

# Parameter Treewidth

Serge Gaspers

## Contents

1	Algorithms for trees	1
2	Tree decompositions	2
3	Monadic Second Order Logic	4
4	Dynamic Programming over Tree Decompositions	6
4.1	SAT	7
4.2	CSP	8
5	Further Reading	8

## 1 Algorithms for trees

### Exercise

**Recall:** An *independent set* of a graph  $G = (V, E)$  is a set of vertices  $S \subseteq V$  such that  $G[S]$  has no edge.

#INDEPENDENT SETS ON TREES

Input: A tree  $T = (V, E)$

Output: The number of independent sets of  $T$ .

- Design a polynomial time algorithm for #INDEPENDENT SETS ON TREES

### Solution

- Select an arbitrary root  $r$  of  $T$
- Bottom-up dynamic programming (starting at the leaves) to compute, for each subtree  $T_x$  rooted at  $x$  the values
  - $\#in(x)$ : the number of independent sets of  $T_x$  containing  $x$ , and
  - $\#out(x)$ : the number of independent sets of  $T_x$  not containing  $x$ .
- If  $x$  is a leaf, then  $\#in(x) = \#out(x) = 1$
- Otherwise,

$$\begin{aligned}\#in(x) &= \prod_{y \in \text{children}(x)} \#out(y) \text{ and} \\ \#out(x) &= \prod_{y \in \text{children}(x)} (\#in(y) + \#out(y))\end{aligned}$$

- The final result is  $\#in(r) + \#out(r)$

## Exercise

**Recall:** A *dominating set* of a graph  $G = (V, E)$  is a set of vertices  $S \subseteq V$  such that  $N_G[S] = V$ .

#DOMINATING SETS ON TREES

Input: A tree  $T = (V, E)$

Output: The number of dominating sets of  $T$ .

- Design a polynomial time algorithm for #DOMINATING SETS ON TREES

## Solution

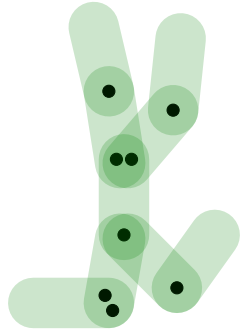
- Select an arbitrary root  $r$  of  $T$
- Bottom-up dynamic programming (starting at the leaves) to compute, for each subtree  $T_x$  rooted at  $x$  the values
  - $\#in(x)$ : the number of dominating sets of  $T_x$  containing  $x$ ,
  - $\#outD(x)$ : the number of dominating sets of  $T_x$  not containing  $x$ , and
  - $\#outND(x)$ : the number of vertex subsets of  $T_x$  dominating  $V(T_x) \setminus \{x\}$ .
- If  $x$  is a leaf, then  $\#in(x) = \#outND(x) = 1$  and  $\#outD(x) = 0$ .
- Otherwise,

$$\begin{aligned}\#in(x) &= \prod_{y \in \text{children}(x)} (\#in(y) + \#outD(y) + \#outND(y)), \\ \#outD(x) &= \prod_{y \in \text{children}(x)} (\#in(y) + \#outD(y)) \\ &\quad - \prod_{y \in \text{children}(x)} \#outD(y) \\ \#outND(x) &= \prod_{y \in \text{children}(x)} \#outD(y)\end{aligned}$$

- The final result is  $\#in(r) + \#outD(r)$

## 2 Tree decompositions

Algorithms using graph decompositions

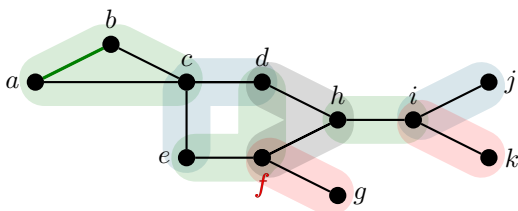


*Idea:* decompose the problem into sub-problems and combine solutions to sub-problems to a global solution.

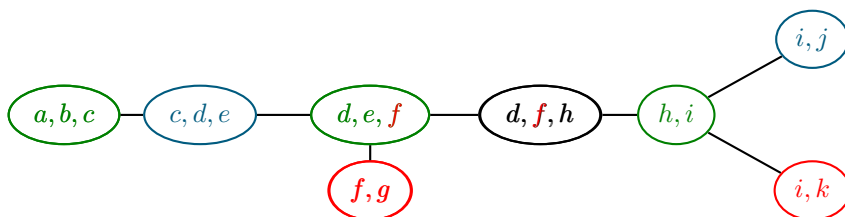
*Parameter:* overlap between subproblems.

Tree decompositions (by example)

- A graph  $G$



- A *tree decomposition* of  $G$



Conditions: **covering** and **connectedness**.

### Tree decomposition (more formally)

- Let  $G$  be a graph,  $T$  a tree, and  $\gamma$  a labeling of the vertices of  $T$  by sets of vertices of  $G$ .
- We refer to the vertices of  $T$  as “nodes”, and we call the sets  $\gamma(t)$  “bags”.
- The pair  $(T, \gamma)$  is a *tree decomposition* of  $G$  if the following three conditions hold:
  1. For every vertex  $v$  of  $G$  there exists a node  $t$  of  $T$  such that  $v \in \gamma(t)$ .
  2. For every edge  $vw$  of  $G$  there exists a node  $t$  of  $T$  such that  $v, w \in \gamma(t)$  (“covering”).
  3. For any three nodes  $t_1, t_2, t_3$  of  $T$ , if  $t_2$  lies on the unique path from  $t_1$  to  $t_3$ , then  $\gamma(t_1) \cap \gamma(t_3) \subseteq \gamma(t_2)$  (“connectedness”).

### Treewidth

- The *width* of a tree decomposition  $(T, \gamma)$  is defined as the maximum  $|\gamma(t)| - 1$  taken over all nodes  $t$  of  $T$ .
- The *treewidth*  $\text{tw}(G)$  of a graph  $G$  is the minimum width taken over all its tree decompositions.

### Basic Facts

- Trees have treewidth 1.
- Cycles have treewidth 2.
- Consider a tree decomposition  $(T, \gamma)$  of a graph  $G$  and two adjacent nodes  $i, j$  in  $T$ . Let  $T_i$  and  $T_j$  denote the two trees obtained from  $T$  by deleting the edge  $ij$ , such that  $T_i$  contains  $i$  and  $T_j$  contains  $j$ . Then, every vertex contained in both  $\bigcup_{a \in V(T_i)} \gamma(a)$  and  $\bigcup_{b \in V(T_j)} \gamma(b)$  is also contained in  $\gamma(i) \cap \gamma(j)$ .
- The complete graph on  $n$  vertices has treewidth  $n - 1$ .
- If a graph  $G$  contains a clique  $K_r$ , then every tree decomposition of  $G$  contains a node  $t$  such that  $K_r \subseteq \gamma(t)$ .

### Complexity of Treewidth

TREEWIDTH	
Input:	Graph $G = (V, E)$ , integer $k$
Parameter:	$k$
Question:	Does $G$ have treewidth at most $k$ ?

- TREEWIDTH is NP-complete.
- TREEWIDTH is FPT: there is a  $k^{O(k^3)} \cdot |V|$  time algorithm (Bodlaender, 1996)

### Easy problems for bounded treewidth

- Many graph problems that are polynomial time solvable on trees are FPT with parameter treewidth.
- Two general methods:
  - *Dynamic programming*: compute local information in a bottom-up fashion along a tree decomposition
  - *Monadic Second Order Logic*: express graph problem in some logic formalism and use a meta-algorithm

### 3 Monadic Second Order Logic

#### Monadic Second Order Logic

- **Monadic Second Order (MSO) Logic** is a powerful formalism for expressing graph properties. One can quantify over vertices, edges, vertex sets, and edge sets.
- **Courcelle's theorem** (Courcelle, 1990). Checking whether a graph  $G$  satisfies an MSO property is FPT parameterized by the treewidth of  $G$  plus the length of the MSO expression.
- **Arnborg et al.'s generalizations** (Arnborg, Lagergren, and Seese, 1991).
  - FPT algorithm for parameter  $\text{tw}(G) + |\phi(X)|$  that takes as input a graph  $G$  and an MSO sentence  $\phi(X)$  where  $X$  is a free (non-quantified) vertex set variable, that computes a minimum-sized set of vertices  $X$  such that  $\phi(X)$  is true in  $G$ .
  - Also, the input vertices and edges may be colored and their color can be tested.

#### Elements of MSO

An MSO formula has

- variables representing vertices  $(u, v, \dots)$ , edges  $(a, b, \dots)$ , vertex subsets  $(X, Y, \dots)$ , or edge subsets  $(A, B, \dots)$  in the graph
- atomic operations
  - $u \in X$ : testing set membership
  - $X = Y$ : testing equality of objects
  - $\text{inc}(u, a)$ : incidence test “is vertex  $u$  an endpoint of the edge  $a$ ?”
- propositional logic on subformulas:  $\phi_1 \wedge \phi_2$ ,  $\phi_1 \vee \phi_2$ ,  $\neg\phi_1$ ,  $\phi_1 \Rightarrow \phi_2$
- Quantifiers:  $\forall X \subseteq V$ ,  $\exists A \subseteq E$ ,  $\forall u \in V$ ,  $\exists a \in E$ , etc.

#### Shortcuts in MSO

We can define some shortcuts

- $u \neq v$  is  $\neg(u = v)$
- $X \subseteq Y$  is  $\forall v \in V. (v \in X) \Rightarrow (v \in Y)$
- $\forall v \in X \varphi$  is  $\forall v \in V. (v \in X) \Rightarrow \varphi$
- $\exists v \in X \varphi$  is  $\exists v \in V. (v \in X) \wedge \varphi$
- $\text{adj}(u, v)$  is  $(u \neq v) \wedge \exists a \in E. (\text{inc}(u, a) \wedge \text{inc}(v, a))$

#### MSO Logic Example

Example: 3-COLORING,

- “there are three independent sets in  $G = (V, E)$  which form a partition of  $V$ ”
- 

$$\begin{aligned} 3\text{COL} := & \exists R \subseteq V. \exists G \subseteq V. \exists B \subseteq V. \\ & \text{partition}(R, G, B) \\ & \wedge \text{independent}(R) \wedge \text{independent}(G) \wedge \text{independent}(B), \end{aligned}$$

where

$$\begin{aligned} \text{partition}(R, G, B) := & \forall v \in V. ((v \in R \wedge v \notin G \wedge v \notin B) \\ & \vee (v \notin R \wedge v \in G \wedge v \notin B) \vee (v \notin R \wedge v \notin G \wedge v \in B)) \end{aligned}$$

and

$$\text{independent}(X) := \neg(\exists u \in X. \exists v \in X. \text{adj}(u, v))$$

By Courcelle's theorem and our 3COL MSO formula, we have:

**Theorem 1.** 3-COLORING is FPT with parameter treewidth.

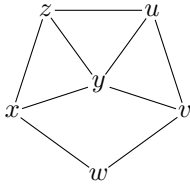
### Treewidth only for graph problems?

Let us use treewidth to solve a Logic Problem

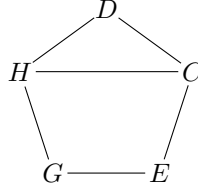
- associate a graph with the instance
- take the tree decomposition of the graph
- most widely used: primal graphs, incidence graphs, and dual graphs of formulas.

### Three Treewidth Parameters

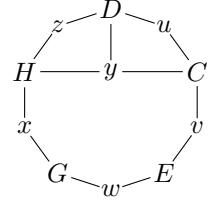
CNF Formula  $F = C \wedge D \wedge E \wedge G \wedge H$  where  $C = (u \vee v \vee \neg y)$ ,  $D = (\neg u \vee z \vee y)$ ,  $E = (\neg v \vee w)$ ,  $G = (\neg w \vee x)$ ,  $H = (x \vee y \vee \neg z)$ .



primal graph



dual graph



incidence graph

This gives rise to parameters *primal treewidth*, *dual treewidth*, and *incidence treewidth*.

**Definition 2.** Let  $F$  be a CNF formula with variables  $\text{var}(F)$  and clauses  $\text{cla}(F)$ . The *primal graph* of  $F$  is the graph with vertex set  $\text{var}(F)$  where two variables are adjacent if they appear together in a clause of  $F$ . The *dual graph* of  $F$  is the graph with vertex set  $\text{cla}(F)$  where two clauses are adjacent if they have a variable in common. The *incidence graph* of  $F$  is the bipartite graph with vertex set  $\text{var}(F) \cup \text{cla}(F)$  where a variable and a clause are adjacent if the variable appears in the clause. The *primal treewidth*, *dual treewidth*, and *incidence treewidth* of  $F$  is the treewidth of the primal graph, the dual graph, and the incidence graph of  $F$ , respectively.

### Incidence treewidth is most general

**Lemma 3.** The incidence treewidth of  $F$  is at most the primal treewidth of  $F$  plus 1.

*Proof.* Start from a tree decomposition  $(T, \gamma)$  of the primal graph with minimum width. For each clause  $C$ :

- There is a node  $t$  of  $T$  with  $\text{var}(C) \subseteq \gamma(t)$ , since  $\text{var}(C)$  is a clique in the primal graph.
- Add to  $t$  a new neighbor  $t'$  with  $\gamma(t') = \gamma(t) \cup \{C\}$ .

□

**Lemma 4.** The incidence treewidth of  $F$  is at most the dual treewidth of  $F$  plus 1.

Primal and dual treewidth are incomparable.

- One big clause alone gives large primal treewidth.
- $\{\{x, y_1\}, \{x, y_2\}, \dots, \{x, y_n\}\}$  gives large dual treewidth.

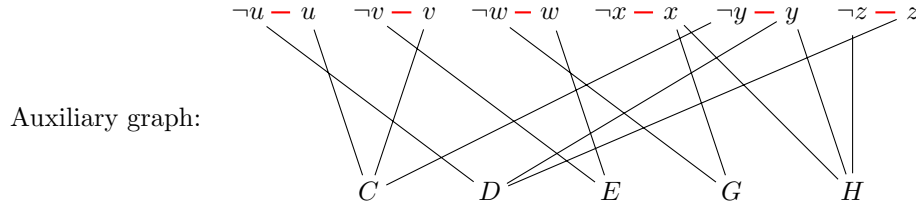
### SAT parameterized by treewidth

SAT	
Input:	A CNF formula $F$
Question:	Is there an assignment of truth values to $\text{var}(F)$ such that $F$ evaluates to true?

**Note:** If SAT is FPT parameterized by incidence treewidth, then SAT is FPT parameterized by primal treewidth and by dual treewidth.

### SAT is FPT for parameter incidence treewidth

CNF Formula  $F = C \wedge D \wedge E \wedge G \wedge H$  where  $C = (u \vee v \vee \neg y)$ ,  $D = (\neg u \vee z \vee y)$ ,  $E = (\neg v \vee w)$ ,  $G = (\neg w \vee x)$ ,  $H = (x \vee y \vee \neg z)$



- MSO Formula: “There exists an independent set of literal vertices that dominates all the clause vertices.”
- The treewidth of the auxiliary graph is at most twice the treewidth of the incidence graph plus one.

### FPT via MSO

**Theorem 5.** SAT is FPT for each of the following parameters: primal treewidth, dual treewidth, and incidence treewidth.

## 4 Dynamic Programming over Tree Decompositions

### Coucelle’s theorem: discussion

Advantages of Courcelle’s theorem:

- general, applies to many problems
- easy to obtain FPT results

Drawback of Courcelle’s theorem

- the resulting running time depends non-elementarily on the treewidth  $t$  and the length  $\ell$  of the MSO-sentence, i.e., a tower of 2’s whose height is  $\omega(1)$

$$2^{2^{2^{\dots^{t+\ell}}}}$$

### Dynamic programming over tree decompositions

Idea: extend the algorithmic methods that work for trees to tree decompositions.

**Step 1** Compute a minimum width tree decomposition using Bodlaender’s algorithm

**Step 2** Transform it into a standard form making computations easier

**Step 3** Bottom-up Dynamic Programming (from the leaves of the tree decomposition to the root)

### Nice tree decomposition

A *nice* tree decomposition  $(T, \gamma)$  is rooted and has only 4 kinds of nodes:

- *leaf node*: leaf  $t$  in  $T$  and  $|\gamma(t)| = 1$
- *introduce node*: node  $t$  with one child  $t'$  in  $T$  and  $\gamma(t) = \gamma(t') \cup \{x\}$
- *forget node*: node  $t$  with one child  $t'$  in  $T$  and  $\gamma(t) = \gamma(t') \setminus \{x\}$
- *join node*: node  $t$  with two children  $t_1, t_2$  in  $T$  and  $\gamma(t) = \gamma(t_1) = \gamma(t_2)$

Every tree decomposition of width  $w$  of a graph  $G$  on  $n$  vertices can be transformed into a nice tree decomposition of width  $w$  and  $O(w \cdot n)$  nodes in polynomial time (Kloks, 1994).

## 4.1 SAT

### Dynamic programming: primal treewidth

- Compute a nice tree decomposition  $(T, \gamma)$  of  $F$ 's primal graph with minimum width rooted at some node  $r$  (Bodlaender, 1996; Kloks, 1994)
- Notation
  - $T_t$  is the subtree of  $T$  rooted at node  $t$
  - $\gamma_\downarrow(t) = \{x \in \gamma(t') : t' \in V(T_t)\}$  is the set of vertices/variables in  $T_t$ 's bags
  - $F_\downarrow(t) = \{C \in \text{cla}(F) : \text{var}(C) \subseteq \gamma_\downarrow(t)\}$  is the set of clauses containing only variables from  $\gamma_\downarrow$
  - For a clause  $C \in \text{cla}(F)$  and an assignment  $\tau : S \rightarrow \{0, 1\}$  to a subset of variables  $S \subseteq \text{var}(F)$ , we can efficiently compute

$$\text{falsifies}(\tau, C) = \begin{cases} 1 & \text{if } \tau \text{ sets each literal of } C \text{ to } 0 \\ 0 & \text{otherwise.} \end{cases}$$

- For each node  $t$  and each assignment  $\tau : \gamma(t) \rightarrow \{0, 1\}$ , our DP algorithm will compute

$$\text{sat}(t, \tau) = \begin{cases} 1 & \text{if } \tau \text{ can be extended to a} \\ & \text{satisfying assignment of } F_\downarrow(t) \\ 0 & \text{otherwise.} \end{cases}$$

- *leaf node*:  $|\gamma(t)| = 1$

$$\text{sat}(t, \tau) = \begin{cases} 0 & \text{if } \exists C \in \text{cla}(F) \text{ s.t. } \text{falsifies}(\tau, C) \\ 1 & \text{otherwise} \end{cases}$$

- *introduce node*:  $\gamma(t) = \gamma(t') \cup \{x\}$ .

$$\text{sat}(t, \tau) = \text{sat}(t', \tau|_{\gamma(t')}) \wedge (\nexists C \in F : \text{falsifies}(\tau, C)).$$

- *forget node*:  $\gamma(t) = \gamma(t') \setminus \{x\}$ .

$$\begin{aligned} \text{sat}(t, \tau) &= \text{sat}(t', \tau_{x=0}) \vee \text{sat}(t', \tau_{x=1}), \\ \text{where } \tau_{x=a}(y) &= \begin{cases} a & \text{if } y = x \\ \tau(y) & \text{otherwise} \end{cases} \end{aligned}$$

- *join node*:  $\gamma(t) = \gamma(t_1) = \gamma(t_2)$

$$\text{sat}(t, \tau) = \text{sat}(t_1, \tau) \wedge \text{sat}(t_2, \tau).$$

- Finally:  $F$  is satisfiable iff  $\exists \tau : \gamma(r) \rightarrow \{0, 1\}$  such that  $\text{sat}(r, \tau) = 1$
- Running time:  $O^*(2^k)$ , where  $k$  is the primal treewidth of  $F$
- Also extends to computing the number of satisfying assignments

### Direct Algorithms

Known treewidth based algorithms for SAT:

$k = \text{primal tw}$	$k = \text{dual tw}$	$k = \text{incidence tw}$
$O^*(2^k)$	$O^*(2^k)$	$O^*(2^k)$

- These algorithms all count the number of satisfying assignments
- The algorithm for incidence treewidth (Slivovsky and Szeider, 2020) uses Fast Subset Convolution

## 4.2 CSP

### Constraint Satisfaction Problem

CSP

Input: A set of variables  $X$ , a domain  $D$ , and a set of constraints  $C$

Question: Is there an assignment  $\tau : X \rightarrow D$  satisfying all the constraints in  $C$ ?

A *constraint* has a *scope*  $S = (s_1, \dots, s_r)$  with  $s_i \in X, i \in \{1, \dots, r\}$ , and a *constraint relation*  $R$  consisting of  $r$ -tuples of values in  $D$ . An assignment  $\tau : X \rightarrow D$  *satisfies* a constraint  $c = (S, R)$  if there exists a tuple  $(d_1, \dots, d_r)$  in  $R$  such that  $\tau(s_i) = d_i$  for each  $i \in \{1, \dots, r\}$ .

### Bounded Treewidth for Constraint Satisfaction

- Primal, dual, and incidence graphs are defined similarly as for SAT.

**Theorem 6** ((Gottlob, Scarcello, and Sideri, 2002)). *CSP is FPT for parameter primal treewidth if  $|D| = O(1)$ .*

- What if domains are unbounded?

### Unbounded domains

**Theorem 7.** *CSP is  $W[1]$ -hard for parameter primal treewidth.*

*Proof Sketch.* Parameterized reduction from CLIQUE. Let  $(G = (V, E), k)$  be an instance of CLIQUE. Take  $k$  variables  $x_1, \dots, x_k$ , each with domain  $V$ . Add  $\binom{k}{2}$  binary constraints  $E_{i,j}$ ,  $1 \leq i < j \leq k$ . A constraint  $E_{i,j}$  has scope  $(x_i, x_j)$  and its constraint relation contains the tuple  $(u, v)$  if  $uv \in E$ . The primal treewidth of this CSP instance is  $k - 1$ .  $\square$

## 5 Further Reading

- Chapter 7, *Treewidth* in (Cygan et al., 2015)
- Chapter 5, *Treewidth* in (Fomin and Kratsch, 2010)
- Chapter 10, *Tree Decompositions of Graphs* in (Niedermeier, 2006)
- Chapter 10, *Treewidth and Dynamic Programming* in (Downey and Fellows, 2013)
- Chapter 13, *Courcelle's Theorem* in (Downey and Fellows, 2013)

## References

- Stefan Arnborg, Jens Lagergren, and Detlef Seese (1991). “Easy problems for tree-decomposable graphs”. In: *Journal of Algorithms* 12.2, pp. 308–340.
- Hans L. Bodlaender (1996). “A linear-time algorithm for finding tree-decompositions of small treewidth”. In: *SIAM Journal on Computing* 25.6, pp. 1305–1317.
- Bruno Courcelle (1990). “The monadic second-order logic of graphs. I. Recognizable sets of finite graphs”. In: *Information and Computation* 85.1, pp. 12–75.
- Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh (2015). *Parameterized Algorithms*. Springer. DOI: 10.1007/978-3-319-21275-3.
- Rodney G. Downey and Michael R. Fellows (2013). *Fundamentals of Parameterized Complexity*. Springer. DOI: 10.1007/978-1-4471-5559-1.
- Fedor V. Fomin and Dieter Kratsch (2010). *Exact Exponential Algorithms*. Springer. DOI: 10.1007/978-3-642-16533-7.
- Georg Gottlob, Francesco Scarcello, and Martha Sideri (2002). “Fixed-parameter complexity in AI and nonmonotonic reasoning”. In: *Journal of Artificial Intelligence* 138.1-2, pp. 55–86.
- Ton Kloks (1994). *Treewidth: Computations and Approximations*. Berlin: Springer.



- Rolf Niedermeier (2006). *Invitation to Fixed Parameter Algorithms*. Oxford University Press. DOI: 10.1093/ACPROF:OSO/9780198566076.001.0001.
- Friedrich Slivovsky and Stefan Szeider (2020). “A Faster Algorithm for Propositional Model Counting Parameterized by Incidence Treewidth”. In: *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT 2020)*. Vol. 12178. Lecture Notes in Computer Science. Springer, pp. 267–276.