

7a. Parameter Treewidth
COMP6741: Parameterized and Exact Computation

Serge Gaspers

Semester 2, 2018

Contents

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

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

Exercise

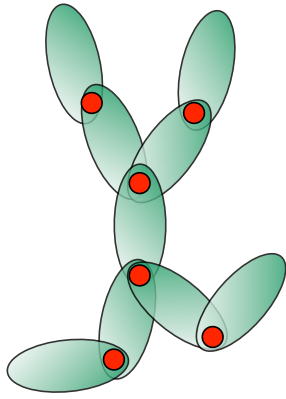
Recall: A *dominating set* of a graph $G = (V, E)$ is a set of vertices $S \subseteq V$ such that $N_G[S] = V$.

#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

2 Tree decompositions

Algorithms using graph decompositions

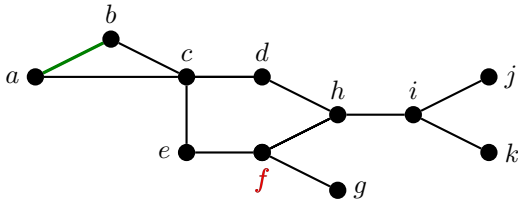


Idea: decompose the problem into sub-problems and combine solutions to sub-problems to a global solution.

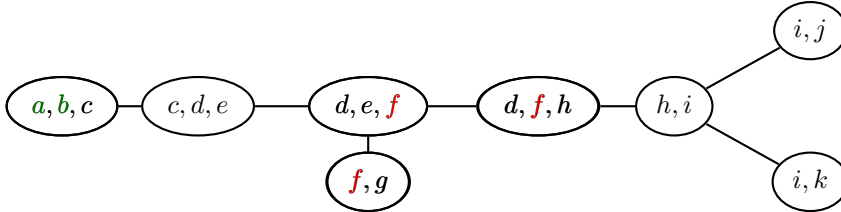
Parameter: overlap between subproblems.

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 γ 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, γ) 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, γ) 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, γ) 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, due to a $k^{O(k^3)} \cdot |V|$ time algorithm by [Bodlaender '96]

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*: Checking whether a graph G satisfies an MSO property is FPT parameterized by the treewidth of G plus the length of the MSO expression. [Courcelle, '90]
- *Arnborg et al.'s generalization*: Several generalizations. For example, 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. [Arnborg, Lagergren, Seese, '91]

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 $\exists v \in V (v \in X \wedge \varphi)$
- $adj(u, v)$ is $(u \neq v) \wedge \exists a \in E (inc(u, a) \wedge inc(v, a))$

MSO Logic Example

Example: 3-Coloring,

- “there are three independent sets in $G = (V, E)$ which form a partition of V ”
- $3COL := \exists R \subseteq V \exists G \subseteq V \exists B \subseteq V \text{partition}(R, G, B) \wedge \text{independent}(R) \wedge \text{independent}(G) \wedge \text{independent}(B)$
 where
 $\text{partition}(R, G, B) := \forall v \in V ((v \in R \wedge v \notin G \wedge v \notin B) \vee (v \notin R \wedge v \in G \wedge v \notin B) \vee (v \notin R \wedge v \notin G \wedge v \in B))$
 and
 $\text{independent}(X) := \neg(\exists u \in X \exists v \in X adj(u, v))$

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

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

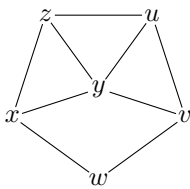
Treewidth only for graph problems?

Let us use treewidth to solve a Logic Problem

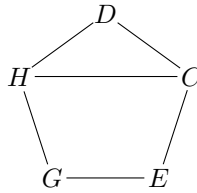
- associate a graph with the instance
- take the tree decomposition of the graph
- most widely used: primal graphs, incidence graphs, and dual graphs of formulas.

Three Treewidth Parameters

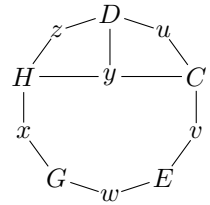
CNF Formula $F = C \wedge D \wedge E \wedge G \wedge H$ where $C = (u \vee v \vee \neg y)$, $D = (\neg u \vee z \vee y)$, $E = (\neg v \vee w)$, $G = (\neg w \vee x)$, $H = (x \vee y \vee \neg z)$.



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, γ) 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

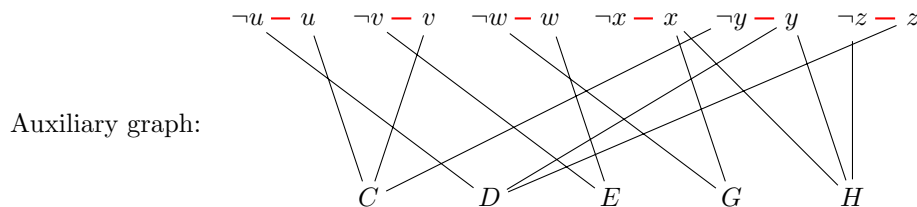
Input: A CNF formula F

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 ℓ of the MSO-sentence, i.e., a tower of 2’s whose height is $\omega(1)$

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

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, γ) has 4 kinds of bags:

- *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 '94].

4.1 Sat

Dynamic programming: primal treewidth

- Compute a nice tree decomposition (T, γ) of F 's primal graph with minimum width [Bodlaender '96; Kloks '94]
- Select an arbitrary root r of T
- Denote T_t the subtree of T rooted at t
- Denote $\gamma_{\downarrow}(t) = \{x \in \gamma(t') : t' \in V(T_t)\}$
- Denote $F_{\downarrow}(t) = \{C \in F : \text{var}(C) \subseteq \gamma_{\downarrow}(t)\}$
- For a node t and an assignment $\tau : \gamma(t) \rightarrow \{0, 1\}$, define

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

Denote $x^1 = x$ and $x^0 = \neg x$. We will view F as a set of clauses and each clause as a set of literals; e.g. $F = \{\{x, \neg y\}, \{\neg x, y, z\}\}$ instead of $F = (x \vee \neg y) \wedge (\neg x \vee y \vee z)$

- *leaf node*: $\text{sat}(t, \{x = a\}) = \begin{cases} 1 & \text{if } \{x^{1-a}\} \notin F \\ 0 & \text{otherwise} \end{cases}$

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

$$\begin{aligned} \text{sat}(t, \{x = a\} \cup \{x_i = a_i\}_i) &= \text{sat}(t', \{x_i = a_i\}_i) \\ &\wedge \nexists C \in F : C \subseteq \{x^{1-a}\} \cup \{x_i^{1-a_i}\}_i. \end{aligned}$$

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

$$\begin{aligned} \text{sat}(t, \{x_i = a_i\}_i) &= \text{sat}(t', \{x = 0\} \cup \{x_i = a_i\}_i) \\ &\vee \text{sat}(t', \{x = 1\} \cup \{x_i = a_i\}_i). \end{aligned}$$

- *join node*:

$$\begin{aligned} \text{sat}(t, \{x_i = a_i\}_i) &= \text{sat}(t_1, \{x_i = a_i\}_i) \\ &\quad \wedge \text{sat}(t_2, \{x_i = a_i\}_i). \end{aligned}$$

- 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 , supposed we are given a minimum width tree decomposition
- Also extends to computing the number of satisfying assignments

Direct Algorithms

Known treewidth based algorithms for SAT:

$$\begin{array}{ccc} k = \text{primal tw} & k = \text{dual tw} & k = \text{incidence tw} \\ O^*(2^k) & O^*(2^k) & O^*(4^k) \end{array}$$

- It is still worth considering primal treewidth and dual treewidth.
- These algorithms all count the number of satisfying assignments.

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, Sideri '02]). *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 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Parameterized Algorithms. Springer, 2015.
- Chapter 5, *Treewidth* in Fedor V. Fomin and Dieter Kratsch. Exact Exponential Algorithms. Springer, 2010.
- Chapter 10, *Tree Decompositions of Graphs* in Rolf Niedermeier. Invitation to Fixed Parameter Algorithms. Oxford University Press, 2006.

- Chapter 10, *Treewidth and Dynamic Programming* in Rodney G. Downey and Michael R. Fellows. Fundamentals of Parameterized Complexity. Springer, 2013.
- Chapter 13, *Courcelle's Theorem* in Rodney G. Downey and Michael R. Fellows. Fundamentals of Parameterized Complexity. Springer, 2013.