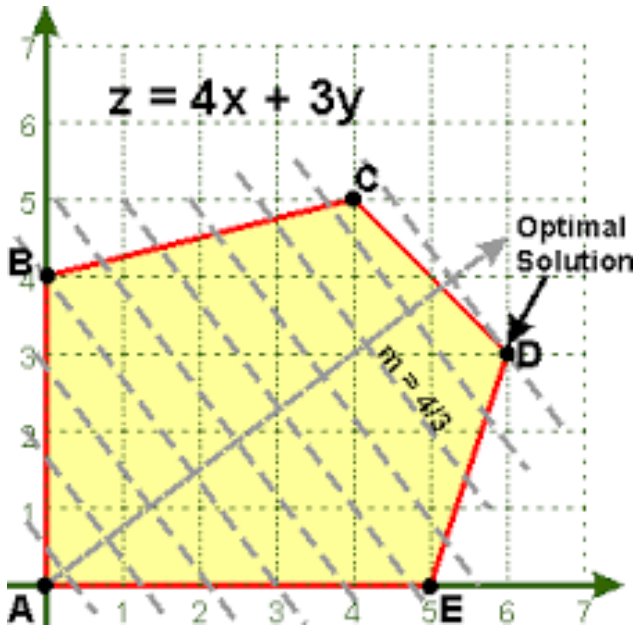
Parameterized Complexity of Integer Linear Programming (ILP)

Sebastian Ordyniak

PCSS 2017, Wien

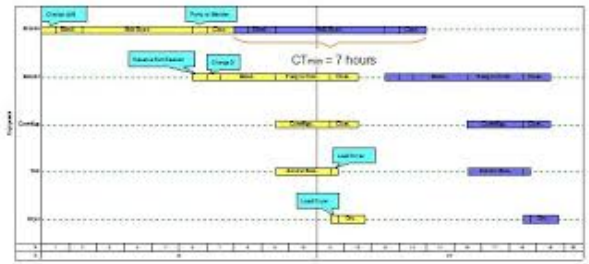
Integer Linear Programming (ILP)

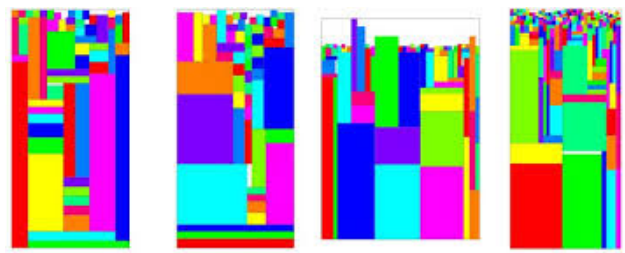
- archetypical problem for **NP**-complete optimization problems
- very general and successful paradigm for solving intractable optimization problems in practice

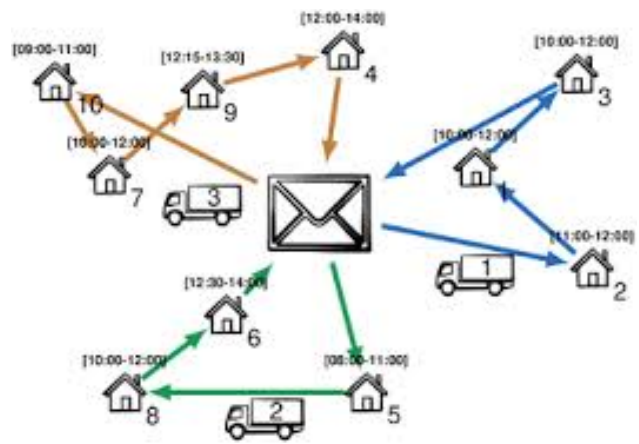


Applications

- process scheduling
- planning
- vehicle routing
- packing
- ...

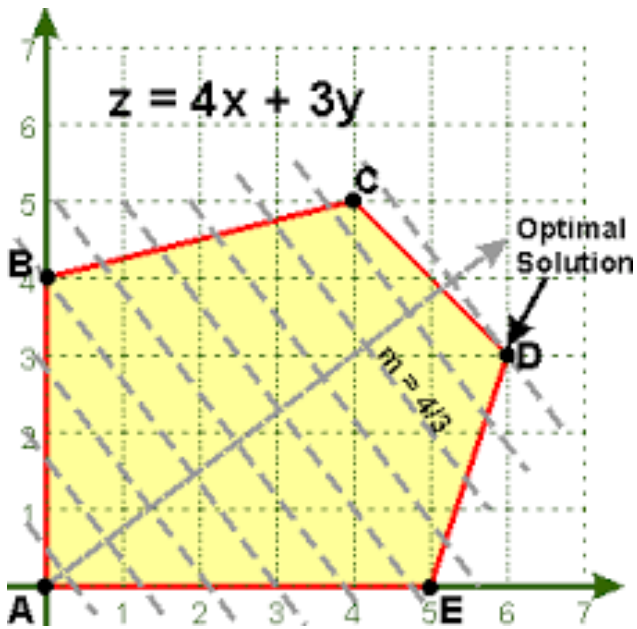






Problem Formulation: ILP

maximize	$\vec{c} \cdot \vec{x}$
subject to	$A\vec{x} \leq \vec{b}$
	$\vec{x} \in \mathbb{Z}^n$

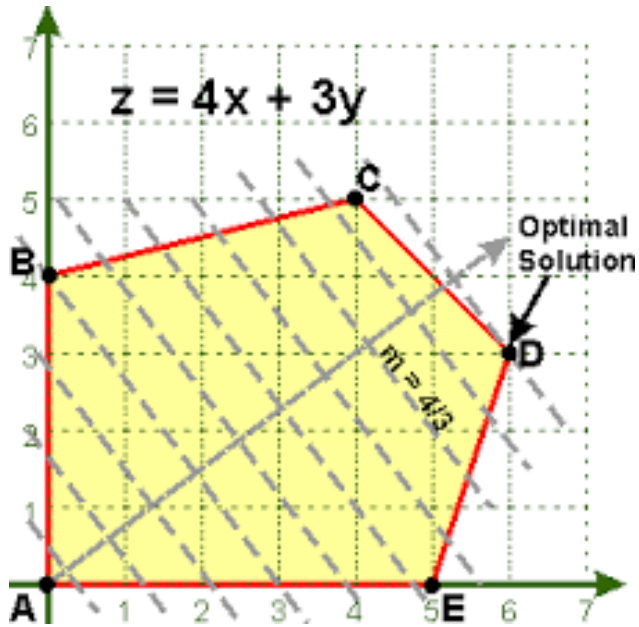


(where $A \in \mathbb{Z}^{m \times n}$, $\vec{b} \in \mathbb{Z}^m$, and $\vec{c} \in \mathbb{Z}^n$)

Problem Formulation: ILP

maximize $\vec{c} \cdot \vec{x}$
subject to $A\vec{x} \leq \vec{b}$
$\vec{x} \in \mathbb{Z}^n$

maximize $\vec{c} \cdot \vec{x}$
subject to $A\vec{x} = \vec{b}$
$\vec{l} \leq \vec{x} \leq \vec{u}; \quad \vec{x} \in \mathbb{Z}^n$



(where $A \in \mathbb{Z}^{m \times n}$, $\vec{b} \in \mathbb{Z}^m$, and $\vec{c} \in \mathbb{Z}^n$)

ILP: Example

$$\text{maximize } \sum_{1 \leq i \leq n} c_i x_i$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \leq \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{pmatrix}$$

ILP: Example

$$\text{maximize } \sum_{1 \le i \le n} c_i x_i$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \le \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{pmatrix}$$

columns $\approx$ variables

ILP: Example

$$\text{maximize } \sum_{1 \le i \le n} c_i x_i$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \le \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{pmatrix}$$

rows $\approx$ constraints

ILP: Example

$$\text{maximize } \sum_{1 \le i \le n} c_i x_i$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \le \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{pmatrix}$$

$\ell_A \approx$ the maximum coefficient in A

ILP: Example

$$\text{maximize } \sum_{1 \le i \le n} c_i x_i$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \le \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{pmatrix}$$

$\ell \approx$ the maximum coefficient in A, $\vec{b}$, and $\vec{c}$

ILP: Example

$$\text{maximize } \sum_{1 \le i \le n} c_i x_i$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \le \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{pmatrix}$$

maximization function; (maximum value $\approx$ maximum value of the maximization function for any feasible assignment)

Variants of ILP

We will distinguish the following variants of ILP:

- **ILP-feasibility**: maximization function is empty
- **Unary ILP**: the coefficients are given in unary encoding and
- **Mixed ILP**: not all variables are required to be integer
- combinations of the above

State-of-the-art

ILP and ILP-feasibility are NP-complete and only very few tractable cases are known, e.g.,:

- **totally unimodular** matrices (Papadimitriou, Steiglitz 1982),
- fixed number of variables (Lenstra 1983),
- **block matrices** (4-block N-fold) with fixed sized blocks and max coefficient (Hemmecke et al., 2010 and 2013;De Loera et al., 2013).

ILP: Structural Restrictions

- in this lecture we will focus on the complexity of ILP w.r.t. structural restrictions on the constraint matrix,
- we will represent the structure of the ILP instance in terms of its **primal graph** and **incidence graph** (very similar to the corresponding graphs for SAT and CSP)
- we then analyze the complexity of ILP w.r.t. structural parameterizations of the primal/incidence graph

Primal Graph

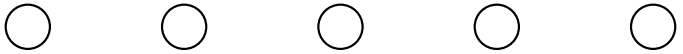
Primal Graph

The **primal graph** of an ILP instance $\mathcal{I}$, denoted by $\mathsf{P}(\mathcal{I})$, has:

- one vertex for every variable of $\mathcal{I}$,
- an edge between two variables x and y iff x and y occur together in a constraint of $\mathcal{I}$.

Primal Graph: Example

$$A := \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -5 & -2 \end{pmatrix}$$

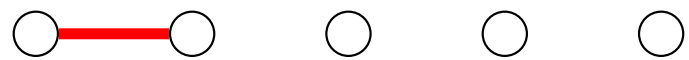


Primal Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$

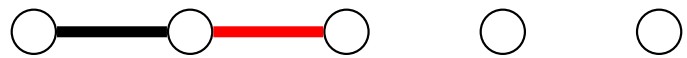
Primal Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$



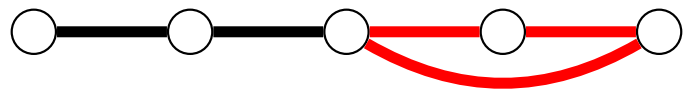
Primal Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$



Primal Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$



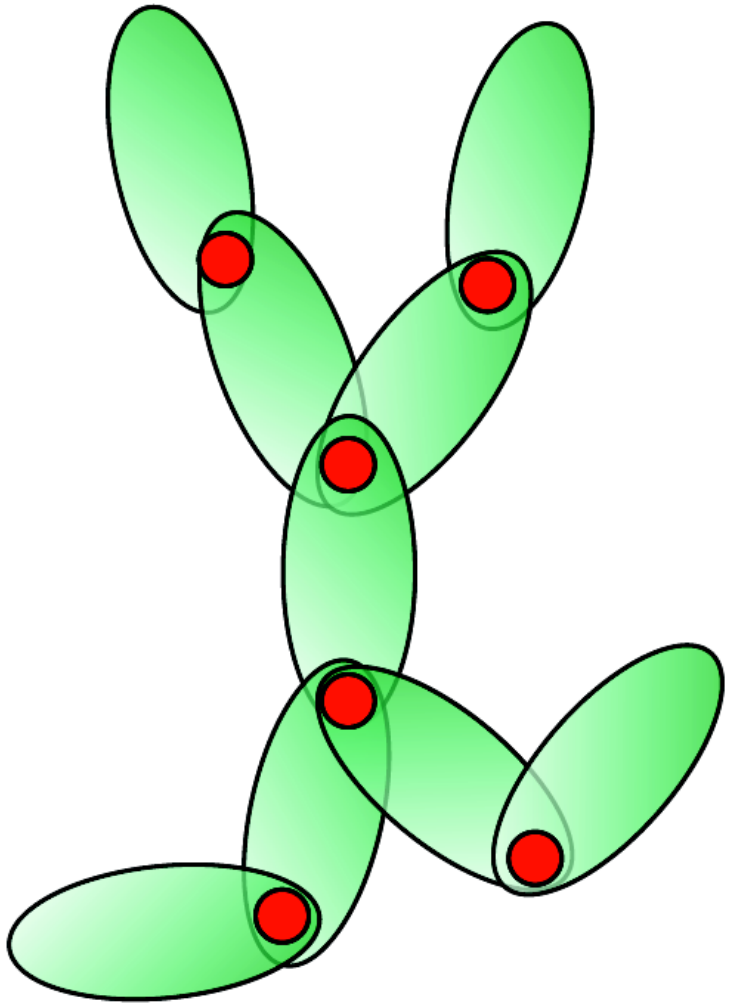
Primal Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$

Decompositional Parameters: Examples

several "out-of-the-box" decompositions available:

- **treedepth**,
- **treewidth**,
- clique-width,
- rank-width



An Algorithm using Treewidth and Domain

Theorem (Jansen and Kratsch, 2015)

ILP is fixed-parameter tractable parameterized by treewidth and the maximum absolute domain value D of any variable ($\mathcal{O}((2D+1)^{\mathsf{tw}}|\mathcal{I}|)$.

Main Idea

- dynamic programming algorithm on a tree decomposition of the primal graph,
- For each bag of the tree decomposition store which of the at most $(2D+1)^{\mathsf{tw}}$ many assignments of the variables in the bag can be extended to a feasible assignment for the subinstance represented by the current subtree.

Can we avoid the "artificial" bound on the domain?

Theorem (Ganian and O., 2016)

ILP-feasibility is NP-complete even if the primal graph has treewidth at most 3 and the maximum absolute value of any coefficient is 2.

- the theorem seems to exclude any use for treewidth without domain,
- however, the instances constructed in the reduction all share the property that they have arbitrary long paths

Treedepth

- a well-known structural parameter more restrictive than treewidth and pathwidth,
- many equivalent characterizations, e.g.:
  - **treedepth $\approx$ "length of a longest path"**,
  - treedepth $=$ cycle-rank,
  - treedepth is bounded if and only if there is a bounded width tree decomposition whose tree has bounded height

Is Treedepth sufficient on its own?

Theorem (Dvořák, Eiben, Ganian, Knop, and O., 2017)

Unary-ILP-feasibility is NP-complete even if the primal graph has treedepth at most 3.

Hence treedepth on its own is not sufficient.

What about treedepth and the maximum absolute value of any coefficient (ℓ)?

An Algorithm using Treedepth

Theorem (Ganian and O., 2016)

ILP-feasibility is fixed-parameter tractable parameterized by the treedepth of the primal graph and the maximum absolute value of any coefficient.

An Algorithm using Treedepth

Theorem (Ganian and O., 2016)

ILP-feasibility is fixed-parameter tractable parameterized by the treedepth of the primal graph and the maximum absolute value of any coefficient.

Remark

The theorem can be generalized to ILP, however, then one needs to add the number of variables in the optimization function as an additional parameter.

An Algorithm using Treedepth

Theorem (Ganian and O., 2016)

ILP-feasibility is fixed-parameter tractable parameterized by the treedepth of the primal graph and the maximum absolute value of any coefficient.

Remark

The theorem can be generalized to ILP, however, then one needs to add the number of variables in the optimization function as an additional parameter.

Open Problem

Is it possible to get rid of this additional parameter for ILP?

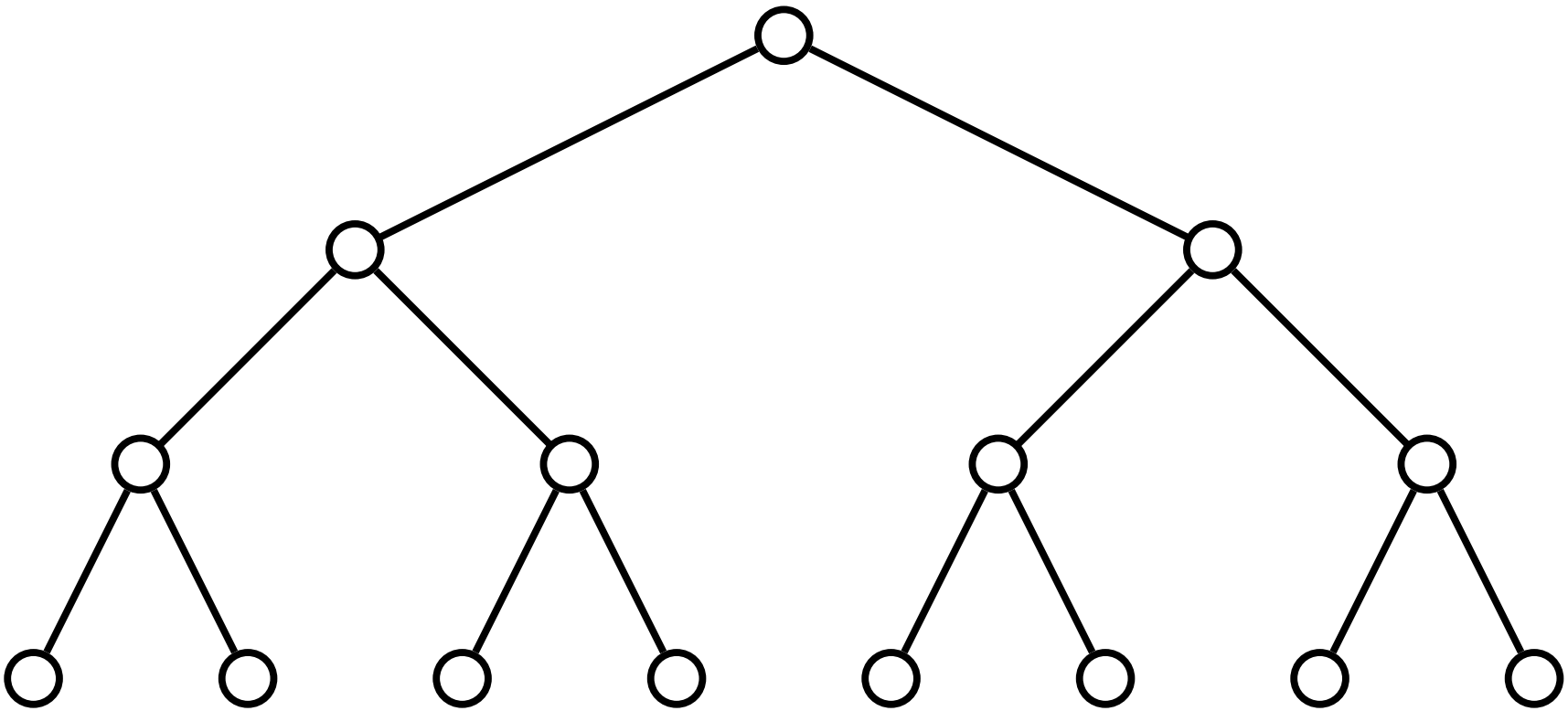
Treedepth decomposition

Definition

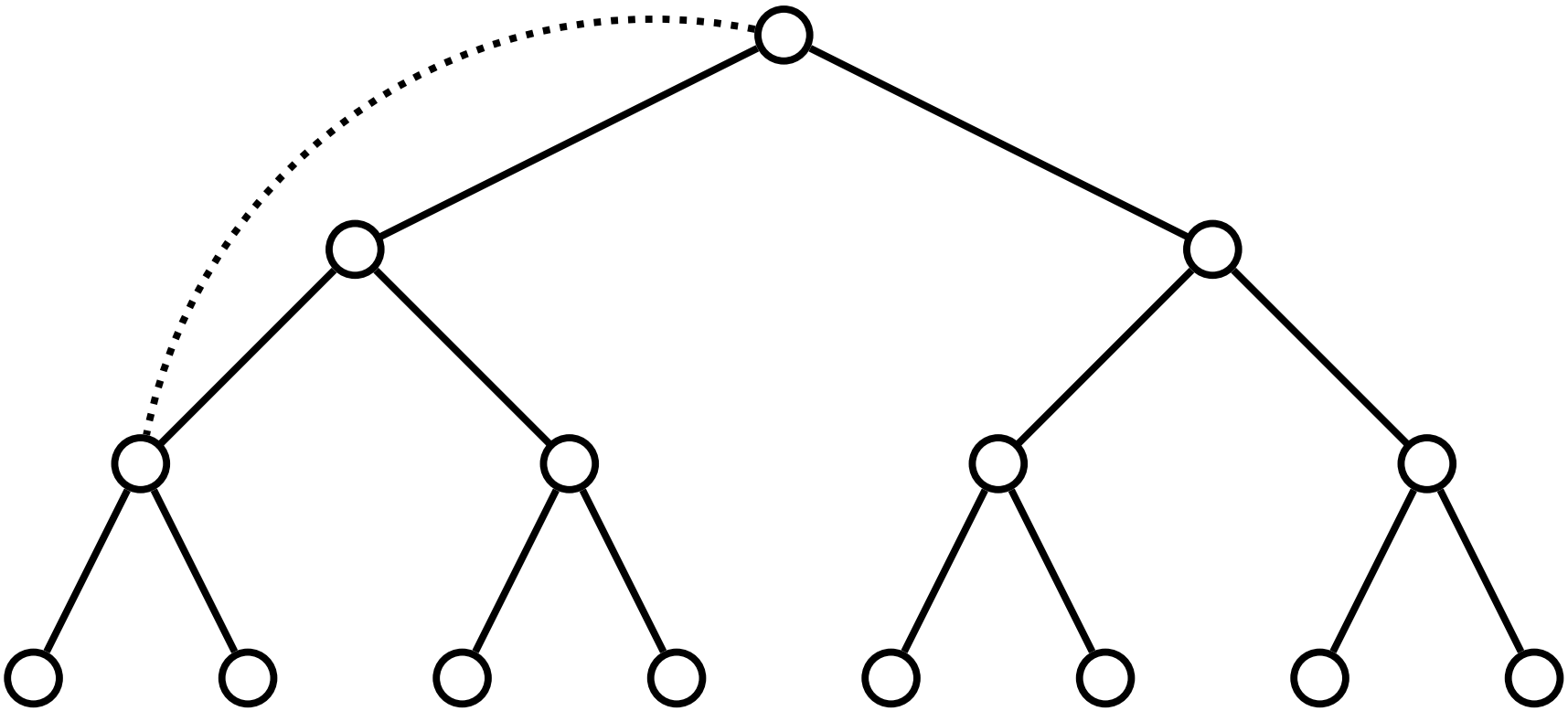
A graph G has treedepth at most k if and only if there is a rooted tree T on $V(G)$ of height at most k such that every edge in G is between ancestors and descendants of T.

The tree T is called a **treedepth decomposition** of G.

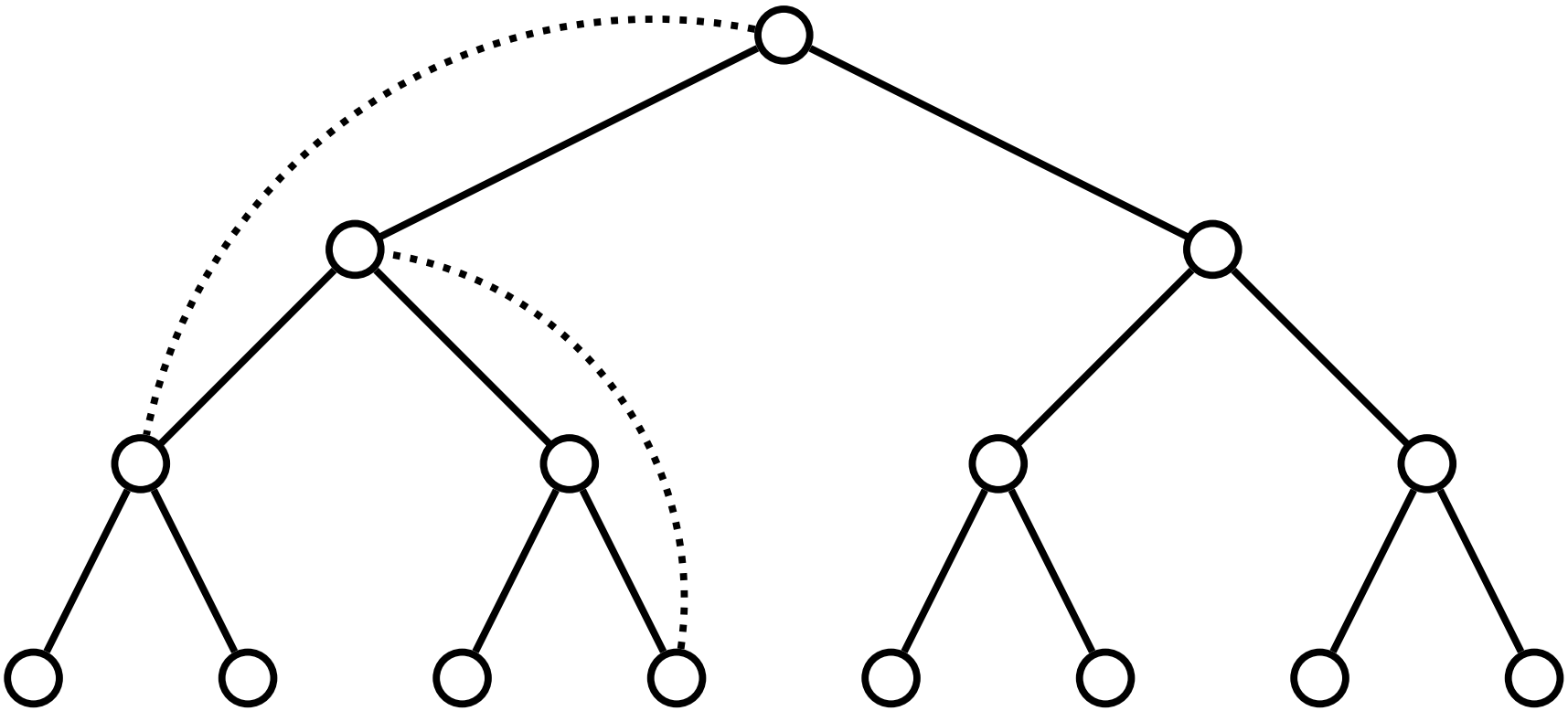
Treedepth decomposition



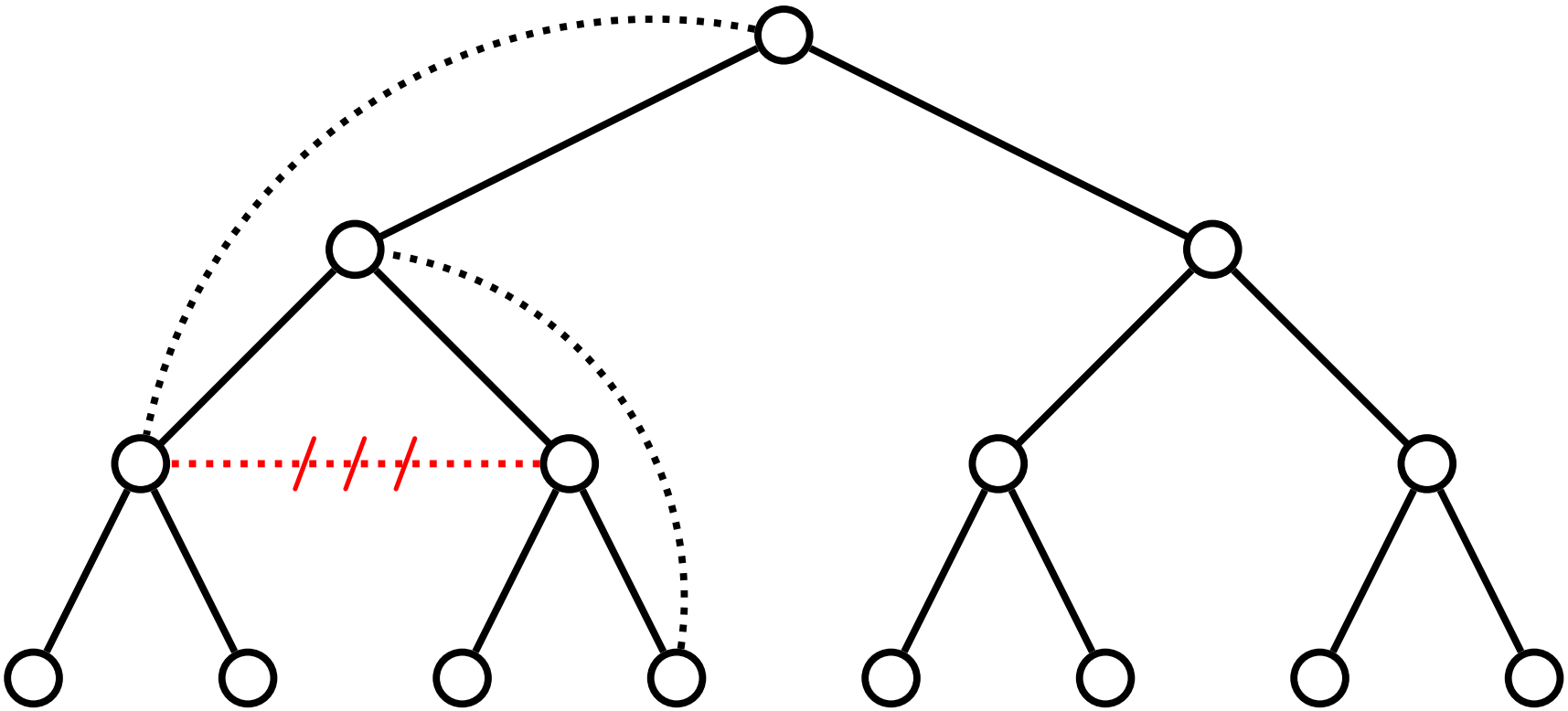
Treedepth decomposition



Treedepth decomposition



Treedepth decomposition



Main Ideas

$$\begin{aligned} 3x + 2y + \mathbf{z} &\leq 5 \\ -2y - 2\mathbf{z} &\leq -4 \\ -3x + 2\mathbf{z} &\leq 3 \end{aligned} \qquad \begin{aligned} 3x + 2y + \mathbf{z}' &\leq 5 \\ -2y - 2\mathbf{z}' &\leq -4 \\ -3x + 2\mathbf{z}' &\leq 3 \end{aligned}$$

Observation (1)

If the set of constraints in which two variables occur are equal (up to renaming one variable into the other), both variables have the same set of feasible assignments.

Main Ideas

$$\begin{aligned} 3x + 2y + \mathbf{z} &\leq 5 \\ -2y - 2\mathbf{z} &\leq -4 \\ -3x + 2\mathbf{z} &\leq 3 \end{aligned} \qquad \begin{aligned} 3x + 2y + \mathbf{z}' &\leq 5 \\ -2y - 2\mathbf{z}' &\leq -4 \\ -3x + 2\mathbf{z}' &\leq 3 \end{aligned}$$

Observation (1)

If the set of constraints in which two variables occur are equal (up to renaming one variable into the other), both variables have the same set of feasible assignments.

Hence, if (at least) one of them does not occur in the optimization function, then it can be removed from the instance.

Main Ideas

$$\begin{aligned} 3x + 2y + \mathbf{z} &\leq 5 \\ -2y - 2\mathbf{z} &\leq -4 \\ -3x + 2\mathbf{z} &\leq 3 \end{aligned} \qquad \begin{aligned} 3x + 2y + \mathbf{z}' &\leq 5 \\ -2y - 2\mathbf{z}' &\leq -4 \\ -3x + 2\mathbf{z}' &\leq 3 \end{aligned}$$

Observation (1)

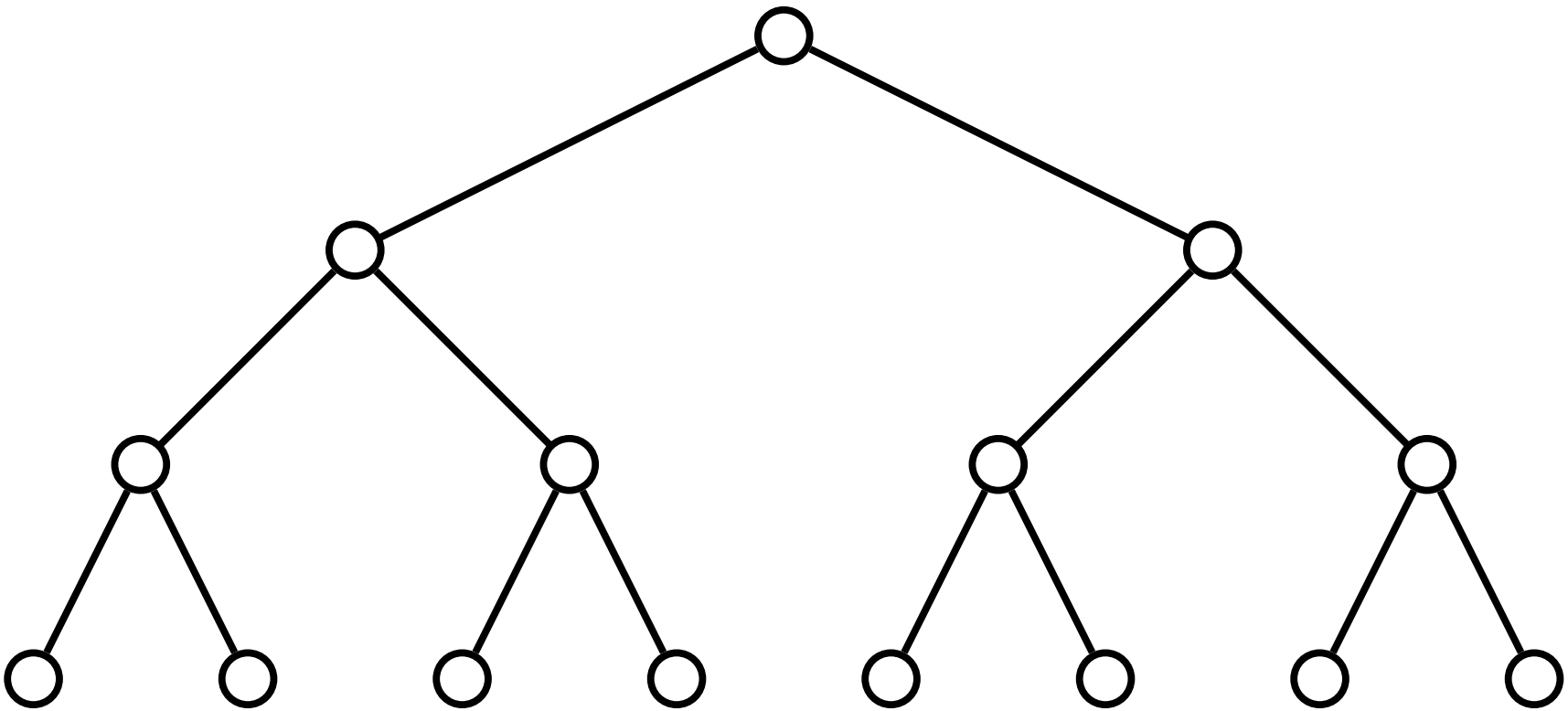
If the set of constraints in which two variables occur are equal (up to renaming one variable into the other), both variables have the same set of feasible assignments.

Hence, if (at least) one of them does not occur in the optimization function, then it can be removed from the instance.

The observation can be extended to two disjoint sets of variables instead of two variables.

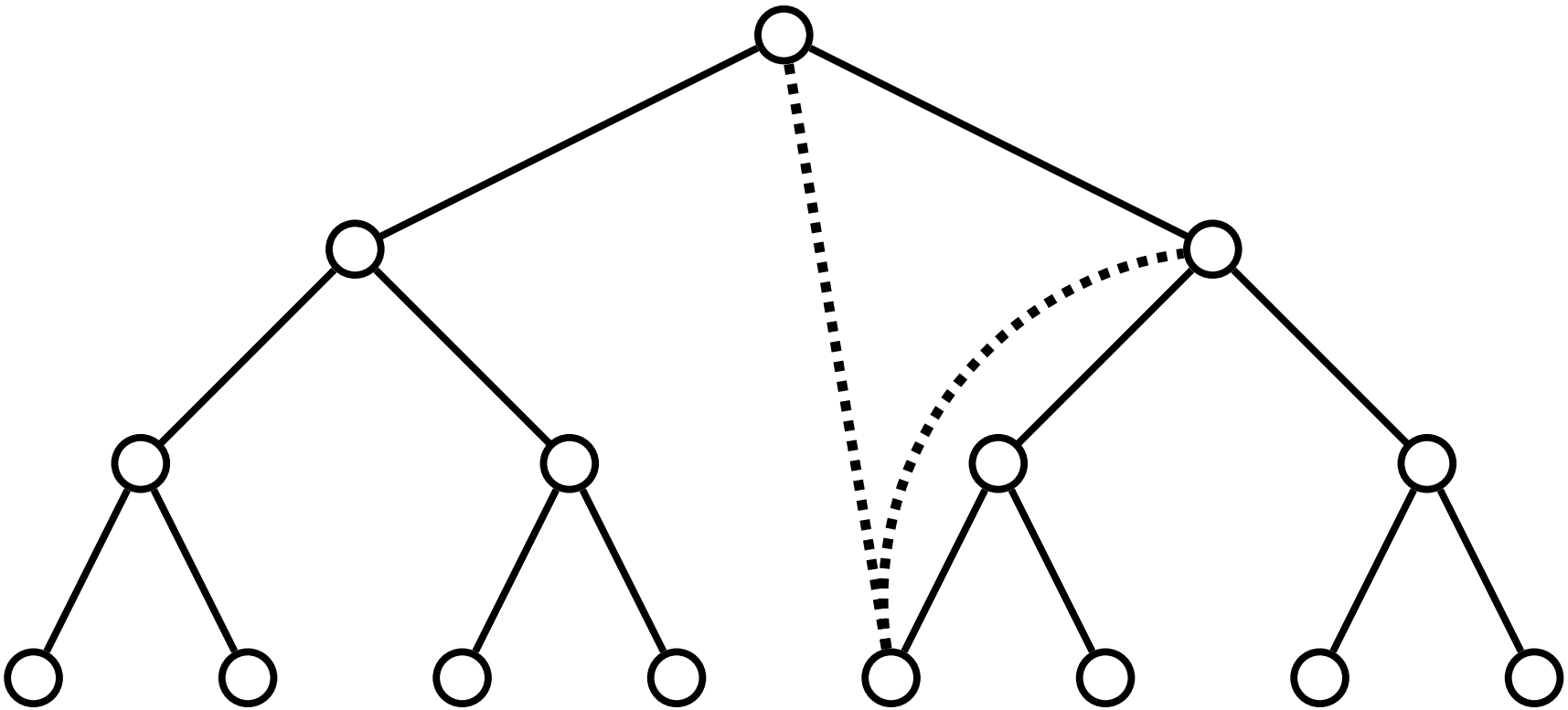
Main Idea

Apply Observation (1) to every level of the treedepth decomposition.



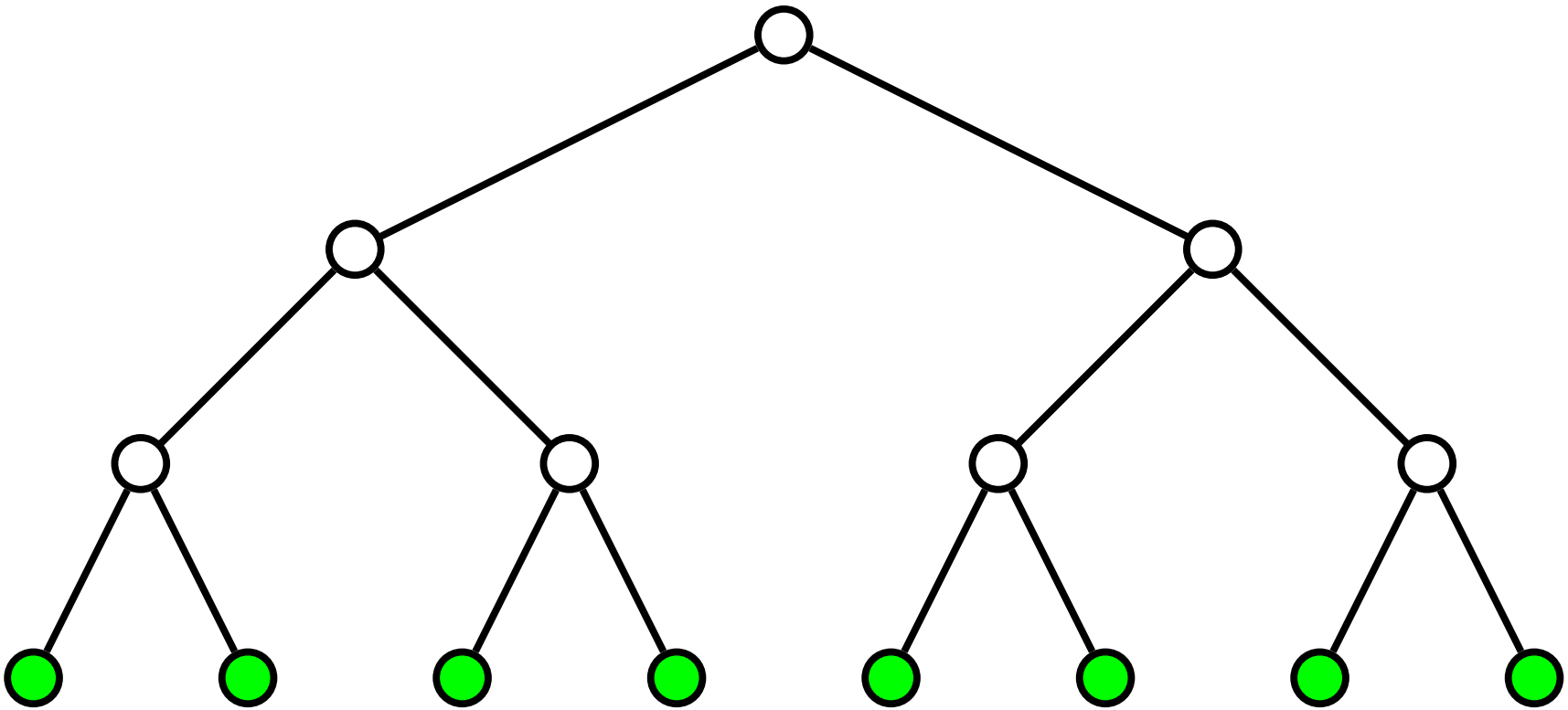
Main Idea

Apply Observation (1) to every level of the treedepth decomposition.



Main Idea

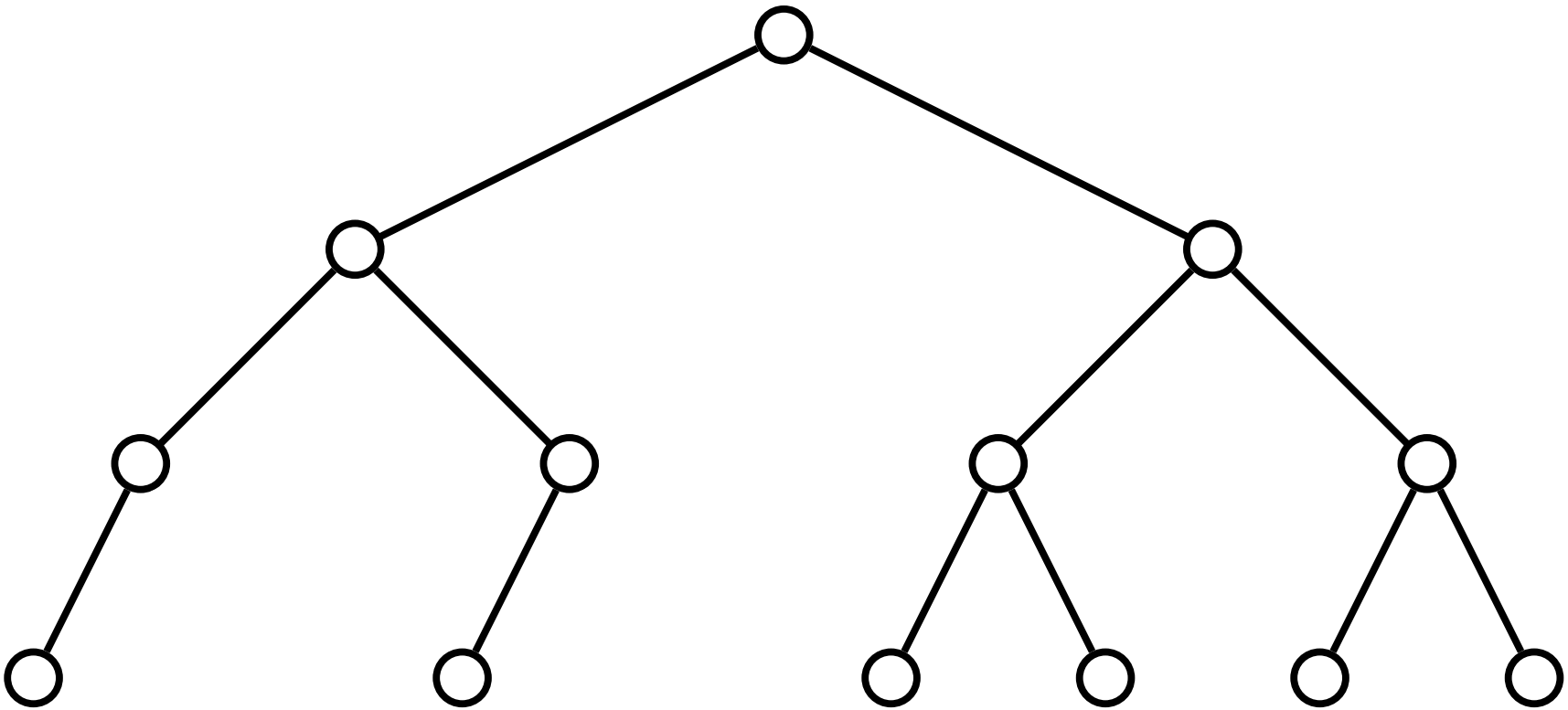
Apply Observation (1) to every level of the treedepth decomposition.



Application of Observation (1) to the bottom level (leaves).

Main Idea

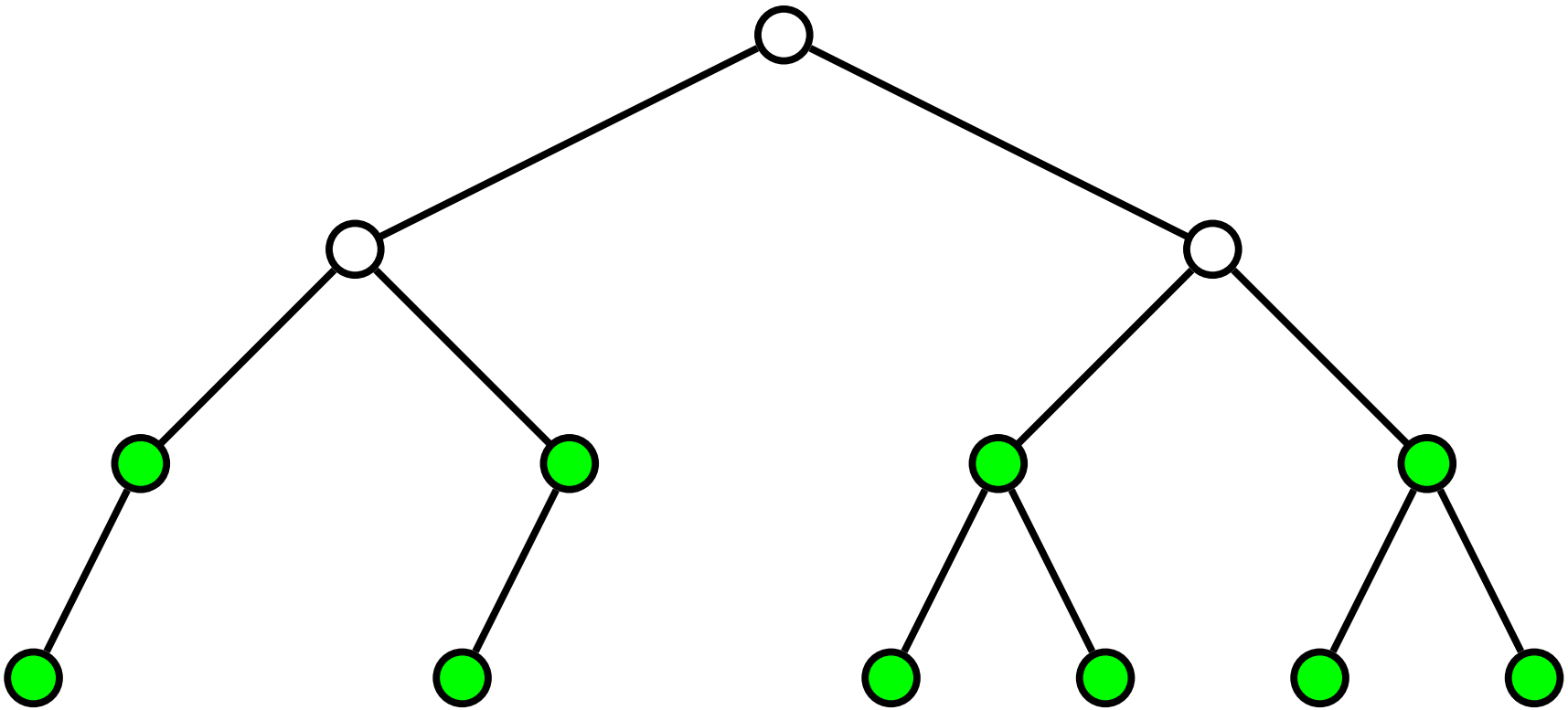
Apply Observation (1) to every level of the treedepth decomposition.



Application of Observation (1) to the bottom level (leaves).

Main Idea

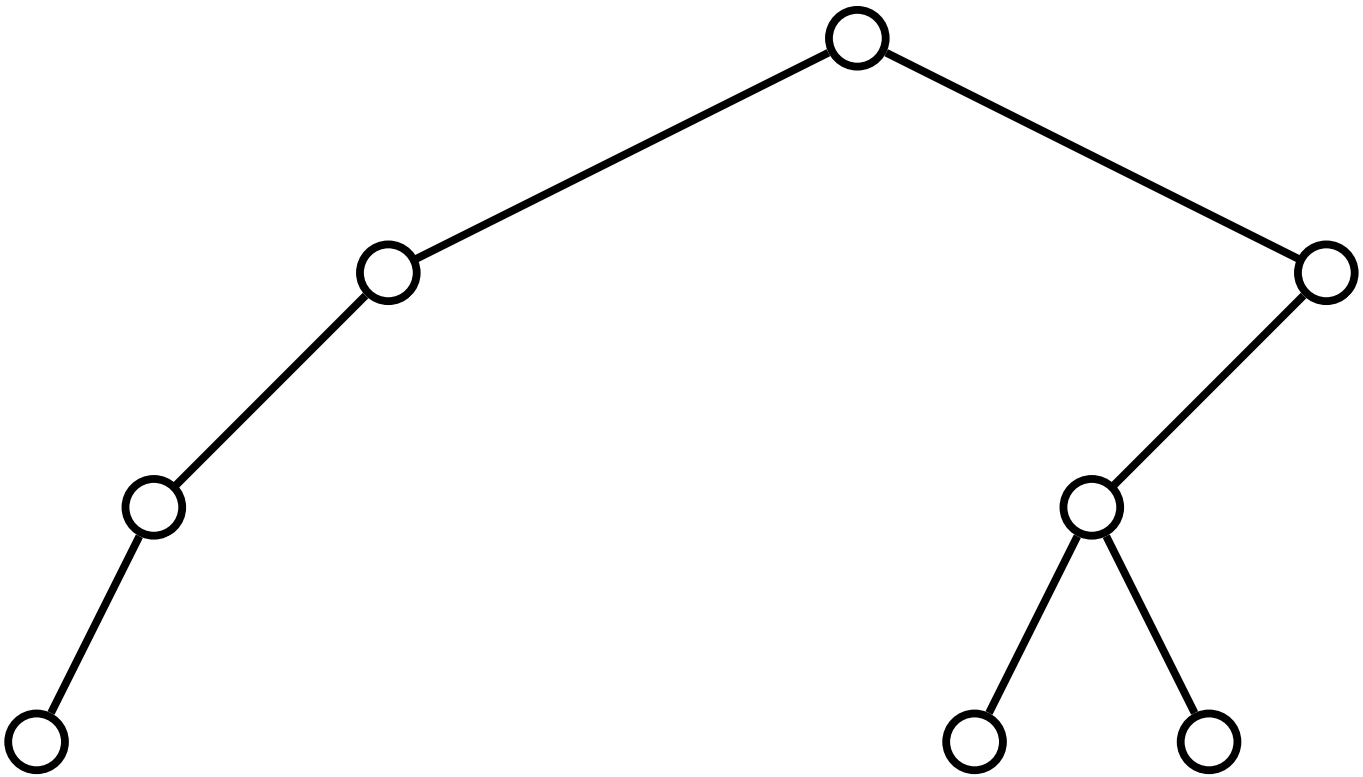
Apply Observation (1) to every level of the treedepth decomposition.



Application of Observation (1) to the level above the bottom level.

Main Idea

Apply Observation (1) to every level of the treedepth decomposition.



Application of Observation (1) to the level above the bottom level.

Main Ideas

One can show that the number of variables that remain after this preprocessing procedure is bounded in terms of the treedepth of the primal graph and the maximum value of any coefficient.

Hence, we can solve the reduced ILP instance by applying the following result.

Theorem (Lenstra 1983)

ILP parameterized by the number of variables is fixed-parameter tractable.

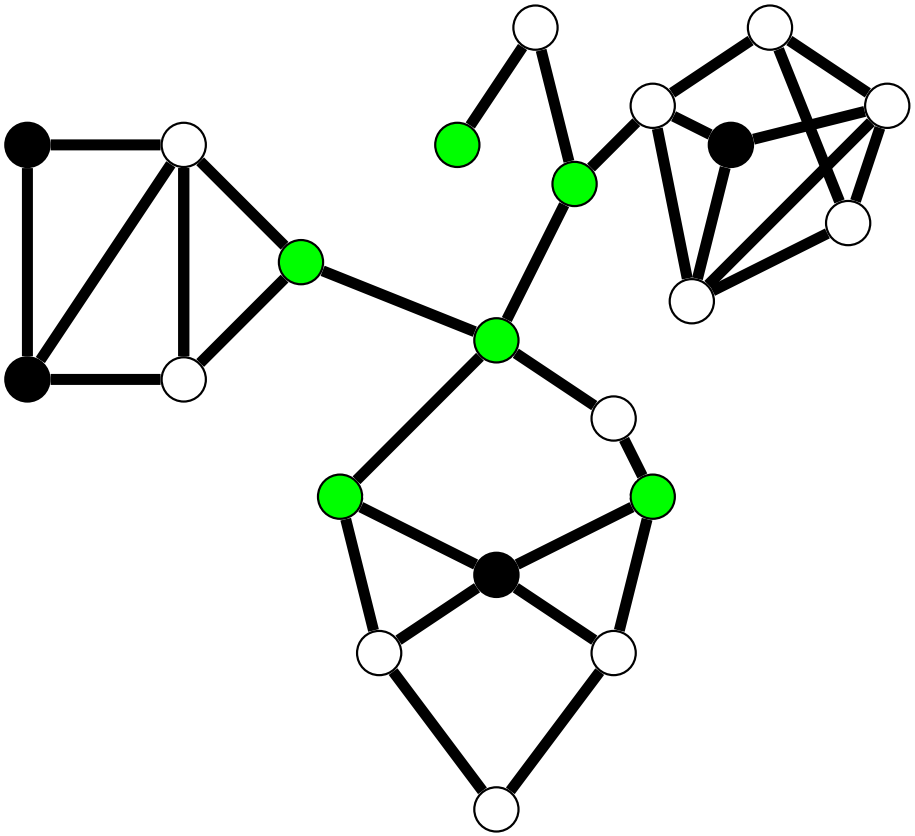
Torso-width

Width parameter specifically designed for MILP and ILP with a mixture of bounded and unbounded domain variables.

Main Algorithmic Idea

- use the treewidth algorithm on the part of the instance that has bounded domain,
- use Lenstra's algorithm for the remaining part.

Torso-width by Example

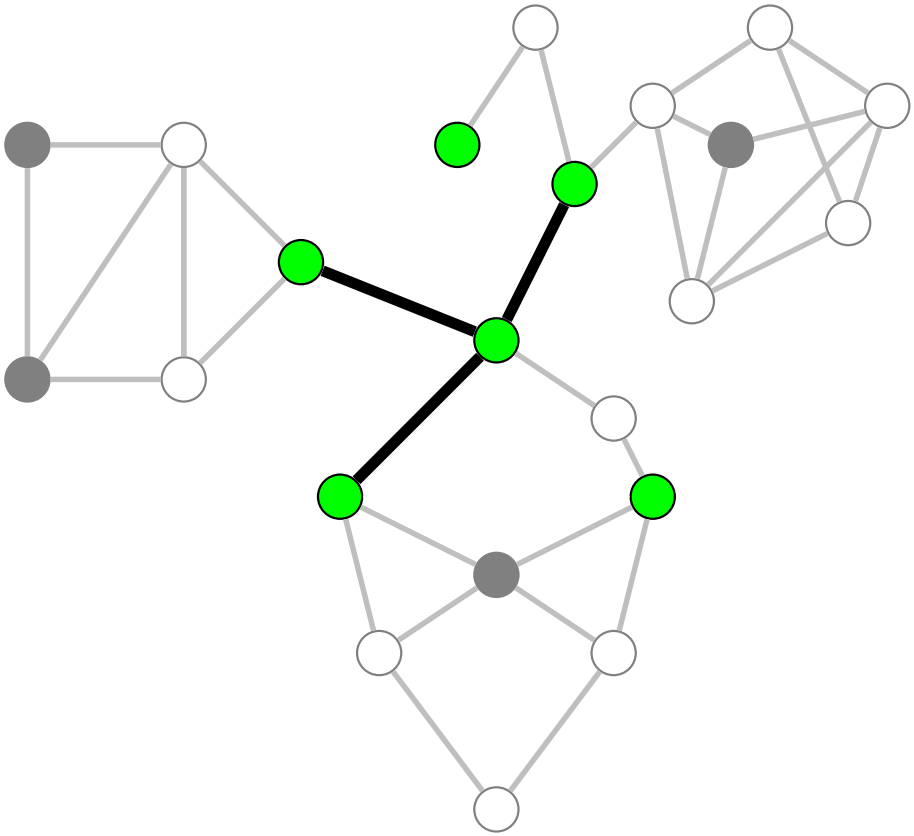


Primal graph of MILP instance:

- **white**: non-integer variables
- **black**: integer variables
- **green**: bounded domain integer variables

- Parts with few integer variables can be handled efficiently (by Lenstra's algorithm)
- Collapse them to obtain a **torso** with bounded domains (Treewidth-based dynamic programming)

Torso-width by Example

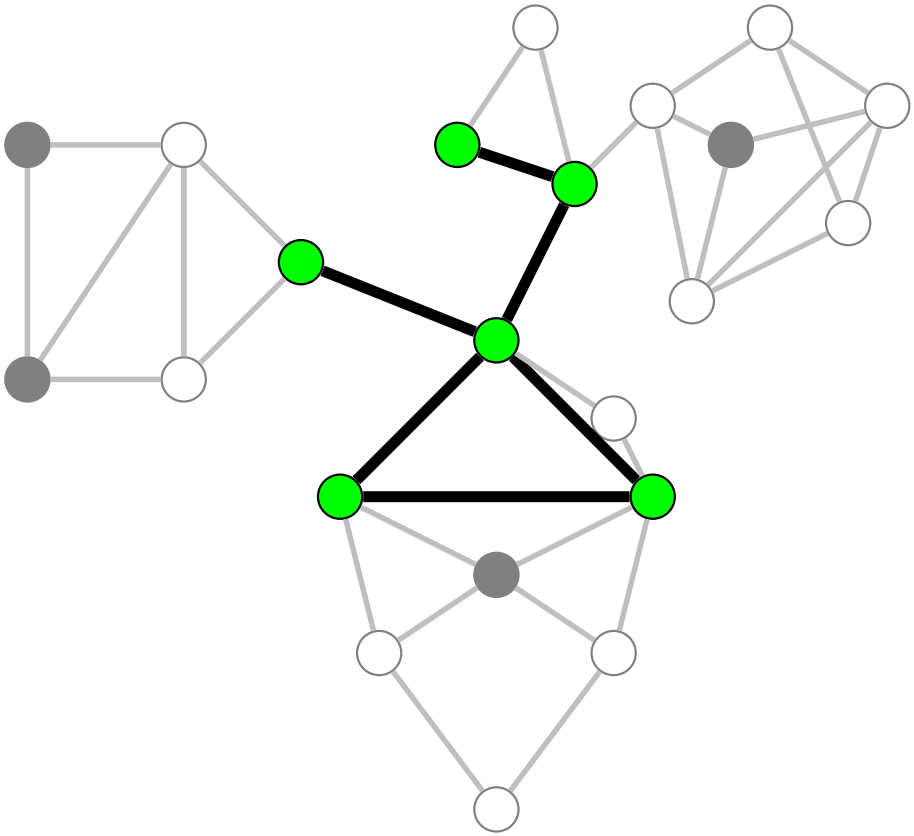


Primal graph of MILP instance:

- **white**: non-integer variables
- **black**: integer variables
- **green**: bounded domain integer variables

- Parts with few integer variables can be handled efficiently (by Lenstra's algorithm)
- Collapse them to obtain a **torso** with bounded domains (Treewidth-based dynamic programming)

Torso-width by Example

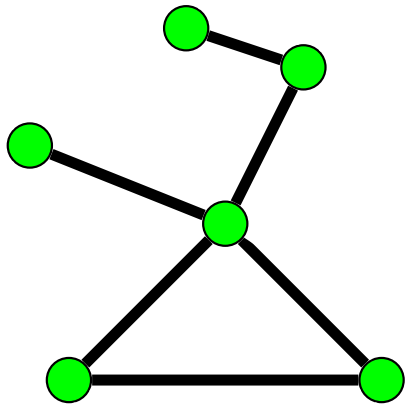


Primal graph of MILP instance:

- **white**: non-integer variables
- **black**: integer variables
- **green**: bounded domain integer variables

- Parts with few integer variables can be handled efficiently (by Lenstra's algorithm)
- Collapse them to obtain a **torso** with bounded domains (Treewidth-based dynamic programming)

Torso-width by Example



Primal graph of MILP instance:

- **white**: non-integer variables
- **black**: integer variables
- **green**: bounded domain integer variables

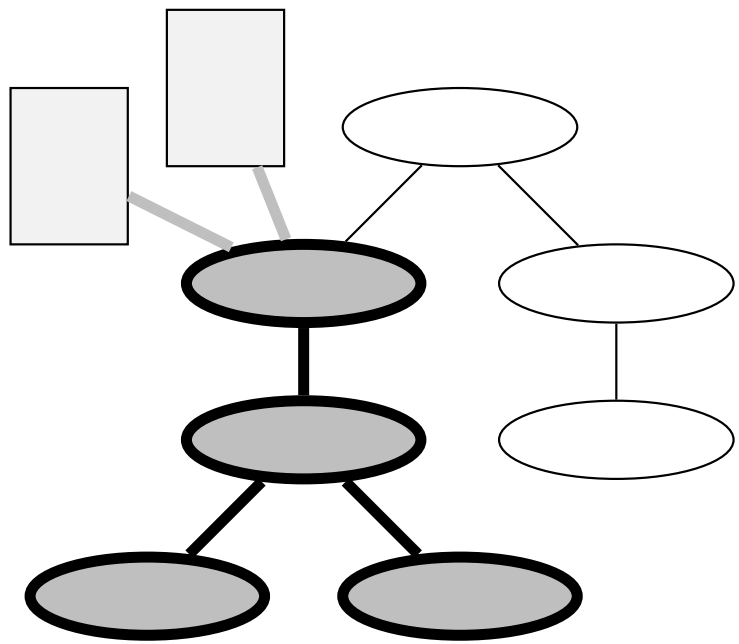
- Parts with few integer variables can be handled efficiently (by Lenstra's algorithm)
- Collapse them to obtain a **torso** with bounded domains (Treewidth-based dynamic programming)

Torso-width $\approx$ max. of the treewidth of optimal torso and the max. number of integer variables in any outside component

Algorithm for Torso-width

Solve the ILP instance by:

- use the DP-algorithm for treewidth and domain on the torso and
- verify at each node that all assignments in the records are also feasible for the attached components (using Lenstra)



Torso-width Result

Theorem (Ganian, O., and Ramanujan, 2017)

MILP is FPT parameterized by torso-width.

When to use torso-width instead of treewidth?

- MILP
- ILP if some (not all) variables have bounded domain

Summary: Primal Graph

Using structural restrictions of the primal graph leads to three main tractable cases for ILP:

- treewidth and domain,
- treedepth, coefficients, and number of variables in the maximization function,
- torso-width.

Summary: Primal Graph

Using structural restrictions of the primal graph leads to three main tractable cases for ILP:

- treewidth and domain,
- treedepth, coefficients, and number of variables in the maximization function,
- torso-width.

All other combinations can be shown to be **para-NP**-hard.

Summary: Primal Graph

Using structural restrictions of the primal graph leads to three main tractable cases for ILP:

- treewidth and domain,
- treedepth, coefficients, and number of variables in the maximization function,
- torso-width.

All other combinations can be shown to be **para-NP**-hard.

Open Problem

Is it possible to get rid of the parameter "number of variables in the maximization function" for the treedepth-based algorithm?

Incidence Graph

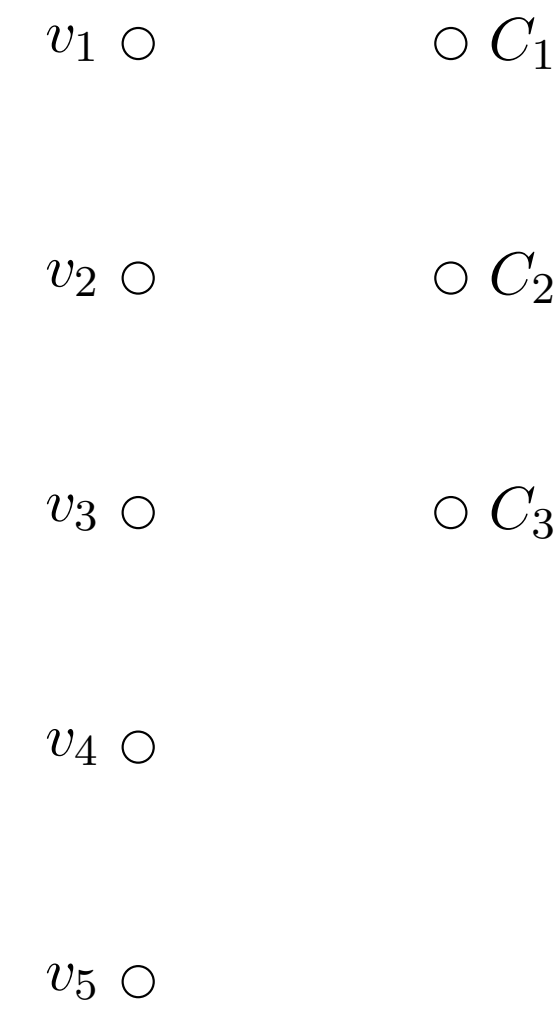
Incidence Graph

The **incidence graph** of an ILP instance $\mathcal{I}$, denoted by $\mathsf{I}(\mathcal{I})$, has:

- one vertex for every variable of $\mathcal{I}$,
- one vertex for every constraint of $\mathcal{I}$, and
- an edge between a variable x and a constraint c iff x occurs (has a non-zero coefficient) in c.

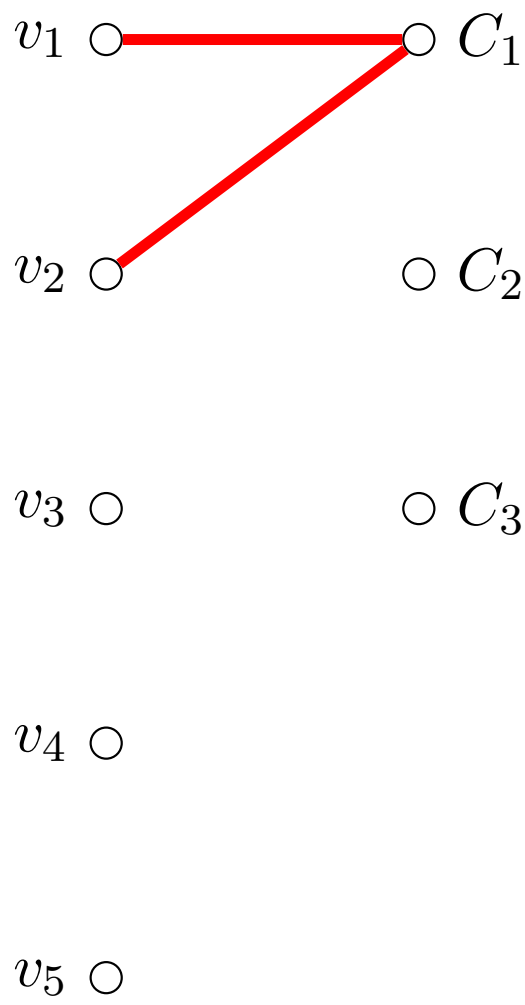
Incidence Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$



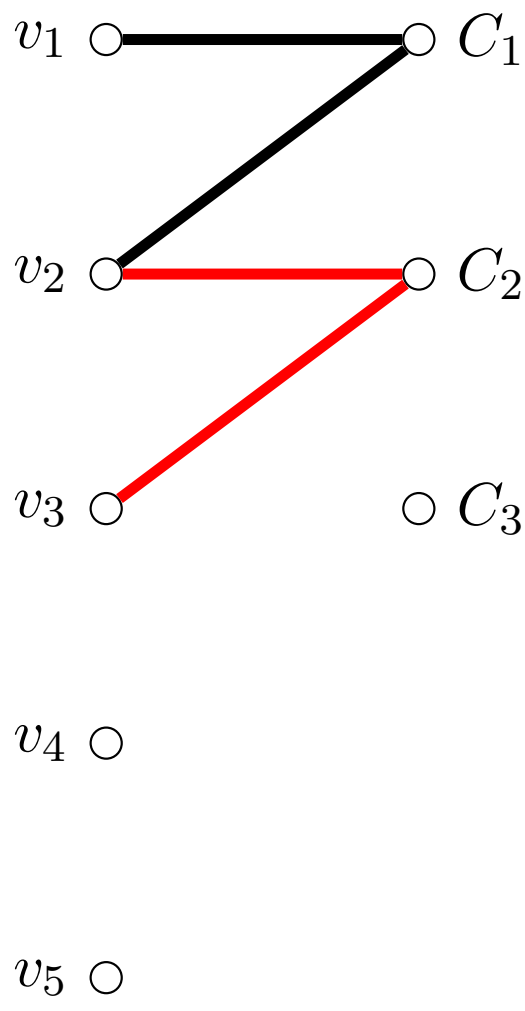
Incidence Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$



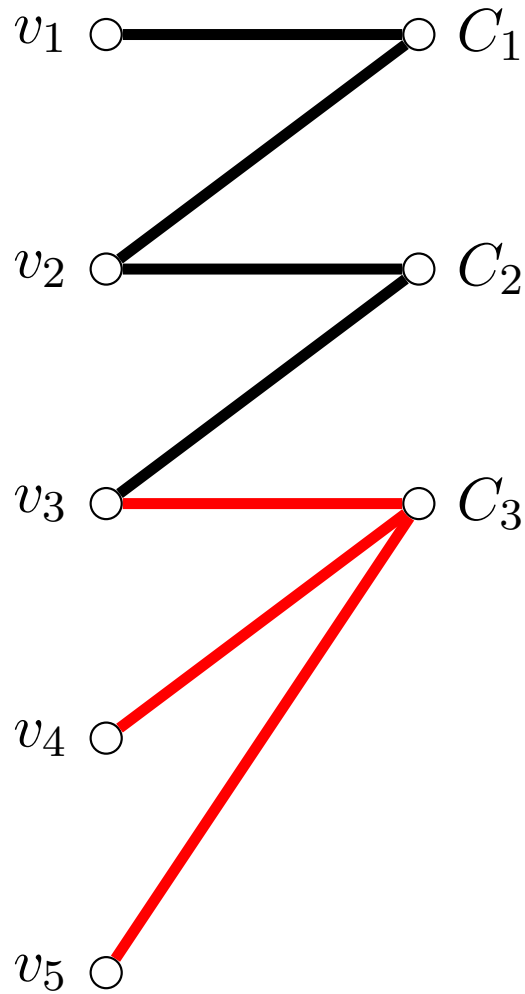
Incidence Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$



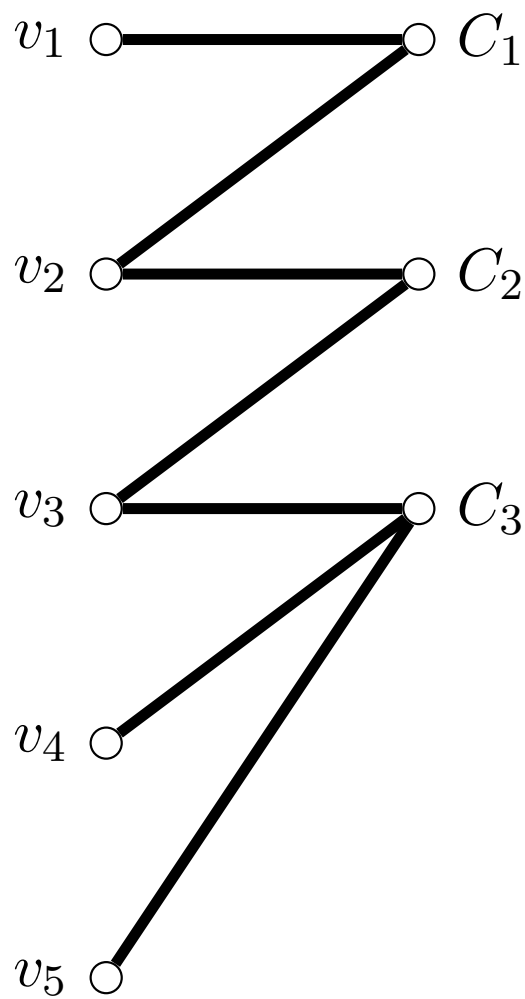
Incidence Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$



Incidence Graph: Example

$$A := \begin{pmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & * \end{pmatrix}$$



Incidence Graph vs. Primal Graph

- the incidence graph contains more information about the ILP instance,
- the treewidth of the incidence graph is always at most the treewidth of the primal graph plus one; but it can be arbitrary smaller,
- the main difference between incidence and primal treewidth is that incidence treewidth can be small even for instances with large arity

Incidence Graph vs. Primal Graph

Theorem

ILP-feasibility is NP-complete even if the incidence treewidth is one and all variables have binary domain.

Incidence Graph vs. Primal Graph

Theorem

ILP-feasibility is NP-complete even if the incidence treewidth is one and all variables have binary domain.

The main idea is a reduction from Subset Sum:

Subset Sum

Input: A set $S = \{s_1, \ldots, s_n\}$ of natural numbers and a target number t.

Question: Is there a subset S' of S such that $\sum_{s \in S'} s = t$?

The Reduction

Given an instance $\mathcal{S} = (\{s_1, \ldots, s_n\}, t)$ of Subset Sum, then the ILP instance $\mathcal{I}$ with n binary variables $x_1, \ldots, x_n$ and the constraint:

$$\sum_{1 \leq i \leq n} s_i x_i = t$$

is equivalent to $\mathcal{S}$.

The Reduction

Given an instance $\mathcal{S} = (\{s_1, \ldots, s_n\}, t)$ of Subset Sum, then the ILP instance $\mathcal{I}$ with n binary variables $x_1, \ldots, x_n$ and the constraint:

$$\sum_{1 \leq i \leq n} s_i x_i = t$$

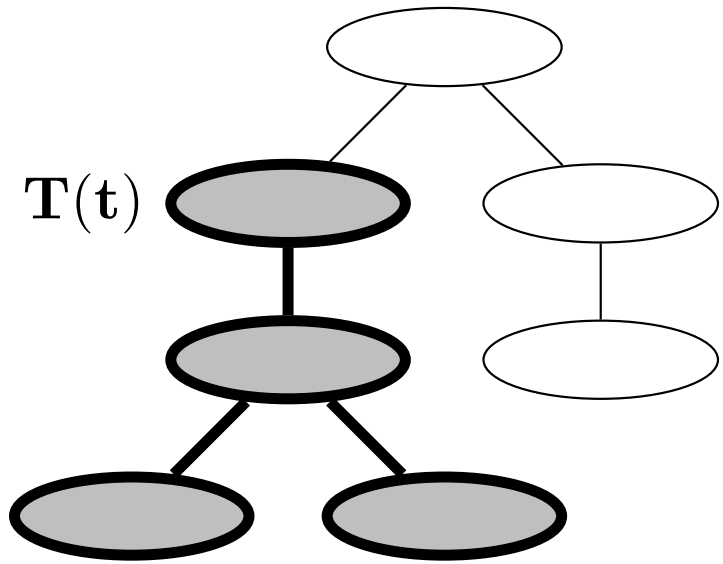
is equivalent to $\mathcal{S}$.

Moreover, the incidence treewidth of $\mathcal{I}$ is at most 1 and all variables are binary.

An Algorithm using Treewidth and Domain (Main Idea)

Question?

Which information do we need to store for feasible assignments τ of $\mathcal{I}(t)$?



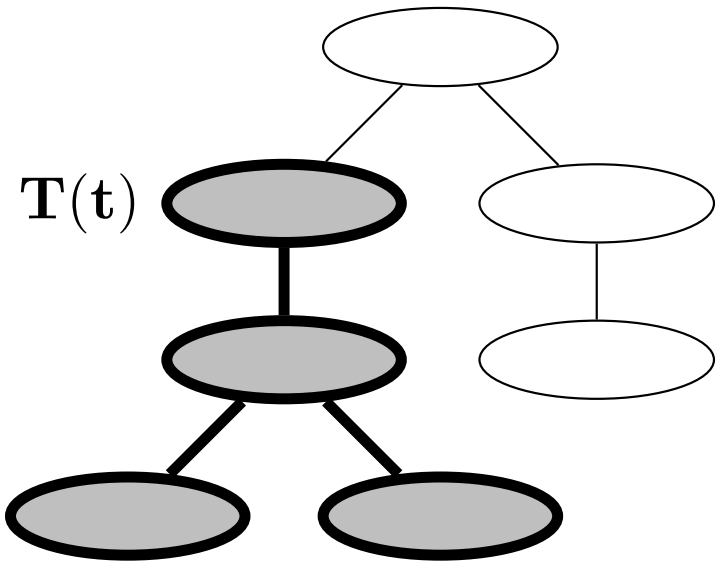
An Algorithm using Treewidth and Domain (Main Idea)

Question?

Which information do we need to store for feasible assignments τ of $\mathcal{I}(t)$?

Answer:

- For the variables in $X(t)$ it is again sufficient to store their assignments since "future" constraints only share the variables in $X(t)$ with $V(t)$.



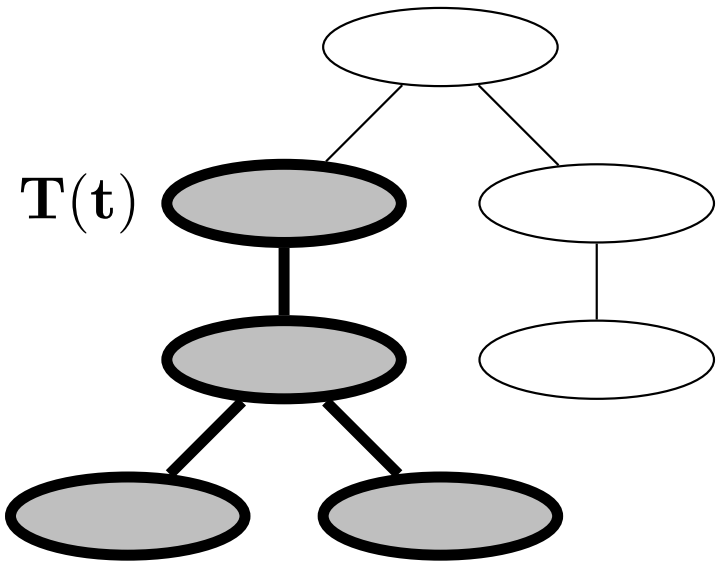
An Algorithm using Treewidth and Domain (Main Idea)

Question?

Which information do we need to store for feasible assignments τ of $\mathcal{I}(t)$?

Answer:

- For the variables in $X(t)$ it is again sufficient to store their assignments since "future" constraints only share the variables in $X(t)$ with $V(t)$.
- However, since the constraints in $X(t)$ can be over variables both inside and outside of $V(t)$, we need to know all possible values they can evaluate to.



An Algorithm using Incidence Treewidth

Definition

For a constraint C and a (partial) assignment τ of (some of) the variables of C, let $C(\tau)$ denote the evaluation of C under τ.

Example

Let $C = 2x_1 + 3x_2 + 5x_3 = 8$ and let $\tau(x_1) = 5$ and $\tau(x_3) = 2$, then:

$$C(\tau) = 2\tau(x_1) + 5\tau(x_3) = 2 \cdot 5 + 5 \cdot 2 = 20$$

An Algorithm using Incidence Treewidth

Definition

For a constraint C and a (partial) assignment τ of (some of) the variables of C, let $C(\tau)$ denote the evaluation of C under τ.

Example

Let $C = 2x_1 + 3x_2 + 5x_3 = 8$ and let $\tau(x_1) = 5$ and $\tau(x_3) = 2$, then:

$$C(\tau) = 2\tau(x_1) + 5\tau(x_3) = 2 \cdot 5 + 5 \cdot 2 = 20$$

Definition

Let $\Gamma(\mathcal{I})$ be the maximum absolute value $C(\tau)$ over any constraint C of $\mathcal{I}$ and any feasible partial assignment τ of $\mathcal{I}$.

Discussion of the Algorithm

Theorem (Ganian, O., and Ramanujan, 2017)

An ILP instance $\mathcal{I}$ with n variables and m constraints can be solved in time:

$$\mathcal{O}(\Gamma(\mathcal{I})^{\mathsf{tw}(\mathcal{I})}(n+m))$$

Discussion of the Algorithm

Theorem (Ganian, O., and Ramanujan, 2017)

An ILP instance $\mathcal{I}$ with n variables and m constraints can be solved in time:

$$\mathcal{O}(\Gamma(\mathcal{I})^{\mathsf{tw}(\mathcal{I})}(n+m))$$

Remark

Because $\Gamma(\mathcal{I}) \leq \ell_A \times D \times n$, it follows that ILP can be solved in polynomial-time for bounded incidence treewidth provided that both ℓ and D are polynomially bounded in the input size.

Discussion of the Algorithm

Theorem (Ganian, O., and Ramanujan, 2017)

An ILP instance $\mathcal{I}$ with n variables and m constraints can be solved in time:

$$\mathcal{O}(\Gamma(\mathcal{I})^{\mathsf{tw}(\mathcal{I})}(n+m))$$

Remark

Because $\Gamma(\mathcal{I}) \leq \ell_A \times D \times n$, it follows that ILP can be solved in polynomial-time for bounded incidence treewidth provided that both ℓ and D are polynomially bounded in the input size.

Remark

Our previous hardness results for primal treewidth shows that ILP becomes NP-complete again if only one of ℓ or D are allowed to grow exponentially in the input size.

Discussion of the Algorithm

Remark

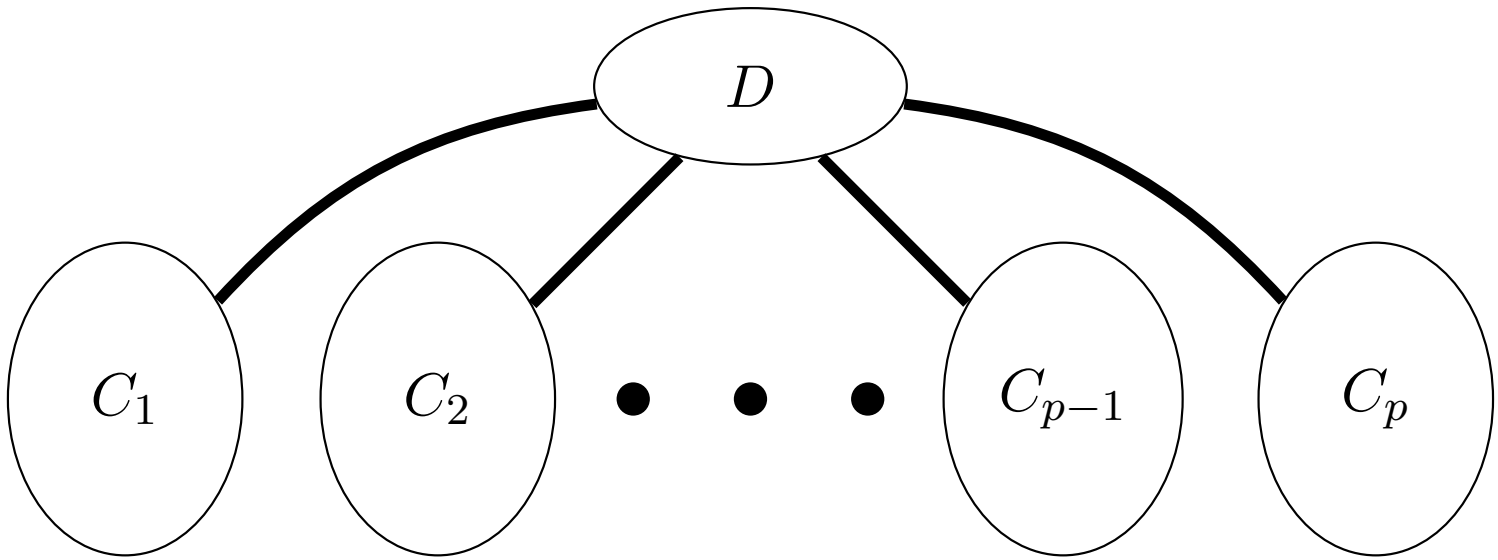
If we want to find an fpt-algorithm parameterized by ℓ and some additional structural parameter of the incidence graph, we need to employ a more restrictive parameter than treewidth.

- A natural candidate would be treedepth, however, at this point it remains open whether ILP is fixed-parameter tractable parameterized by treedepth and ℓ.
- Nevertheless, we can show such a result for a slightly more restrictive parameter than treedepth, which we call the **fracture number**.

Fracture Number

Definition

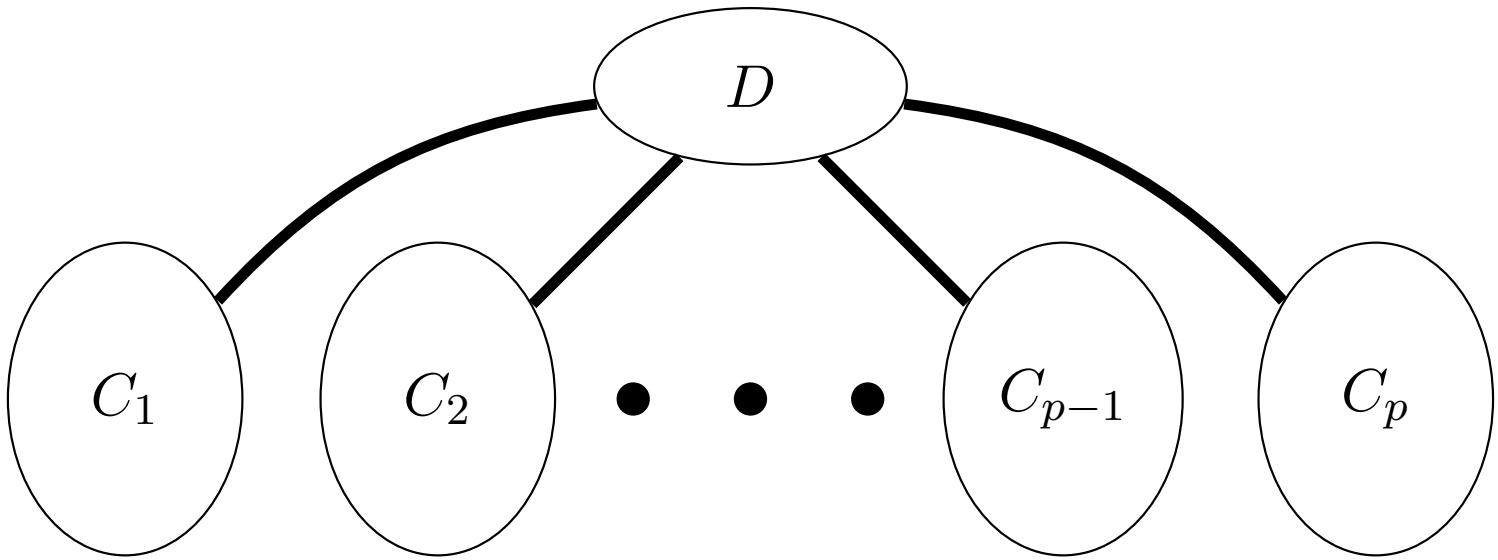
The **fracture number** of a graph G is the minimum number k such that G has a deletion set D of at most k vertices such that every component of $G \setminus D$ has size at most k.



Fracture Number

Definition

The **fracture number** of a graph G is the minimum number k such that G has a deletion set D of at most k vertices such that every component of $G \setminus D$ has size at most k.



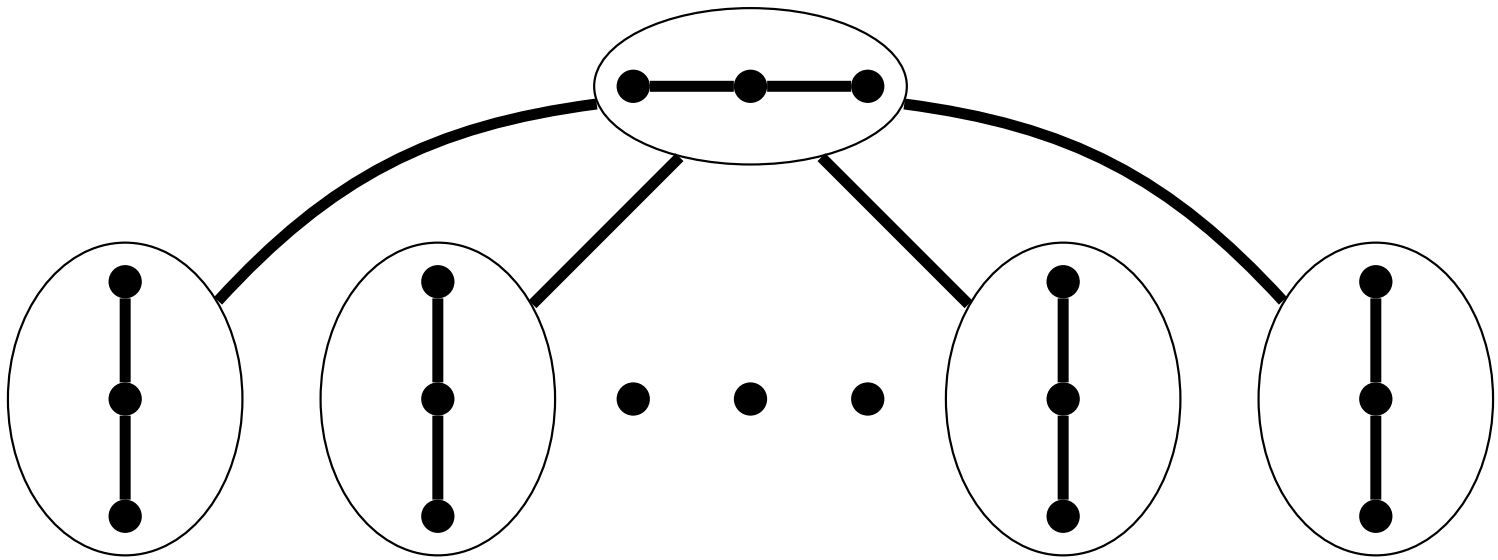
Remark

The treedepth/treewidth is at most two times the fracture number.

Fracture Number

Definition

The **fracture number** of a graph G is the minimum number k such that G has a deletion set D of at most k vertices such that every component of $G \setminus D$ has size at most k.



Remark

The treedepth/treewidth is at most two times the fracture number.

Fracture Number for ILP

$$
\begin{array}{c}
 \\
D \cap C \\
C_1 \cap C \\
C_2 \cap C \\
\vdots \\
C_p \cap C
\end{array}
\begin{array}{c}
\begin{array}{ccccc}
D \cap V & C_1 \cap V & C_2 \cap V & \cdots & C_p \cap V
\end{array} \\
\left(\begin{array}{ccccc}
* & * & * & \cdots & * \\
* & * & 0 & \cdots & 0 \\
* & 0 & * & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
* & 0 & 0 & \cdots & *
\end{array}\right)
\end{array}
$$

Fracture Number for ILP

$$
\begin{array}{c}
 \\
D \cap C \\
C_1 \cap C \\
C_2 \cap C \\
\vdots \\
C_p \cap C
\end{array}
\begin{array}{c}
\begin{array}{c} D \cap V \end{array} \\
\left(\begin{array}{c}
* \\ * \\ * \\ \vdots \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ *
\end{array}\right.
\end{array}
\begin{array}{c}
\begin{array}{c} C_1 \cap V \end{array} \\
\begin{array}{c}
* \\ * \\ * \\ \vdots \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ *
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c} C_2 \cap V \end{array} \\
\begin{array}{c}
* \\ * \\ * \\ \vdots \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ *
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c} \cdots \end{array} \\
\begin{array}{c}
\cdots \\ \cdots \\ \cdots \\ \ddots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \ddots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \ddots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \ddots \\ \cdots
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c} C_p \cap V \end{array} \\
\left.\begin{array}{c}
* \\ * \\ * \\ \vdots \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ *
\end{array}\right)
\end{array}
$$

global variables

Fracture Number for ILP

	$D \cap V$	$C_1 \cap V$	$C_2 \cap V$	$\cdots$	$C_p \cap V$
$D \cap C$	$*$	$*$	$*$	$\cdots$	$*$
$C_1 \cap C$	$*$	$*$	0	$\cdots$	0
$C_2 \cap C$	$*$	0	$*$	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$C_p \cap C$	$*$	0	0	$\cdots$	$*$

global constraints

Fracture Number for ILP

$$
\begin{array}{c}
 \\
D \cap C \\
C_1 \cap C \\
C_2 \cap C \\
\vdots \\
C_p \cap C
\end{array}
\begin{array}{c}
\begin{array}{cccccc}
D \cap V & C_1 \cap V & C_2 \cap V & \cdots & C_p \cap V
\end{array} \\
\left(\begin{array}{cccccc}
* & * & * & \cdots & * \\
* & * & 0 & \cdots & 0 \\
* & 0 & * & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
* & 0 & 0 & \cdots & *
\end{array}\right)
\end{array}
$$

local variables

Fracture Number for ILP

	$D \cap V$	$C_1 \cap V$	$C_2 \cap V$	$\cdots$	$C_p \cap V$
$D \cap C$	$*$	$*$	$*$	$\cdots$	$*$
$C_1 \cap C$	$*$	$*$	0	$\cdots$	0
$C_2 \cap C$	$*$	0	$*$	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$C_p \cap C$	$*$	0	0	$\cdots$	$*$

local constraints

Finding a Fracture Deletion Set

We distinguish three types of fracture deletion sets (**FDS**), namely:

- **variable FDS** are only allowed to contain (global) variables,
- **constraint FDS** are only allowed to contain (global) constraints,
- **mixed FDS** are allowed to contain both.

Finding a Fracture Deletion Set

We distinguish three types of fracture deletion sets (**FDS**), namely:

- **variable FDS** are only allowed to contain (global) variables,
- **constraint FDS** are only allowed to contain (global) constraints,
- **mixed FDS** are allowed to contain both.

Theorem (Dvořák, Eiben, Ganian, Knop, and O., 2017)

Finding a variable/constraint/mixed FDS of size k is **NP**-hard, but **FPT** parameterized by k.

Solving ILP using a small Fracture Deletion Set

Our algorithm for solving ILP using a small fracture deletion set is inspired by **block matrices**:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}^N = \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{B} & \cdots & \mathbf{B} \\ \mathbf{C} & \mathbf{D} & 0 & \cdots & 0 \\ \mathbf{C} & 0 & \mathbf{D} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C} & 0 & 0 & \cdots & \mathbf{D} \end{pmatrix}$$

Solving ILP using a small Fracture Deletion Set

Our algorithm for solving ILP using a small fracture deletion set is inspired by **block matrices**:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}^N = \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{B} & \cdots & \mathbf{B} \\ \mathbf{C} & \mathbf{D} & 0 & \cdots & 0 \\ \mathbf{C} & 0 & \mathbf{D} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C} & 0 & 0 & \cdots & \mathbf{D} \end{pmatrix}$$

Theorem (Hemmecke et al., 2010)

ILP is **XP** parameterized by ℓ_A and the max. number of rows/columns in $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$.

Solving ILP using a small Fracture Deletion Set

Our algorithm for solving ILP using a small fracture deletion set is inspired by **block matrices**:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}^N = \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{B} & \cdots & \mathbf{B} \\ \mathbf{C} & \mathbf{D} & 0 & \cdots & 0 \\ \mathbf{C} & 0 & \mathbf{D} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C} & 0 & 0 & \cdots & \mathbf{D} \end{pmatrix}$$

Theorem (De Loera et al., 2013)

If $\mathbf{A}$ and $\mathbf{C}$ are omitted, then ILP is **FPT** parameterized by ℓ_A and the max. number of rows/columns in $\mathbf{B}, \mathbf{D}$.

Solving ILP using a small Fracture Deletion Set

Our algorithm for solving ILP using a small fracture deletion set is inspired by **block matrices**:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}^N = \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{B} & \cdots & \mathbf{B} \\ \mathbf{C} & \mathbf{D} & 0 & \cdots & 0 \\ \mathbf{C} & 0 & \mathbf{D} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C} & 0 & 0 & \cdots & \mathbf{D} \end{pmatrix}$$

Theorem (Hemmecke et al., 2013)

If $\mathbf{A}$ and $\mathbf{B}$ are omitted, then ILP is **FPT** parameterized by ℓ_A and the max. number of rows/columns in $\mathbf{C}, \mathbf{D}$.

Solving ILP using a small Fracture Deletion Set

Our algorithm for solving ILP using a small fracture deletion set is inspired by **block matrices**:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}^N = \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{B} & \cdots & \mathbf{B} \\ \mathbf{C} & \mathbf{D} & 0 & \cdots & 0 \\ \mathbf{C} & 0 & \mathbf{D} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C} & 0 & 0 & \cdots & \mathbf{D} \end{pmatrix}$$

Essentially, these are ILPs with few global variables and/or global constraints that interact uniformly with the rest.

- Several applications (e.g. Knop, Kouteck, Mnich, 2017).

From Fracture Number to Block Matrices

$$
\begin{array}{c}
 \\
D \cap C \\
C_1 \cap C \\
C_2 \cap C \\
\vdots \\
C_p \cap C
\end{array}
\begin{array}{c}
\begin{array}{ccccc}
D \cap V & C_1 \cap V & C_2 \cap V & \cdots & C_p \cap V
\end{array} \\
\left(\begin{array}{ccccc}
* & * & * & \cdots & * \\
* & * & 0 & \cdots & 0 \\
* & 0 & * & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
* & 0 & 0 & \cdots & *
\end{array}\right)
\end{array}
$$

From Fracture Number to Block Matrices

$$
\begin{array}{c}
 \\
D \cap C \\
C_1 \cap C \\
C_2 \cap C \\
\vdots \\
C_p \cap C
\end{array}
\begin{array}{c}
\begin{array}{ccccc}
D \cap V & C_1 \cap V & C_2 \cap V & \cdots & C_p \cap V
\end{array} \\
\left(\begin{array}{ccccc}
\mathbf{A} & \mathbf{B}_1 & \mathbf{B}_2 & \cdots & \mathbf{B}_p \\
\mathbf{C}_1 & \mathbf{D}_1 & 0 & \cdots & 0 \\
\mathbf{C}_2 & 0 & \mathbf{D}_2 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\mathbf{C}_p & 0 & 0 & \cdots & \mathbf{D}_p
\end{array}\right)
\end{array}
$$

From Fracture Number to Block Matrices

$$\begin{pmatrix} \mathbf{A} & \mathbf{B}_1 & \mathbf{B}_2 & \cdots & \mathbf{B}_p \\ \mathbf{C}_1 & \mathbf{D}_1 & 0 & \cdots & 0 \\ \mathbf{C}_2 & 0 & \mathbf{D}_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_p & 0 & 0 & \cdots & \mathbf{D}_p \end{pmatrix} \overset{?}{\Rightarrow} \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{B} & \cdots & \mathbf{B} \\ \mathbf{C} & \mathbf{D} & 0 & \cdots & 0 \\ \mathbf{C} & 0 & \mathbf{D} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C} & 0 & 0 & \cdots & \mathbf{D} \end{pmatrix}$$

Equivalent Components

We say "components" C_i and C_j are equivalent ($C_i \sim C_j$), if we can obtain

$$\begin{pmatrix} * & \mathbf{B}_i \\ \mathbf{C}_i & \mathbf{D}_i \end{pmatrix} \quad \text{from} \quad \begin{pmatrix} * & \mathbf{B}_j \\ \mathbf{C}_j & \mathbf{D}_j \end{pmatrix}$$

by rearranging columns and rows of the local part.

Lemma

The equivalence $\sim$ has at most $(2\ell_A + 1)^{2k^2}$ equivalence classes, where k is the size of the fracture backdoor.

Equivalent Components

We say "components" C_i and C_j are equivalent ($C_i \sim C_j$), if we can obtain

$$\begin{pmatrix} * & \mathbf{B}_i \\ \mathbf{C}_i & \mathbf{D}_i \end{pmatrix} \quad \text{from} \quad \begin{pmatrix} * & \mathbf{B}_j \\ \mathbf{C}_j & \mathbf{D}_j \end{pmatrix}$$

by rearranging columns and rows of the local part.

Lemma

The equivalence $\sim$ has at most $(2\ell_A + 1)^{2k^2}$ equivalence classes, where k is the size of the fracture backdoor.

Idea: We can group one component of each type into a single block.

Grouping of Components

Assume we have k component types given by the matrices $\mathbf{B}^i$, $\mathbf{C}^i$, and $\mathbf{D}^i$.

$$\begin{pmatrix} * & \mathbf{B}^i \\ \mathbf{C}^i & \mathbf{D}^i \end{pmatrix}$$

Grouping of Components

Assume we have k component types given by the matrices $\mathbf{B}^i$, $\mathbf{C}^i$, and $\mathbf{D}^i$.

$$\begin{pmatrix} * & \mathbf{B}^i \\ \mathbf{C}^i & \mathbf{D}^i \end{pmatrix}$$

By grouping one component of each type into a block, we obtain a block with a bounded number of columns and rows.

$$\begin{pmatrix} * & \mathbf{B^1} & \mathbf{B^2} & \cdots & \mathbf{B^k} \\ \mathbf{C^1} & \mathbf{D^1} & 0 & \cdots & 0 \\ \mathbf{C^2} & 0 & \mathbf{D^2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C^k} & 0 & 0 & \cdots & \mathbf{D^k} \end{pmatrix}$$

Grouping of Components

Assume we have k component types given by the matrices $\mathbf{B}^i$, $\mathbf{C}^i$, and $\mathbf{D}^i$.

$$\begin{pmatrix} * & \mathbf{B}^i \\ \mathbf{C}^i & \mathbf{D}^i \end{pmatrix}$$

By grouping one component of each type into a block, we obtain a block with a bounded number of columns and rows.

$$\begin{pmatrix} * & \mathbf{B^1} & \mathbf{B^2} & \cdots & \mathbf{B^k} \\ \mathbf{C^1} & \mathbf{D^1} & 0 & \cdots & 0 \\ \mathbf{C^2} & 0 & \mathbf{D^2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C^k} & 0 & 0 & \cdots & \mathbf{D^k} \end{pmatrix}$$

We can now repeat this block as many times as the maximum number of components of any type to obtain our block matrix.

Grouping of Components

Assume we have k component types given by the matrices $\mathbf{B}^i$, $\mathbf{C}^i$, and $\mathbf{D}^i$.

$$\begin{pmatrix} * & \mathbf{B}^i \\ \mathbf{C}^i & \mathbf{D}^i \end{pmatrix}$$

By grouping one component of each type into a block, we obtain a block with a bounded number of columns and rows.

$$\begin{pmatrix} * & \mathbf{B^1} & \mathbf{B^2} & \cdots & \mathbf{B^k} \\ \mathbf{C^1} & \mathbf{D^1} & 0 & \cdots & 0 \\ \mathbf{C^2} & 0 & \mathbf{D^2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C^k} & 0 & 0 & \cdots & \mathbf{D^k} \end{pmatrix}$$

We can now repeat this block as many times as the maximum number of components of any type to obtain our block matrix.

Caveat: We do not have the same number of components for every component type.

Solution to the Caveat

We extend each type

$$\begin{pmatrix} * & \mathbf{B}_i \\ \mathbf{C}_i & \mathbf{D}_i \end{pmatrix} \rightsquigarrow \begin{pmatrix} * & \mathbf{B}_i\mathbf{0} \\ \mathbf{C}_i & \mathbf{D}_i\mathbf{D}_i \end{pmatrix}$$

and we add fresh copies of each type.

Using domain restrictions on single variables, we then turn a component:

- ON by forcing the copied variables to be 0 and
- OFF by forcing the original variables to be 0.

Summary: Fracture Number

ℓ_A	Variable	Constraint	Mixed
param.	**FPT**	**FPT**	**XP**

Summary: Fracture Number

ℓ_A	Variable	Constraint	Mixed
param.	**FPT**	**FPT**	**XP**
unary	**para-NP**	**XP**, **W[1]**-h	**para-NP**
arbitrary	**para-NP**	**para-NP**	**para-NP**

Summary: Fracture Number

ℓ_A	Variable	Constraint	Mixed
param.	**FPT**	**FPT**	**XP**
unary	**para-NP**	**XP**, **W[1]**-h	**para-NP**
arbitrary	**para-NP**	**para-NP**	**para-NP**

Remark

The **XP** result for Unary ILP involves two main steps:

- employs a classical proof technique from (Papadimitriou, 1981) adapted to the restricted structure of our matrix to show that the domain D is bounded by a polynomial in the input size,
- it then employs the result on bounded incidence treewidth.

Summary: Fracture Number

ℓ_A	Variable	Constraint	Mixed
param.	**FPT**	**FPT**	**XP**
unary	**para-NP**	**XP**, **W[1]**-h	**para-NP**
arbitrary	**para-NP**	**para-NP**	**para-NP**

Open Problem

- Is ILP in **FPT** for mixed backdoor sets?
- The corresponding problem for N-**fold 4-block matrices** is a long-standing and prominent open problem for ILP.

Summary: Incidence Graph

- In contrast to the primal graph algorithms for the incidence graph also allow us to solve instances with high arity.
- The results on the fracture number provide a natural generalization on the known meta-theorems on block matrices.

Open Problems

- Is ILP in **FPT** for mixed backdoor sets?
- Can the results for fracture number be lifted to treedepth?

Summary and Conclusions

- great potential for novel meta-theorems for NP-complete optimization problems.
- the study of ILP w.r.t. to structural restrictions on the constraint matrix is still in its infancy,
- many interesting directions remain to explore, e.g., combining structural parameterizations with properties of the coefficients (some form of "signed incidence clique-width"),

Thank You!

Thank You!

(. . . see you at the ALGO Welcome Reception!)