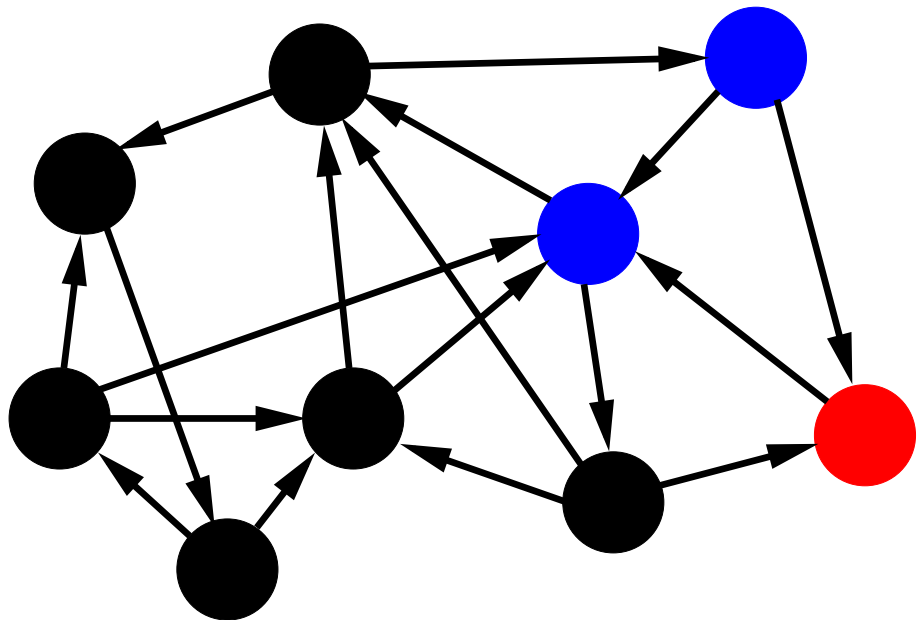
## Directed cops and robbers



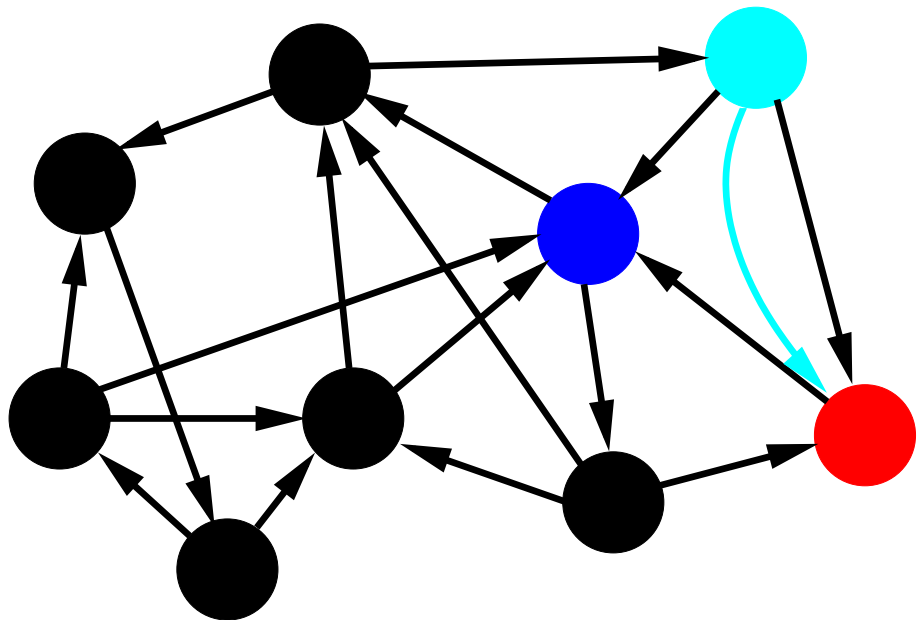
## Directed cops and robbers



## Directed cops and robbers



## Directed cops and robbers



# Observations

Let  $\text{game-width}(\mathcal{G})$  be the minimal number of cops required to catch a robber on  $\mathcal{G}$ .

- $\text{directed tree-width}(\mathcal{G}) \leq \text{game-width}(\mathcal{G}) \leq \text{tree-width}(\mathcal{G})$
- $\text{game-width}(\mathcal{G}) = 1$  iff  $\mathcal{G}$  is a DAG
- game-width of directed union is maximum width of components
- game-width is **not** preserved under edge reversal

## Problem

Find a decomposition that corresponds to game-width

# Observations

Let  $\text{game-width}(\mathcal{G})$  be the minimal number of cops required to catch a robber on  $\mathcal{G}$ .

- $\text{directed tree-width}(\mathcal{G}) \leq \text{game-width}(\mathcal{G}) \leq \text{tree-width}(\mathcal{G})$
- $\text{game-width}(\mathcal{G}) = 1$  iff  $\mathcal{G}$  is a DAG
- game-width of directed union is maximum width of components
- game-width is **not** preserved under edge reversal

## Problem

Find a decomposition that corresponds to game-width

# Observations

Let  $\text{game-width}(\mathcal{G})$  be the minimal number of cops required to catch a robber on  $\mathcal{G}$ .

- $\text{directed tree-width}(\mathcal{G}) \leq \text{game-width}(\mathcal{G}) \leq \text{tree-width}(\mathcal{G})$
- $\text{game-width}(\mathcal{G}) = 1$  iff  $\mathcal{G}$  is a DAG
- game-width of directed union is maximum width of components
- game-width is **not** preserved under edge reversal

## Problem

Find a decomposition that corresponds to game-width

# Observations

Let  $\text{game-width}(\mathcal{G})$  be the minimal number of cops required to catch a robber on  $\mathcal{G}$ .

- $\text{directed tree-width}(\mathcal{G}) \leq \text{game-width}(\mathcal{G}) \leq \text{tree-width}(\mathcal{G})$
- $\text{game-width}(\mathcal{G}) = 1$  iff  $\mathcal{G}$  is a DAG
- game-width of directed union is maximum width of components
- game-width is **not** preserved under edge reversal

## Problem

Find a decomposition that corresponds to game-width

# Observations

Let  $\text{game-width}(\mathcal{G})$  be the minimal number of cops required to catch a robber on  $\mathcal{G}$ .

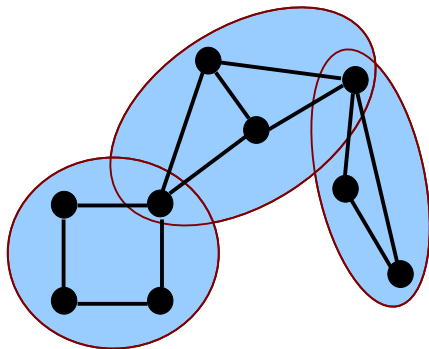
- $\text{directed tree-width}(\mathcal{G}) \leq \text{game-width}(\mathcal{G}) \leq \text{tree-width}(\mathcal{G})$
- $\text{game-width}(\mathcal{G}) = 1$  iff  $\mathcal{G}$  is a DAG
- game-width of directed union is maximum width of components
- game-width is **not** preserved under edge reversal

## Problem

Find a decomposition that corresponds to game-width

## Another observation...

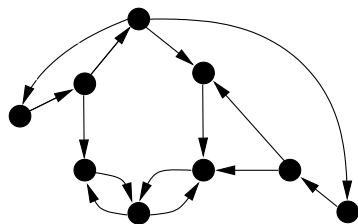
In a tree decomposition, an edge only leaves a subtree through its connection with the rest of the tree



## DAG-width

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

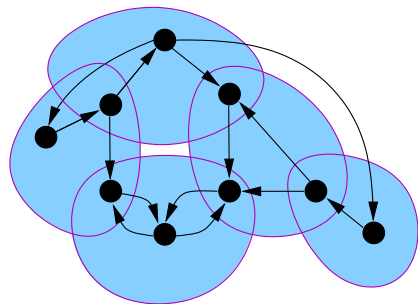
A graph has **DAG-width**  $\leq k$  if it can be covered by **subsets** of size  $\leq k$  in a DAG-like fashion such that an edge only leaves a sub-DAG through its (root's) connection with the rest of the DAG



# DAG-width

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

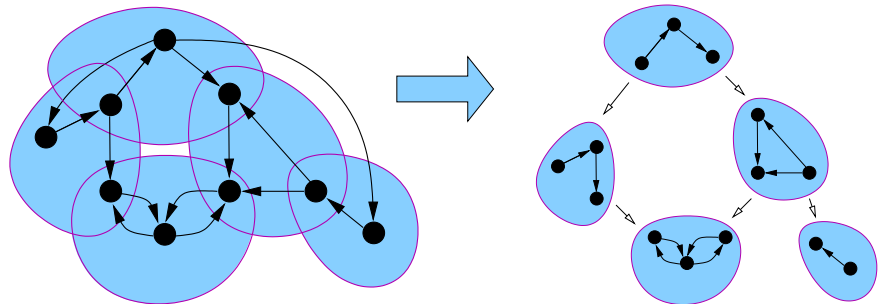
A graph has **DAG-width**  $\leq k$  if it can be covered by **subsets** of size  $\leq k$  in a DAG-like fashion such that an edge only leaves a sub-DAG through its (root's) connection with the rest of the DAG



# DAG-width

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

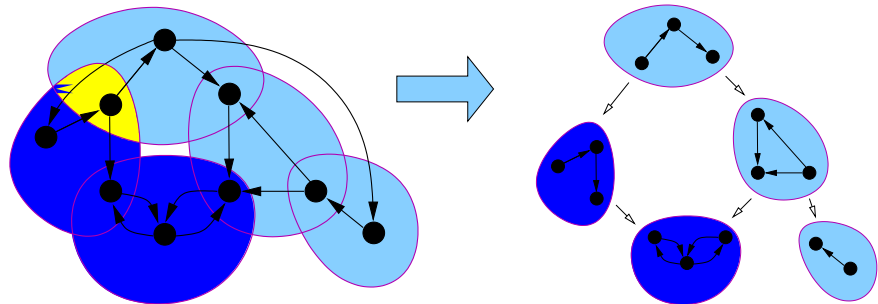
A graph has **DAG-width**  $\leq k$  if it can be covered by **subsets** of size  $\leq k$  in a DAG-like fashion such that an edge only leaves a sub-DAG through its (root's) connection with the rest of the DAG



# DAG-width

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

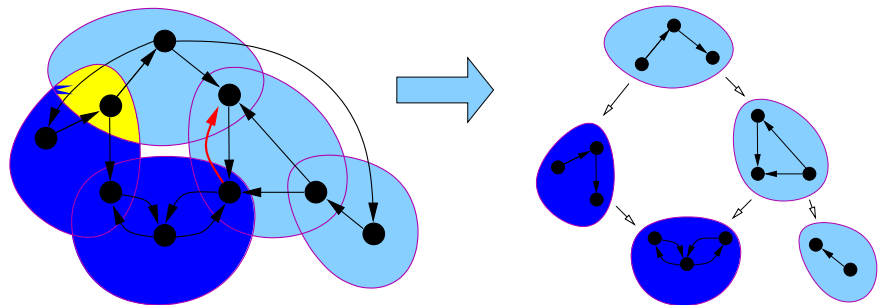
A graph has **DAG-width**  $\leq k$  if it can be covered by **subsets** of size  $\leq k$  in a DAG-like fashion such that an edge only leaves a sub-DAG through its (root's) connection with the rest of the DAG



# DAG-width

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

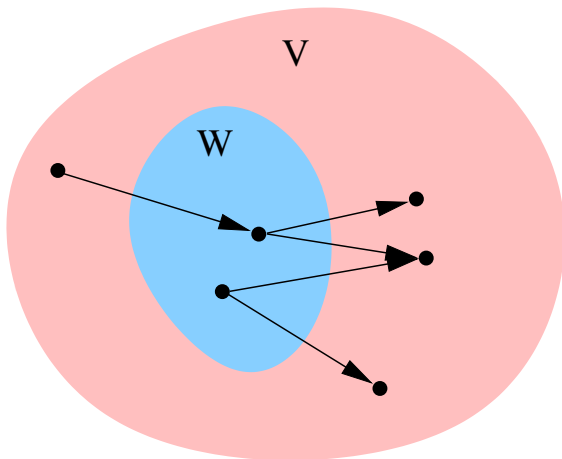
A graph has **DAG-width**  $\leq k$  if it can be covered by **subsets** of size  $\leq k$  in a DAG-like fashion such that an edge only leaves a sub-DAG through its (root's) connection with the rest of the DAG



# Guarding

## Definition

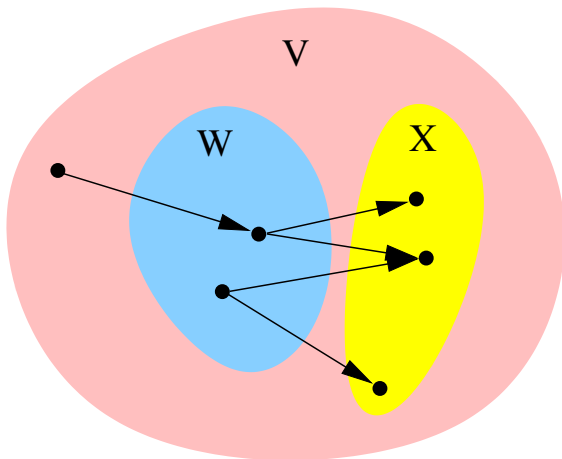
If  $\mathcal{G}$  is a directed graph,  $W, X \subseteq V(\mathcal{G})$ , we say  $X$  **guards**  $W$  if every edge which leaves  $W$  ends in  $X$ .



# Guarding

## Definition

If  $\mathcal{G}$  is a directed graph,  $W, X \subseteq V(\mathcal{G})$ , we say  $X$  **guards**  $W$  if every edge which leaves  $W$  ends in  $X$ .



# 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  $(\bigcup_{d \preceq_{\mathcal{D}} d'} X_{d'}) \setminus X_c$ . 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.

# 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  $(\bigcup_{d \preceq_{\mathcal{D}} d'} X_{d'}) \setminus X_c$ . 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.

# 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  $(\bigcup_{d \preceq_{\mathcal{D}} d'} X_{d'}) \setminus X_c$ . 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 PTIME*
- *Given  $\mathcal{G}$  and  $k$ , deciding if  $\mathcal{G}$  has DAG-width  $\leq k$  is NP-complete*

# 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 PTIME*
- *Given  $\mathcal{G}$  and  $k$ , deciding if  $\mathcal{G}$  has DAG-width  $\leq k$  is NP-complete*

## More results...

- Results from game-width carry over to DAG-width
  - ▶  $\text{dtw}(\mathcal{G}) \leq \text{game-width}(\mathcal{G}) \leq \text{DAG-width}(\mathcal{G}) \leq \text{tw}(\mathcal{G})$
  - ▶ Directed unions
- $\text{DAG-width}(\mathcal{G}) \leq \text{entanglement}(\mathcal{G}) + 1$
- $\text{DAG-width}(\mathcal{G}) \leq \text{directed path-width}(\mathcal{G})$

### Theorem

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

## More results...

- Results from game-width carry over to DAG-width
  - ▶  $\text{dtw}(\mathcal{G}) \leq \text{game-width}(\mathcal{G}) \leq \text{DAG-width}(\mathcal{G}) \leq \text{tw}(\mathcal{G})$
  - ▶ Directed unions
- $\text{DAG-width}(\mathcal{G}) \leq \text{entanglement}(\mathcal{G}) + 1$
- $\text{DAG-width}(\mathcal{G}) \leq \text{directed path-width}(\mathcal{G})$

### Theorem

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

## More results...

- Results from game-width carry over to DAG-width
  - ▶  $\text{dtw}(\mathcal{G}) \leq \text{game-width}(\mathcal{G}) \leq \text{DAG-width}(\mathcal{G}) \leq \text{tw}(\mathcal{G})$
  - ▶ Directed unions
- $\text{DAG-width}(\mathcal{G}) \leq \text{entanglement}(\mathcal{G}) + 1$
- $\text{DAG-width}(\mathcal{G}) \leq \text{directed path-width}(\mathcal{G})$

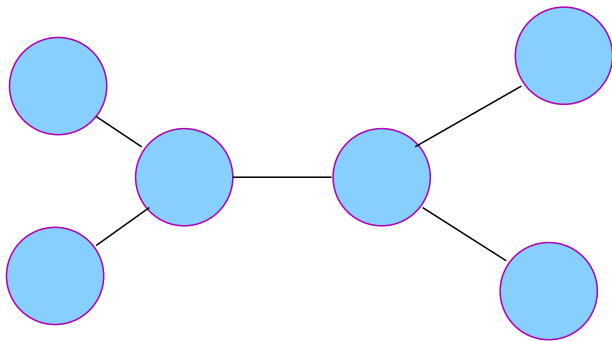
### Theorem

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

# Parity games algorithm

Similar to Obdržálek's algorithm for bounded tree-width

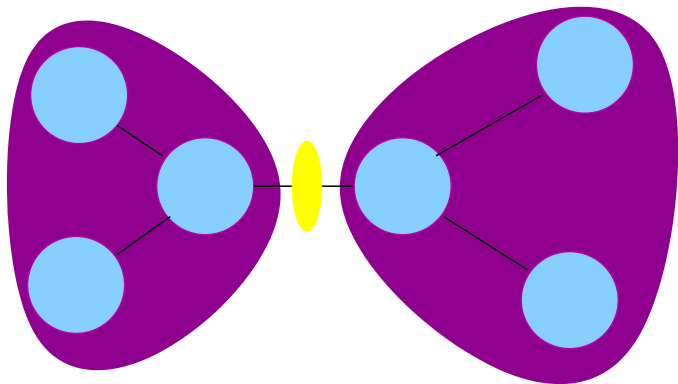
- 1 An edge (or node) of a tree decomposition separates the graph
- 2 Positional strategies can then be represented as functions from the interface to itself (border)
- 3 Compute borders in a bottom-up manner



# Parity games algorithm

Similar to Obdržálek's algorithm for bounded tree-width

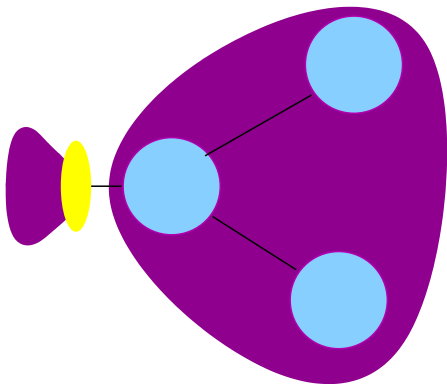
- 1 An edge (or node) of a tree decomposition separates the graph
- 2 Positional strategies can then be represented as functions from the interface to itself (border)
- 3 Compute borders in a bottom-up manner



# Parity games algorithm

Similar to Obdržálek's algorithm for bounded tree-width

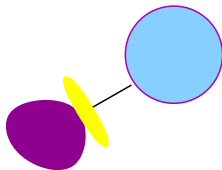
- 1 An edge (or node) of a tree decomposition separates the graph
- 2 Positional strategies can then be represented as functions from the interface to itself (border)
- 3 Compute borders in a bottom-up manner



# Parity games algorithm

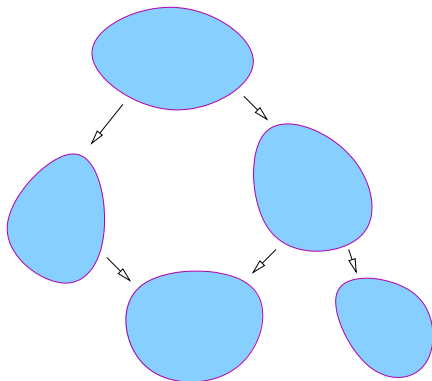
Similar to Obdržálek's algorithm for bounded tree-width

- 1 An edge (or node) of a tree decomposition separates the graph
- 2 Positional strategies can then be represented as functions from the interface to itself (border)
- 3 Compute borders in a bottom-up manner



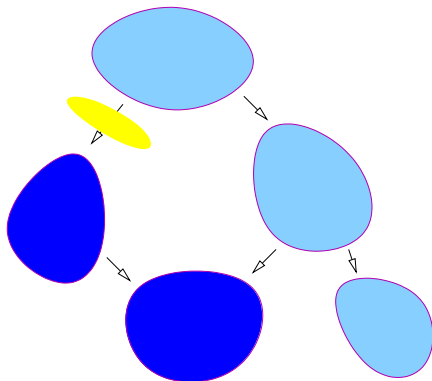
# Extension to DAG-decompositions

- Interface covers edges **leaving** sub-DAG
- Problem 1: Handling edges entering sub-DAG
  - ▶ Solution: Use functions from sub-DAG to interface (frontier)
- Problem 2: Adding vertices to frontiers



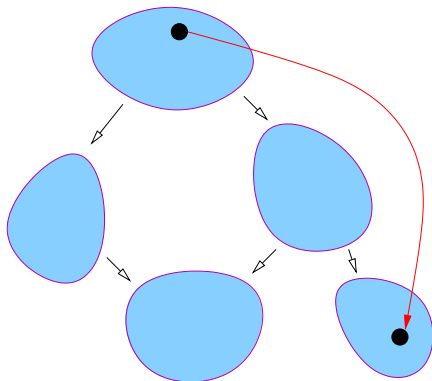
# Extension to DAG-decompositions

- Interface covers edges **leaving** sub-DAG
- Problem 1: Handling edges entering sub-DAG
  - ▶ Solution: Use functions from sub-DAG to interface (frontier)
- Problem 2: Adding vertices to frontiers



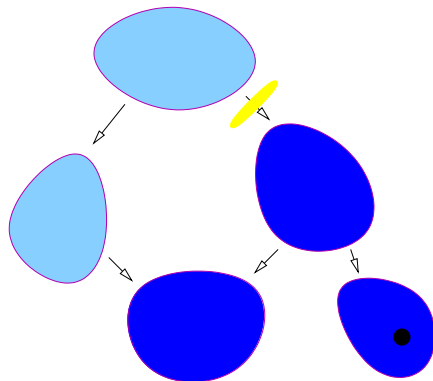
# Extension to DAG-decompositions

- Interface covers edges leaving sub-DAG
- **Problem 1:** Handling edges entering sub-DAG
  - ▶ Solution: Use functions from sub-DAG to interface (frontier)
- Problem 2: Adding vertices to frontiers



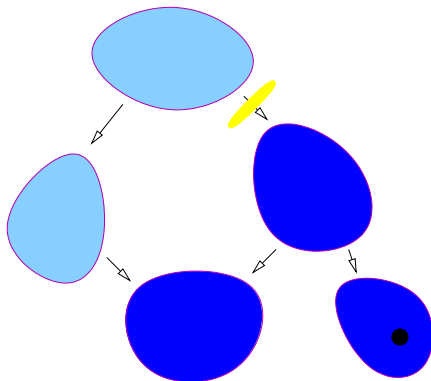
# Extension to DAG-decompositions

- Interface covers edges leaving sub-DAG
- Problem 1: Handling edges entering sub-DAG
  - ▶ **Solution:** Use functions from sub-DAG to interface (frontier)
- Problem 2: Adding vertices to frontiers



## Extension to DAG-decompositions

- Interface covers edges leaving sub-DAG
- Problem 1: Handling edges entering sub-DAG
  - ▶ Solution: Use functions from sub-DAG to interface (frontier)
- **Problem 2:** Adding vertices to frontiers



# 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 results on bounded tree-width and entanglement.
- 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 results on bounded tree-width and entanglement.
- 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 results on bounded tree-width and entanglement.
- 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 results on bounded tree-width and entanglement.
- Are monotone strategies sufficient?
- Generalisation of havens, brambles, minors, separators?
- Generalisation of Courcelle's theorem?