Trees, Tree width, Hypertree width, Fractional hypertree width

Dániel Marx

Humboldt-Universität zu Berlin

Institut für Informatik

Logik in der Informatik

dmarx@informatik.hu-berlin.de

July 15, 2005

Overview

Main theme: tree-structured problems are easy to solve.

- trees
- tree width
- hypertree width
- fractional hypertree width
  (joint work with Martin Grohe)

The Party Problem

PARTY PROBLEM

Problem: Invite some collegues for a party.

Maximize: The number of interesting people invited.

Constraint: Everyone should be having fun.

The Party Problem

PARTY PROBLEM

Problem: Invite some collegues for a party.

Maximize: The number of interesting people invited.

Constraint: Everyone should be having fun.

Do not invite a collegue and his direct boss at the same time!

The Party Problem

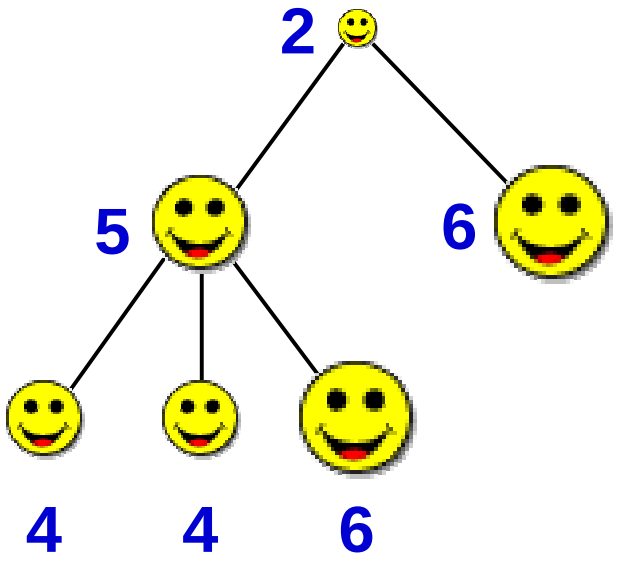
PARTY PROBLEM

Problem: Invite some collegues for a party.

Maximize: The number of interesting people invited.

Constraint: Everyone should be having fun.

Do not invite a collegue and his direct boss at the same time!



- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

The Party Problem

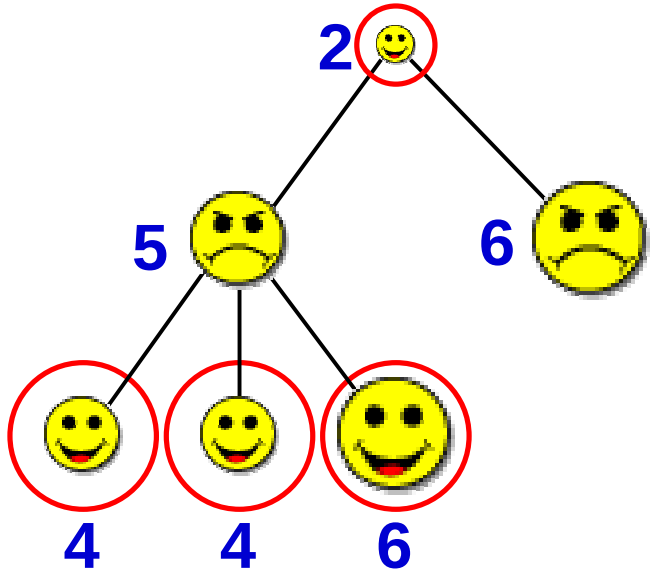
PARTY PROBLEM

Problem: Invite some collegues for a party.

Maximize: The number of interesting people invited.

Constraint: Everyone should be having fun.

Do not invite a collegue and his direct boss at the same time!



- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

Solving the Party Problem

Dynamic programming: We solve subproblems that depend on each other.

T_v: the subtree rooted at v.

$A[v]$: max. weight of an independent set in T_v

$B[v]$: max. weight of an independent set in T_v that does not contain v

Goal: determine $A[r]$ for the root r.

Solving the Party Problem

Dynamic programming: We solve subproblems that depend on each other.

T_v: the subtree rooted at v.
$A[v]$: max. weight of an independent set in T_v
$B[v]$: max. weight of an independent set in T_v that does not contain v

Goal: determine $A[r]$ for the root r.

Method:

Assume $v_1, \ldots, v_k$ are the children of v. Use the recurrence relations

$$B[v] = \sum_{i=1}^{k} A[v_i]$$
$$A[v] = \max\{B[v]\ ,\ w(v) + \sum_{i=1}^{k} B[v_i]\}$$

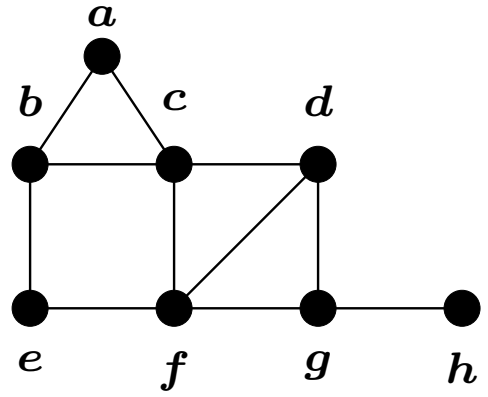
The values $A[v]$ and $B[v]$ can be calculated in a bottom-up order (the leaves are trivial).

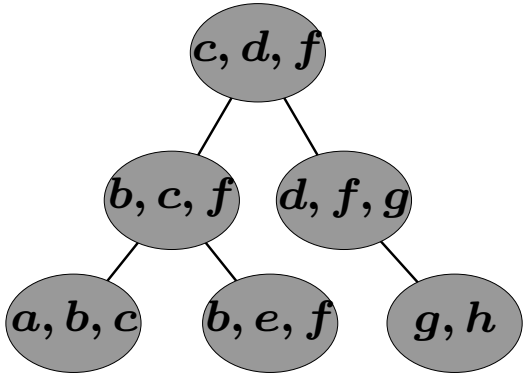
Tree width

Tree width: A measure of how “tree-like” is the graph. (Introduced by Robertson and Seymour.)

Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.



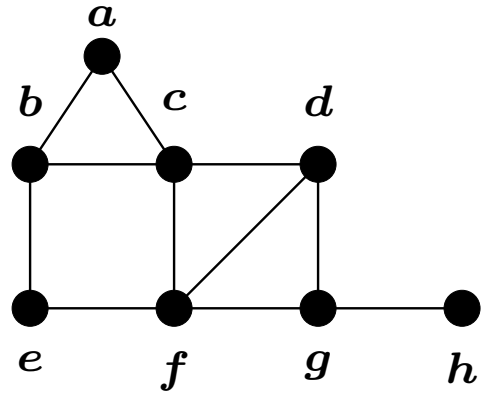


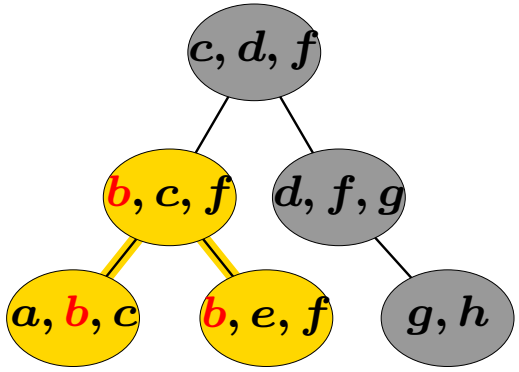
Tree width

Tree width: A measure of how “tree-like” is the graph. (Introduced by Robertson and Seymour.)

Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.



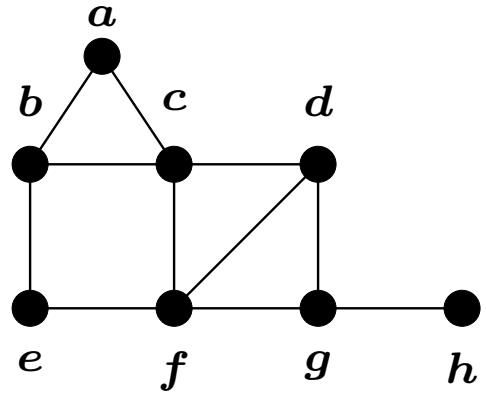


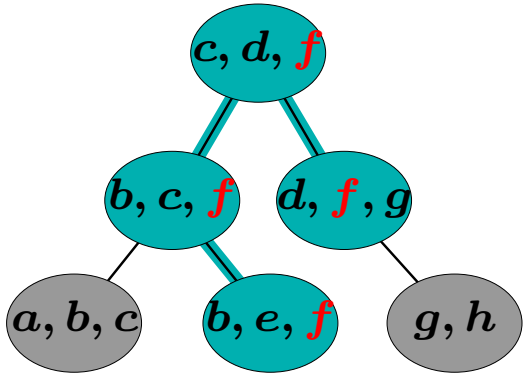
Tree width

Tree width: A measure of how “tree-like” is the graph. (Introduced by Robertson and Seymour.)

Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.



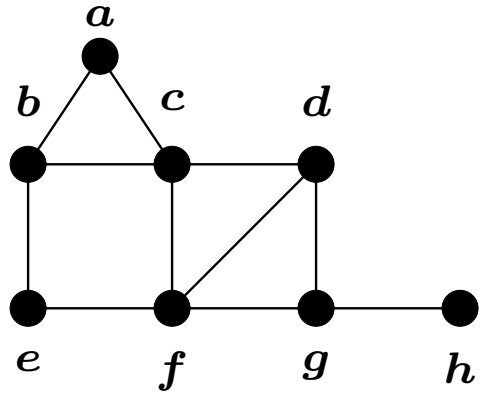


Tree width

Tree width: A measure of how "tree-like" is the graph. (Introduced by Robertson and Seymour.)

Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

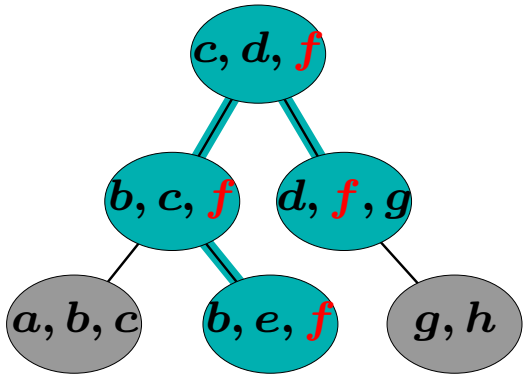
1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.



Width of the decomposition: size of the largest bag minus 1.

Tree width: width of the best decomposition.

Fact: Tree width $= 1 \iff$ graph is a forest

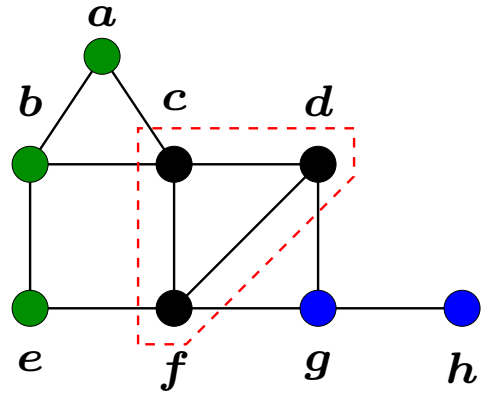


Tree width

Tree width: A measure of how "tree-like" is the graph. (Introduced by Robertson and Seymour.)

Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

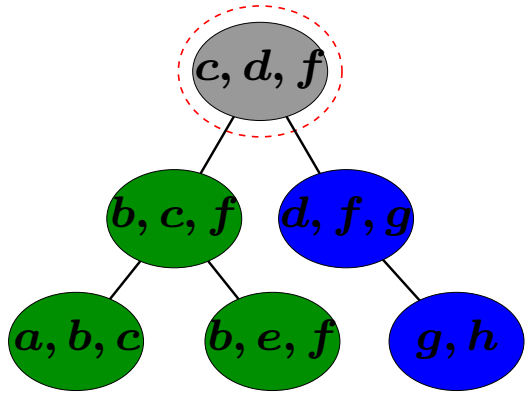
1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.



Width of the decomposition: size of the largest bag minus 1.

Tree width: width of the best decomposition.

Fact: Tree width $= 1 \iff$ graph is a forest

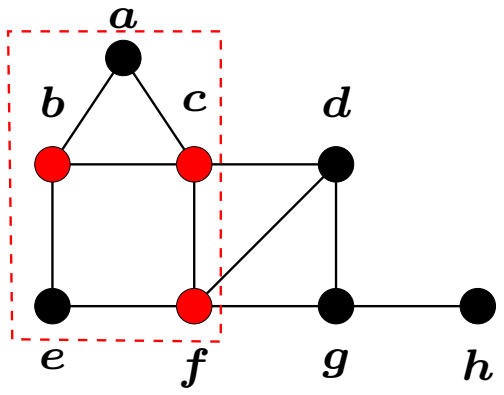


Tree width

Tree width: A measure of how "tree-like" is the graph. (Introduced by Robertson and Seymour.)

Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

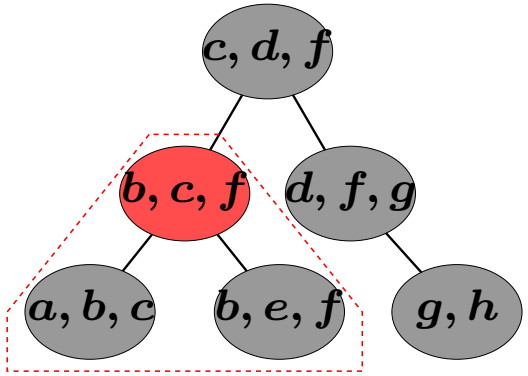
1. If u and v are neighbors, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.



Width of the decomposition: size of the largest bag minus 1.

Tree width: width of the best decomposition.

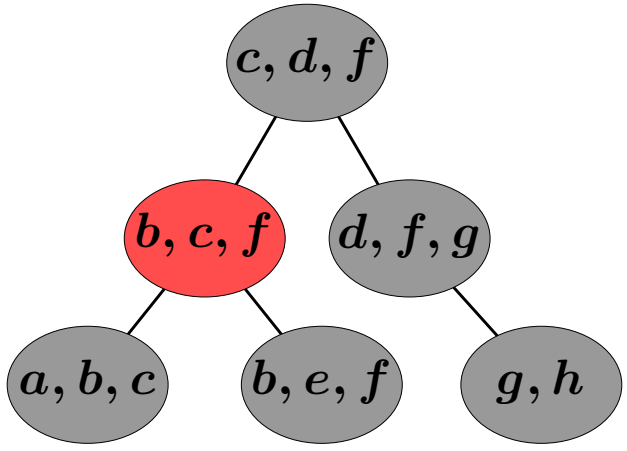
Fact: Tree width $= 1 \iff$ graph is a forest



Using the tree decomposition

Fact: MAX WEIGHTED INDEPENDENT SET can be solved in time $O(2^w \cdot n^2)$ on a graph with n vertices and tree width at most w.

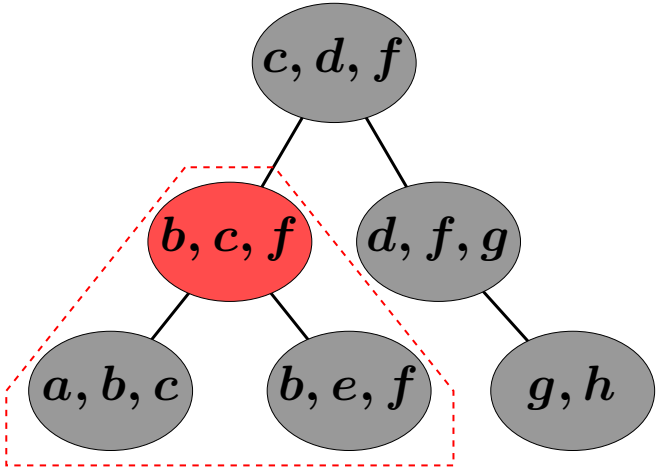
Algorithm: Dynamic programming on the tree decomposition. For each bag, solve the problem for the vertices contained in the descendants of the bag.



Using the tree decomposition

Fact: MAX WEIGHTED INDEPENDENT SET can be solved in time $O(2^w \cdot n^2)$ on a graph with n vertices and tree width at most w.

Algorithm: Dynamic programming on the tree decomposition. For each bag, solve the problem for the vertices contained in the descendants of the bag.



Using the tree decomposition

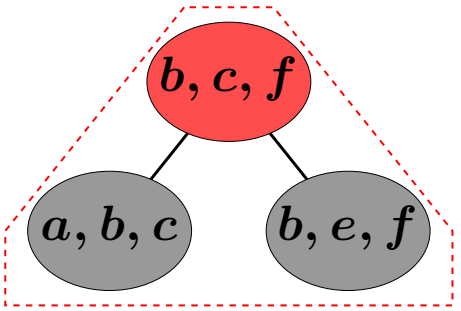
Fact: MAX WEIGHTED INDEPENDENT SET can be solved in time $O(2^w \cdot n^2)$ on a graph with n vertices and tree width at most w.

Algorithm: Dynamic programming on the tree decomposition. For each bag, solve the problem for the vertices contained in the descendants of the bag.

Subproblems at bag: Which vertices of the bag are selected in the solution I?

1. No restriction
2. $b \notin I$
3. $c \notin I$
4. $f \notin I$
5. $b, c \notin I$
6. $b, f \notin I$
7. $c, f \notin I$
8. $b, c, f \notin I$

2^w subproblems has to be solved at each node.



Bounded tree width graphs

Many problems are polynomial-time solvable for bounded tree width graphs:

- VERTEX COLORING
- EDGE COLORING
- HAMILTONIAN CYCLE
- MAXIMUM CLIQUE
- VERTEX DISJOINT PATHS

Bounded tree width graphs

Many problems are polynomial-time solvable for bounded tree width graphs:

- VERTEX COLORING
- EDGE COLORING
- HAMILTONIAN CYCLE
- MAXIMUM CLIQUE
- VERTEX DISJOINT PATHS

Usually, if a problem can be solved on trees by bottom-up dynamic programming, then the same approach works for bounded tree width graphs.

Some exceptions:

Fact: EDGE DISJOINT PATHS is NP-hard for graphs with tree width 2.

Fact: LIST EDGE COLORING is NP-hard for graphs with tree width 2.

Courcelle's Theorem

Courcelle's Theorem: If a graph property can be expressed in **Extended Monadic Second Order Logic**, then for every $w \geq 1$, there is a linear-time algorithm for testing this property in graphs having tree width w.

Extended Monadic Second Order Logic of Graphs:

- first order logic +
- adjacency and incidence relations +
- quantification over **sets of vertices** and **sets of edges.**

Courcelle's Theorem

Courcelle's Theorem: If a graph property can be expressed in **Extended Monadic Second Order Logic**, then for every $w \geq 1$, there is a linear-time algorithm for testing this property in graphs having tree width w.

Extended Monadic Second Order Logic of Graphs:

- first order logic +
- adjacency and incidence relations +
- quantification over **sets of vertices** and **sets of edges.**

Example: 3-colorability can be expressed as

$$\exists V_1, V_2, V_3 \subseteq V : (\forall v \in V : v \in V_1 \vee v \in V_2 \vee v_3 \in V_3) \wedge \forall u, v \in V$$
$$\text{adj}(u, v) \rightarrow ((u \notin V_1 \vee v \notin V_1) \wedge (u \notin V_2 \vee v \notin V_2) \wedge (u \notin V_3 \vee v \notin V_3))$$

Applications of tree width

Tree width appears in many places:

- Crucial role in the Graph Minors theory of Robertson and Seymour.
- Certain classes of graphs have small tree width (e.g., outerplanar graphs, series-parallel graphs).
- Exact/approximate algorithms for planar graphs.
- Real-life instances can have small tree width.

Constraint Satisfaction Problems (CSP)

Many interesting problems can be described as a **Constraint Satisfaction Problem.**

$$C_1(x_1, x_2, x_3) \wedge C_2(x_2, x_4) \wedge C_3(x_1, x_3, x_4)$$

A CSP instance is given by describing the

- variables,
- the domain of the variables,
- the constraints on the variables, and
- the assignments that satisfy the constraints.

Task: Find an assignment that satisfies every constraint.

Constraint Satisfaction Problems (CSP)

Many interesting problems can be described as a **Constraint Satisfaction Problem.**

$$C_1(x_1, x_2, x_3) \wedge C_2(x_2, x_4) \wedge C_3(x_1, x_3, x_4)$$

A CSP instance is given by describing the

- variables,
- the domain of the variables,
- the constraints on the variables, and
- the assignments that satisfy the constraints.

Task: Find an assignment that satisfies every constraint.

Example: 3-COLORING is a CSP problem.
Variables: vertices, Domain: $\{1, 2, 3\}$, Constraints: one for each edge.

CSP and tree width

Primal (Gaifman) graph: vertices are the variables, and two vertices are connected if they appear in a common constraint.

Fact: For every w, there is a linear-time algorithm solving CSP instances where the primal graph have tree width at most w.

CSP and tree width

Primal (Gaifman) graph: vertices are the variables, and two vertices are connected if they appear in a common constraint.

Fact: For every w, there is a linear-time algorithm solving CSP instances where the primal graph have tree width at most w.

This result is best possible.

CSP($\mathcal{G}$): the problem restricted to instances where the primal graph is in $\mathcal{G}$.

Fact:

CSP($\mathcal{G}$) is polynomial-time solvable $\iff$ $\mathcal{G}$ has bounded tree width (assuming FPT $\neq$ W[1]).

CSP and hypergraphs

Hypergraph: edges are arbitrary subsets of vertices.

Hypergraph of a CSP instance: vertices are the variables, each constraint is an edge.

CSP and hypergraphs

Hypergraph: edges are arbitrary subsets of vertices.

Hypergraph of a CSP instance: vertices are the variables, each constraint is an edge.

Considering the hypergraph instead of the primal graph makes the complexity analysis more precise.

$$I_1 = C(x_1, x_2, \ldots, x_n) \text{ vs.}$$

$$I_2 = C(x_1, x_2) \wedge C(x_1, x_3) \wedge \cdots \wedge C(x_{n-1}, x_n)$$

I_1, I_2 have the same primal graph K_n, but I_1 is always easy, I_2 can be hard.

CSP and hypergraphs

Hypergraph: edges are arbitrary subsets of vertices.

Hypergraph of a CSP instance: vertices are the variables, each constraint is an edge.

Considering the hypergraph instead of the primal graph makes the complexity analysis more precise.

$$I_1 = C(x_1, x_2, \ldots, x_n) \text{ vs.}$$

$$I_2 = C(x_1, x_2) \wedge C(x_1, x_3) \wedge \cdots \wedge C(x_{n-1}, x_n)$$

I_1, I_2 have the same primal graph K_n, but I_1 is always easy, I_2 can be hard.

Observation: If there is a constraint that covers every variable, then we have to test at most $\|I\|$ possible assignments.

Observation: If the variables can be covered by k constraints, then we have to test at most $\|I\|^k$ possible assignments.

Hypertree width

In a **hypertree decomposition** of width w, bags of vertices are arranged in a tree structure such that

1. If u and v is connected by an edge, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.
3. For each bag, there are w edges (called the **guards**) that cover the bag.

Hypertree width: width of the best decomposition.

Hypertree width

In a **hypertree decomposition** of width w, bags of vertices are arranged in a tree structure such that

1. If u and v is connected by an edge, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.
3. For each bag, there are w edges (called the **guards**) that cover the bag.

Hypertree width: width of the best decomposition.

Footnote: This is actually called generalized hypertree width for historical reasons.

Hypertree width

Fact: For every w, there is a polynomial-time algorithm for solving CSP on instances with hypergraphs having hypertree width at most w.

Algorithm: Bottom-up dynamic programming for the subtrees, similarly to the algorithm for bounded tree width.

Every bag can be covered by w edges

$\Downarrow$

There are at most $||I||^w$ possible assignments for each bag

$\Downarrow$

There are at most $||I||^w$ subproblems for each bag

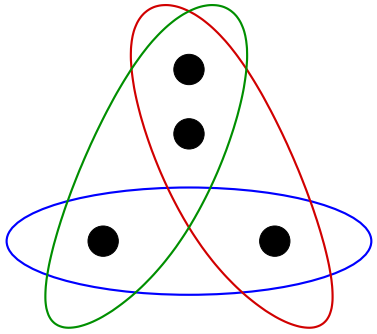
(Fractional) edge covering

An **edge cover** of a hypergraph is a subset of the edges such that every vertex is covered by at least one edge.

$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.

$\varrho^*(H)$: smallest total weight of a fractional edge cover.



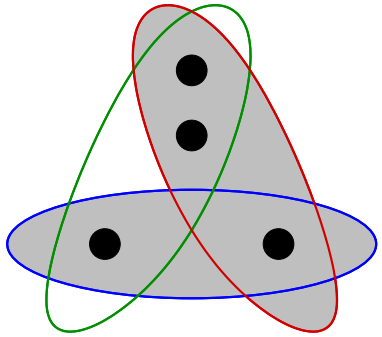
(Fractional) edge covering

An **edge cover** of a hypergraph is a subset of the edges such that every vertex is covered by at least one edge.

$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.

$\varrho^*(H)$: smallest total weight of a fractional edge cover.



$$\varrho(H) = 2$$

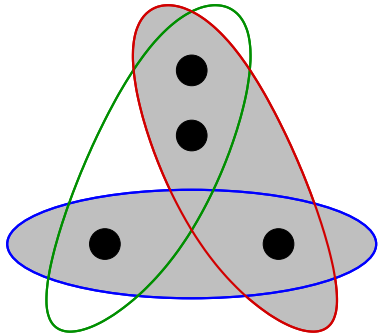
(Fractional) edge covering

An **edge cover** of a hypergraph is a subset of the edges such that every vertex is covered by at least one edge.

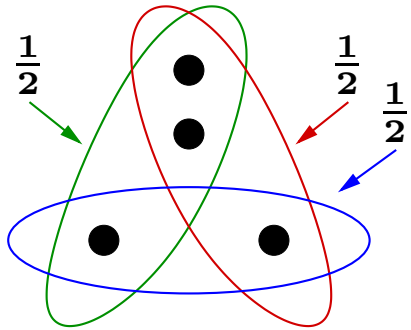
$\varrho(H)$: size of the smallest edge cover.

A **fractional edge cover** is a weight assignment to the edges such that every vertex is covered by total weight at least 1.

$\varrho^*(H)$: smallest total weight of a fractional edge cover.



$\varrho(H) = 2$



$\varrho^*(H) = 1.5$

Fractional hypertree width

In a **fractional hypertree decomposition** of width w, vertices are arranged in a tree structure such that

1. If u and v is connected by an edge, then there is a bag containing both of them.
2. For every vertex v, the bags containing v form a connected subtree.
3. A fractional edge cover of weight w is given for each bag.

Fractional hypertree width: width of the best decomposition.

Fractional hypertree width

Fact: For every w, there is a polynomial-time algorithm for solving CSP on instances with hypergraphs having fractional hypertree width at most w.

Algorithm: Similar to the algorithm for bounded hypertree width (solving the problem separately for the subtree of each bag).

Fractional hypertree width

Fact: For every w, there is a polynomial-time algorithm for solving CSP on instances with hypergraphs having fractional hypertree width at most w.

Algorithm: Similar to the algorithm for bounded hypertree width (solving the problem separately for the subtree of each bag).

Previously: If a bag can be covered by w edges, then only at most $||I||^w$ assignments of the bag has to be considered (easy)
$\Rightarrow$ There are at most $||I||^w$ subproblems for each bag.

Fractional hypertree width

Fact: For every w, there is a polynomial-time algorithm for solving CSP on instances with hypergraphs having fractional hypertree width at most w.

Algorithm: Similar to the algorithm for bounded hypertree width (solving the problem separately for the subtree of each bag).

Previously: If a bag can be covered by w edges, then only at most $||I||^w$ assignments of the bag has to be considered (easy)
$\Rightarrow$ There are at most $||I||^w$ subproblems for each bag.

Now: If a bag has a fractional edge cover of weight w, then only at most $||I||^w$ assignments of the bag has to be considered (follows from an entropy argument called Shearer's Lemma)
$\Rightarrow$ There are at most $||I||^w$ subproblems for each bag.

Finding decompositions

Tree width:

Fact: It is NP-hard to determine the tree width of a graph.

Fact: For every w, there is a linear-time algorithm that finds a tree decomposition of width w (if exists).

Finding decompositions

Tree width:

Fact: It is NP-hard to determine the tree width of a graph.

Fact: For every w, there is a linear-time algorithm that finds a tree decomposition of width w (if exists).

Hypertree width:

Complexity of determining the hypertree width is open.

Fact: For every w, there is a polynomial-time algorithm that finds a hypertree decomposition of width $3w$ if the graph has hypertree width at most w.

Finding decompositions

Tree width:

Fact: It is NP-hard to determine the tree width of a graph.

Fact: For every w, there is a linear-time algorithm that finds a tree decomposition of width w (if exists).

Hypertree width:

Complexity of determining the hypertree width is open.

Fact: For every w, there is a polynomial-time algorithm that finds a hypertree decomposition of width $3w$ if the graph has hypertree width at most w.

Fractional hypertree width:

Currently, we do not know the complexity of determining or approximating the fractional hypertree width. In the algorithms we assume that the decomposition is given on the input.

The Robber and Cops game

Game: k cops try to capture a robber in the graph.

- In each step, the cops can move from vertex to vertex arbitrarily with helicopters.
- The robber moves infinitely fast, and sees where the cops will land.

Fact:

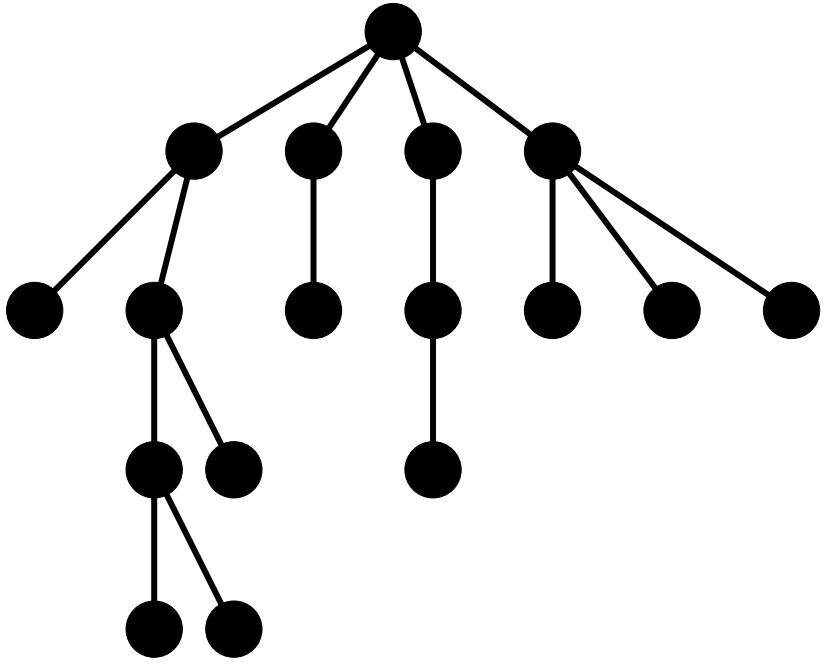
k cops can win the game $\Longleftrightarrow$ the tree width of the graph is at most $k-1$.

The winner of the game can be determined using standard techniques (there are at most n^k positions for the cops)

$\Rightarrow$ tree width can be determined.

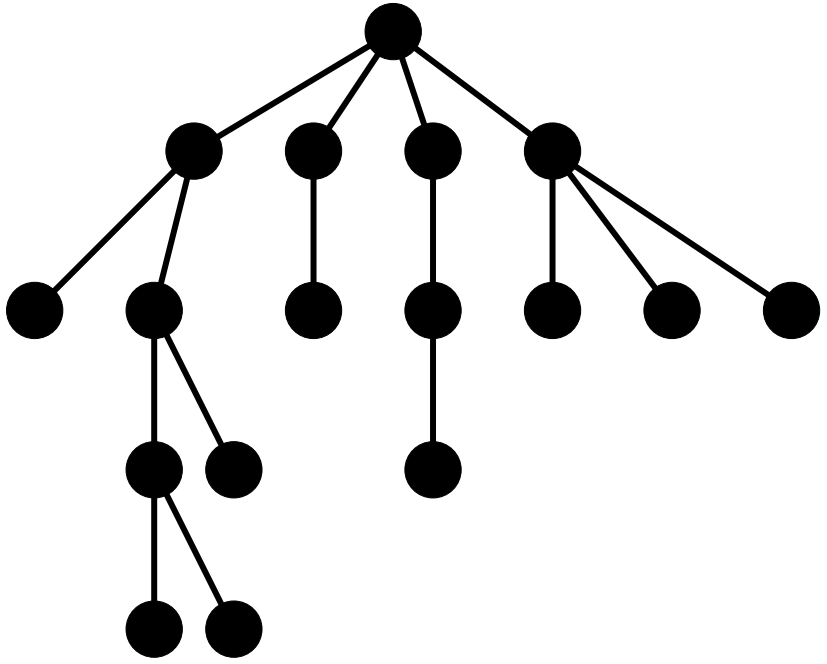
The Robber and Cops game (cont.)

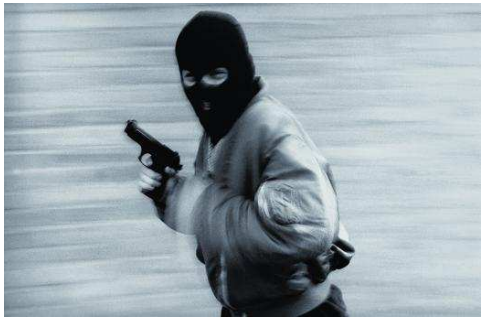
Example: 2 cops have a winning strategy in a tree.



The Robber and Cops game (cont.)

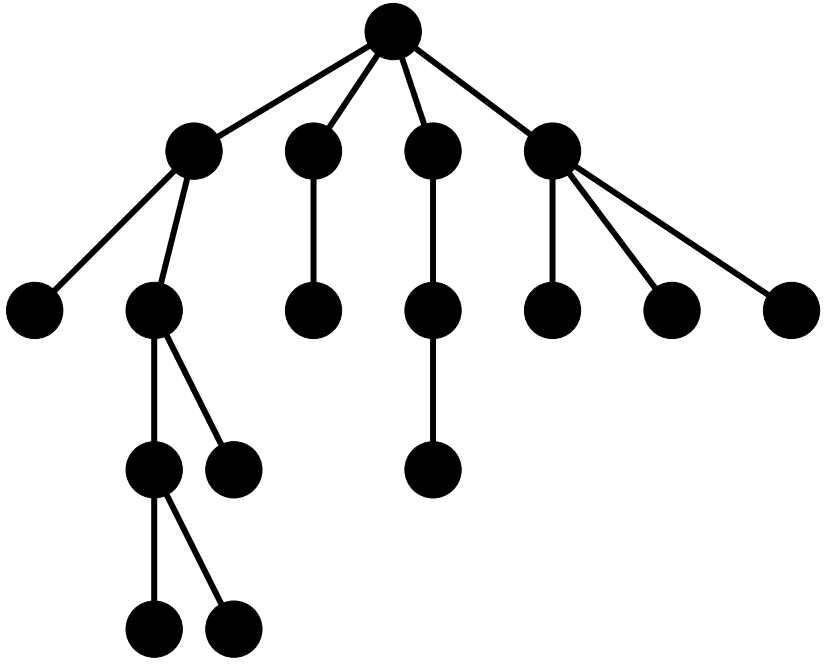
Example: 2 cops have a winning strategy in a tree.

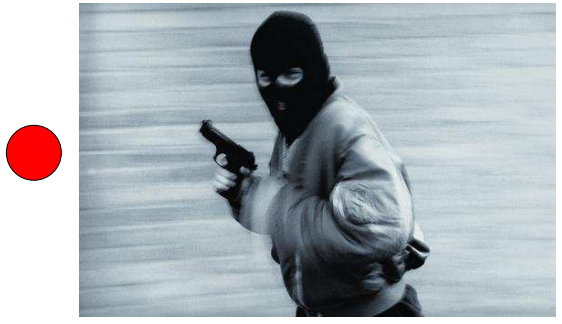




The Robber and Cops game (cont.)

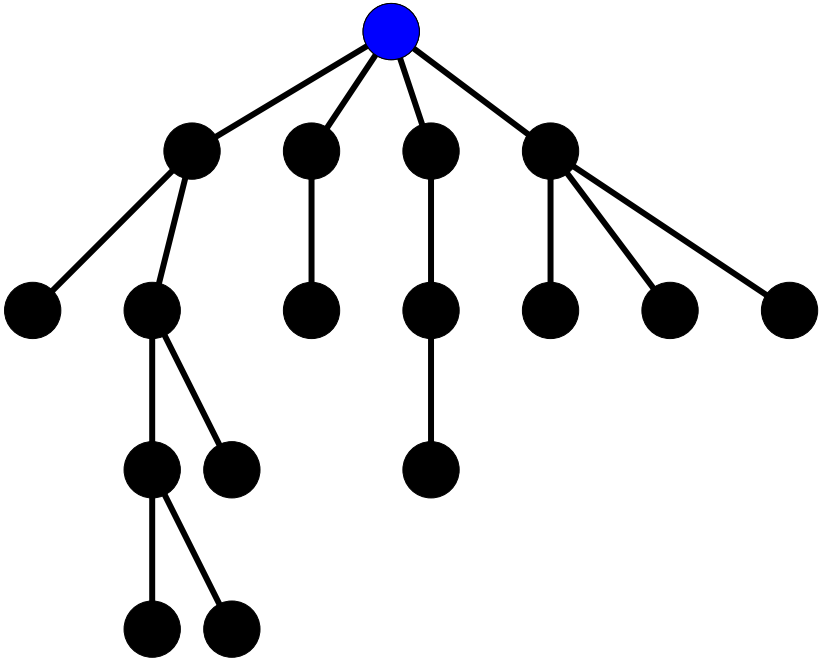
Example: 2 cops have a winning strategy in a tree.

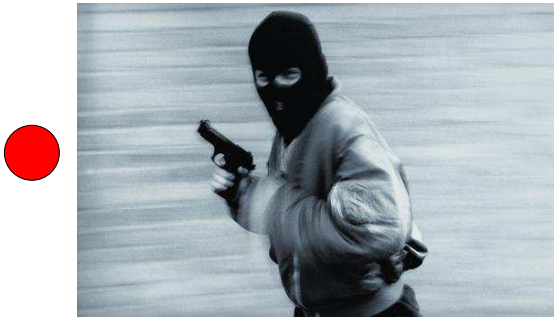




The Robber and Cops game (cont.)

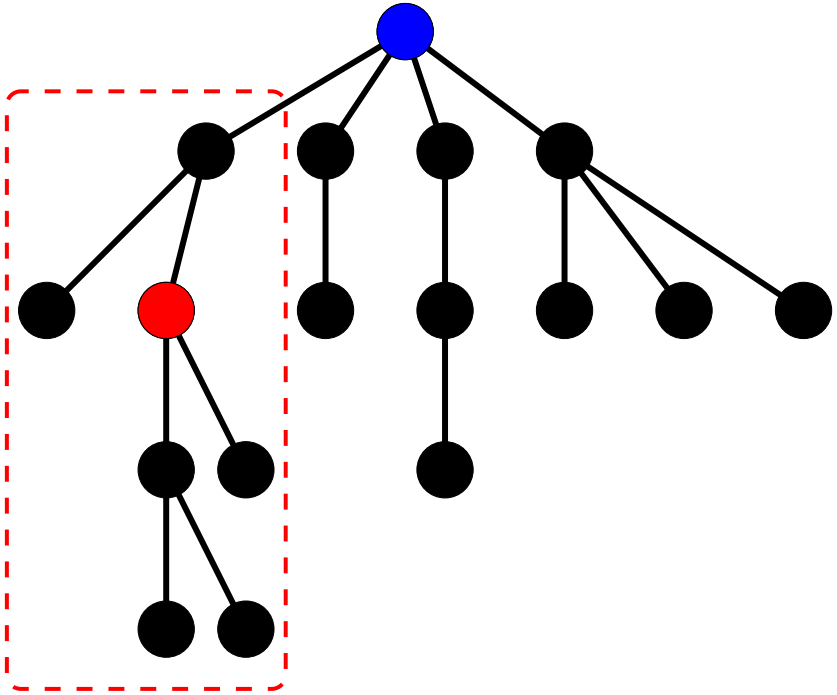
Example: 2 cops have a winning strategy in a tree.

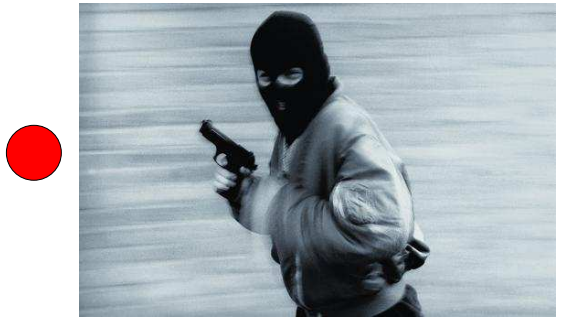




The Robber and Cops game (cont.)

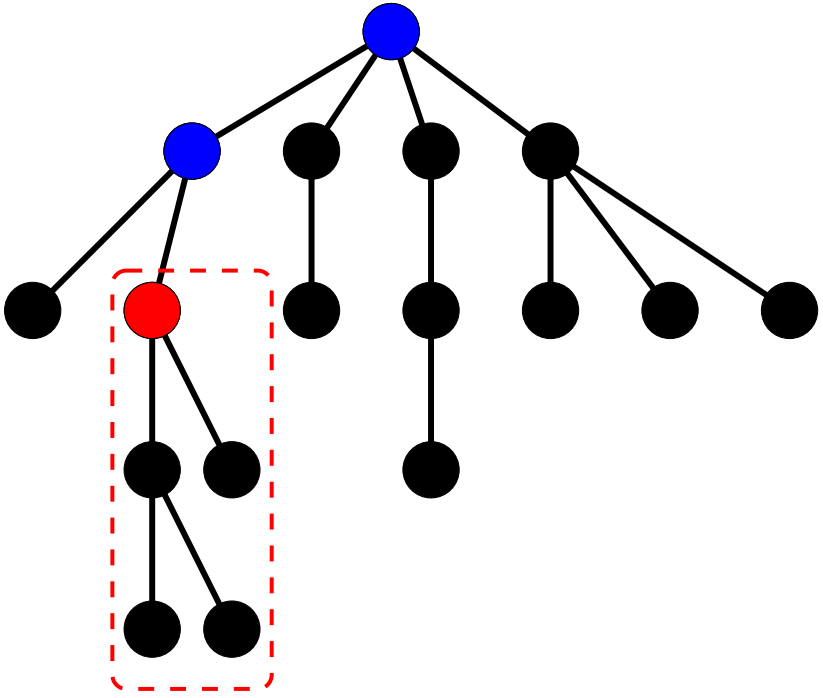
Example: 2 cops have a winning strategy in a tree.

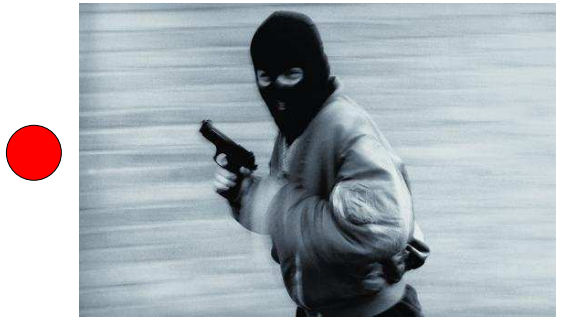




The Robber and Cops game (cont.)

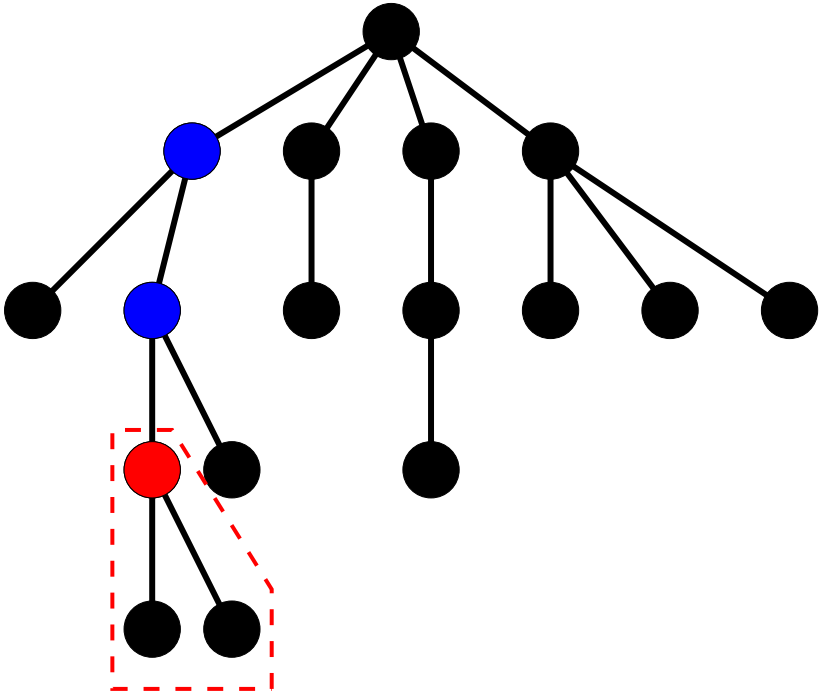
Example: 2 cops have a winning strategy in a tree.

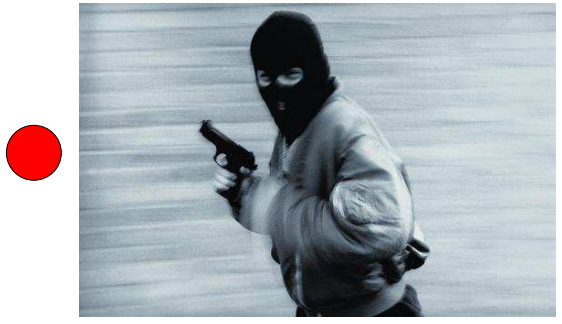




The Robber and Cops game (cont.)

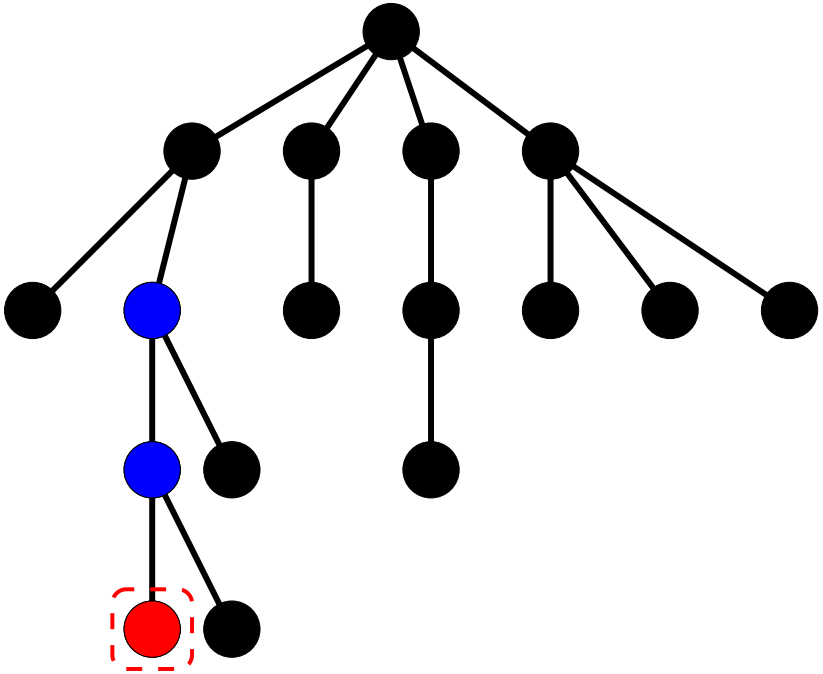
Example: 2 cops have a winning strategy in a tree.





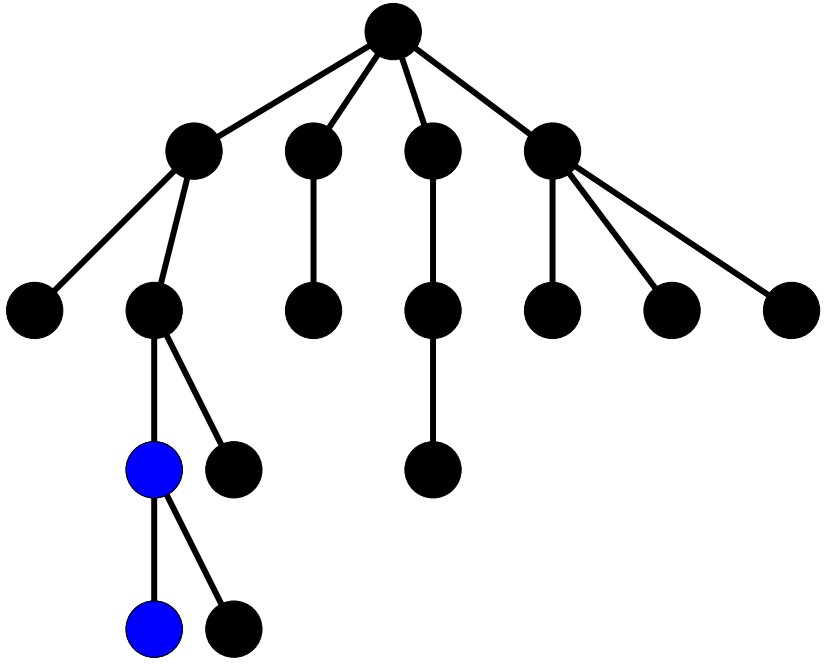
The Robber and Cops game (cont.)

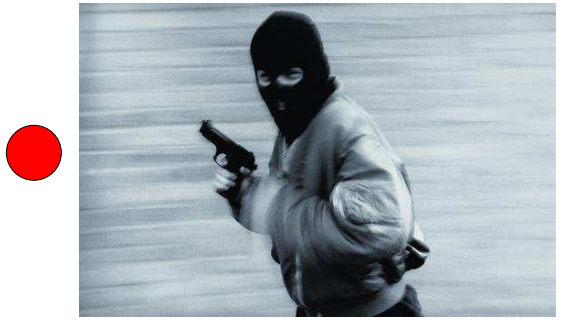
Example: 2 cops have a winning strategy in a tree.



The Robber and Cops game (cont.)

Example: 2 cops have a winning strategy in a tree.





Some more general games

Robber and Marshals:

Played on a hypergraph, a marshal can occupy an edge blocking all the vertices of the edge at the same time.

Fact: k Marshals can win the game if hypertree width is $\leq k$, and they cannot win the game if hypertree width is $\geq 3k + 1$.

$\Rightarrow$ Algorithm for approximating the hypertree width.

Some more general games

Robber and Marshals:

Played on a hypergraph, a marshal can occupy an edge blocking all the vertices of the edge at the same time.

Fact: k Marshals can win the game if hypertree width is $\leq k$, and they cannot win the game if hypertree width is $\geq 3k+1$.

$\Rightarrow$ Algorithm for approximating the hypertree width.

Robber and Army:

A general has k battalions. A battalion can be divided arbitrarily, each part can be assigned to an edge. A vertex is blocked if it is covered by one full battalion.

Fact: k battalions can win the game if fractional hypertree width is $\leq k$, and they cannot win the game if fractional hypertree width is $\geq 3k+2$.

! We don't know how to turn this result into an algorithm (there are too many army positions).

Conclusions

- trees, tree width, hypertree width, **fractional hypertree width**
- polynomial-time algorithm for CSP if the fractional hypertree width is bounded.
- No polynomial algorithm yet for finding fractional hypertree decompositions.
- Are there other classes of hypergraphs where CSP is easy? Can we prove that bounded fractional hypertree width is best possible?