7a. Parameter Treewidth

COMP6741: Parameterized and Exact Computation

Serge Gaspers\(^1\)\(^2\)

\(^1\)School of Computer Science and Engineering, UNSW Sydney, Australia
\(^2\)Decision Sciences, Data61, CSIRO, Australia

Semester 2, 2018
1. Algorithms for trees

2. Tree decompositions

3. Monadic Second Order Logic

4. Dynamic Programming over Tree Decompositions
   - Sat
   - CSP

5. Further Reading
Outline

1. Algorithms for trees
2. Tree decompositions
3. Monadic Second Order Logic
4. Dynamic Programming over Tree Decompositions
   - Sat
   - CSP
5. Further Reading
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**
Outline

1. Algorithms for trees

2. Tree decompositions

3. Monadic Second Order Logic

4. Dynamic Programming over Tree Decompositions
   - Sat
   - CSP

5. Further Reading
Idea: decompose the problem into subproblems and combine solutions to subproblems to a global solution.

Parameter: overlap between subproblems.
Tree decompositions (by example)

- A graph $G$

```plaintext
A tree decomposition of $G$
```

```
a, b, c, d, e, f, h, i, j, k
```
Tree decompositions (by example)

- A graph $G$

- A tree decomposition of $G$
Tree decompositions (by example)

- A graph $G$

- A tree decomposition of $G$

Conditions:
Tree decompositions (by example)

- A graph $G$

- A tree decomposition of $G$

Conditions: covering
Tree decompositions (by example)

- A graph \( G \)

- A tree decomposition of \( G \)

Conditions: covering and connectedness.
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 \textit{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”).
The width of a tree decomposition \((T, \gamma)\) is defined as the maximum \(\vert \gamma(t) \vert - 1\) 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 \(K_r \subseteq \gamma(t)\).
**Treewidth**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Graph $G = (V, E)$, integer $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter:</td>
<td>$k$</td>
</tr>
<tr>
<td>Question:</td>
<td>Does $G$ have treewidth at most $k$?</td>
</tr>
</tbody>
</table>

- **Treewidth** is **NP-complete**.
- **Treewidth** is **FPT**, due to a $k^{O(k^3)} \cdot |V|$ time algorithm by [Bodlaender '96]
Many graph problems that are polynomial time solvable on trees are \textbf{FPT} with parameter treewidth.

Two general methods:

- \textit{Dynamic programming}: compute local information in a bottom-up fashion along a tree decomposition
- \textit{Monadic Second Order Logic}: express graph problem in some logic formalism and use a meta-algorithm
**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 $tw(G) + |\phi(X)|$ 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]
An MSO formula has

- variables representing vertices \((u,v,\ldots)\), edges \((a,b,\ldots)\), vertex subsets \((X,Y,\ldots)\), or edge subsets \((A,B,\ldots)\) in the graph

- atomic operations
  - \(u \in X\): testing set membership
  - \(X = Y\): testing equality of objects
  - \(inc(u,a)\): incidence test “is vertex \(u\) an endpoint of the edge \(a\)?”

- propositional logic on subformulas: \(\phi_1 \land \phi_2, \phi_1 \lor \phi_2, \neg \phi_1, \phi_1 \Rightarrow \phi_2\)

- Quantifiers: \(\forall X \subseteq V, \exists A \subseteq E, \forall u \in V, \exists a \in E\), etc.
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 $\exists v \in V \ (v \in X) \land \varphi$
- $\text{adj}(u, v)$ is $(u \neq v) \land \exists a \in E \ (\text{inc}(u, a) \land \text{inc}(v, a))$
Example: 3-Coloring,

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

$3\text{COL} := \exists R \subseteq V \ \exists G \subseteq V \ \exists B \subseteq V$

$\text{partition}(R, G, B) \land \text{independent}(R) \land \text{independent}(G) \land \text{independent}(B)$

where

$\text{partition}(R, G, B) := \forall v \in V ((v \in R \land v \notin G \land v \notin B) \lor (v \notin R \land v \in G \land v \notin B) \lor (v \notin R \land v \notin G \land v \in B))$

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.
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 \( F = C \land D \land E \land G \land H \) where \( C = (u \lor v \lor \neg y) \), \( D = (\neg u \lor z \lor y) \), \( E = (\neg v \lor w) \), \( G = (\neg w \lor x) \), \( H = (x \lor y \lor \neg z) \).

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

**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.
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.
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.
- $\{ \{x, y_1\}, \{x, y_2\}, \ldots, \{x, y_n\} \}$ 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 $\text{SAT}$ is $\text{FPT}$ parameterized by incidence treewidth, then $\text{SAT}$ is $\text{FPT}$ parameterized by primal treewidth and by dual treewidth.
SAT is FPT for parameter incidence treewidth

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

Auxiliary graph:

- 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.
Theorem 5

\textbf{SAT is FPT} for each of the following parameters: primal treewidth, dual treewidth, and incidence treewidth.
Outline

1. Algorithms for trees
2. Tree decompositions
3. Monadic Second Order Logic
4. Dynamic Programming over Tree Decompositions
   - Sat
   - CSP
5. Further Reading
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 $\omega(1)$

$$2^{2^{2^{\ldots^t \ell}}}$$
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)
A **nice** tree decomposition \((T, \gamma)\) 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].
Outline

1. Algorithms for trees
2. Tree decompositions
3. Monadic Second Order Logic
4. Dynamic Programming over Tree Decompositions
   - Sat
   - CSP
5. Further Reading
Compute a nice tree decomposition \((T, \gamma)\) 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 \(\tau : \gamma(t) \to \{0, 1\}\), define

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

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 \( F = (x \lor \neg y) \land (\neg x \lor y \lor z) \)

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

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
\[
F = (x \lor \neg y) \land (\neg x \lor y \lor z)
\]

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

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

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.

$$F = \{\{x, \neg y\}, \{\neg x, y, z\}\} \text{ instead of } F = (x \lor \neg y) \land (\neg x \lor y \lor z)$$

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

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

$$\text{sat}(t, \{x = a\} \cup \{x_i = a_i\}_i) = \text{sat}(t', \{x_i = a_i\}_i) \land \nexists C \in F : C \subseteq \{x^1 - a\} \cup \{x_i^1 - a_i\}_i.$$
**forget node:**

\[
\gamma(t) = \gamma(t') \cup \{x\}.
\]

\[
\text{sat}(t, \{x = a_i\}) = \text{sat}(t', \{x = 0\} \cup \{x = a_i\}) \lor \text{sat}(t', \{x = 1\} \cup \{x = a_i\}).
\]

**join node:**

\[
\text{sat}(t, \{x = a_i\}) = \text{sat}(t_1, \{x = a_i\}) \land \text{sat}(t_2, \{x = a_i\}).
\]

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

\[
\text{sat}(t, \{x_i = a_i\}_i) = \text{sat}(t', \{x = 0\} \cup \{x_i = a_i\}_i) \\
\quad \lor \text{sat}(t', \{x = 1\} \cup \{x_i = a_i\}_i).
\]

join node:

Finally: \( F \) is satisfiable iff \( \exists \tau: \gamma(r) \rightarrow \{0, 1\} \) such that \( \text{sat}(r, \tau) = 1 \).
• **forget node:** $\gamma(t) = \gamma(t') \setminus \{x\}$.

\[
sat(t, \{x_i = a_i\}) = sat(t', \{x = 0\} \cup \{x_i = a_i\}) \\
\lor sat(t', \{x = 1\} \cup \{x_i = a_i\}).
\]

• **join node:**

\[
sat(t, \{x_i = a_i\}) = sat(t_1, \{x_i = a_i\}) \\
\land sat(t_2, \{x_i = a_i\}).
\]
**forget node:** $\gamma(t) = \gamma(t') \setminus \{x\}$.

$$\text{sat}(t, \{x_i = a_i\}_i) = \text{sat}(t', \{x = 0\} \cup \{x_i = a_i\}_i)$$
$$\lor \text{sat}(t', \{x = 1\} \cup \{x_i = a_i\}_i).$$

**join node:**

$$\text{sat}(t, \{x_i = a_i\}_i) = \text{sat}(t_1, \{x_i = a_i\}_i)$$
$$\land \text{sat}(t_2, \{x_i = a_i\}_i).$$

**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 \( \text{SAT} \):

\[
\begin{align*}
  k &= \text{primal tw} & k &= \text{dual tw} & k &= \text{incidence tw} \\
  O^*(2^k) & & O^*(2^k) & & O^*(4^k)
\end{align*}
\]

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

1. Algorithms for trees
2. Tree decompositions
3. Monadic Second Order Logic
4. Dynamic Programming over Tree Decompositions
   - Sat
   - CSP
5. Further Reading
A constraint has a scope \( S = (s_1, \ldots, s_r) \) with \( s_i \in X, i \in \{1, \ldots, r\} \), 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 \((d_1, \ldots, d_r)\) in \( R \) such that \( \tau(s_i) = d_i \) for each \( i \in \{1, \ldots, r\} \).
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?
Theorem 7

CSP is $\text{W}[1]$-hard for parameter primal treewidth.
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 \(x_1, \ldots, x_k\), 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 \(k - 1\).
Outline

1 Algorithms for trees

2 Tree decompositions

3 Monadic Second Order Logic

4 Dynamic Programming over Tree Decompositions
   - Sat
   - CSP

5 Further Reading
Further Reading


