9. Parameter Treewidth

COMP6741: Parameterized and Exact Computation

Serge Gaspers$^{1,2}$

$^1$School of Computer Science and Engineering, UNSW Australia
$^2$Optimisation Resarch Group, NICTA

Semester 2, 2015
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
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,

$$\#_{in}(x) = \prod_{y \text{ child of } x} \#_{out}(y)$$

and

$$\#_{out}(x) = \prod_{y \text{ child of } x} (\#_{in}(y) + \#_{out}(y))$$

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**
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 \( x \),
- \( \#outDom(x) \): 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 \( \#outDom(x) = 0 \).

Otherwise,

\[
\begin{align*}
\#in(x) &= \prod_{y \text{ child of } x} (\#in(y) + \#outDom(y) + \#outNd(y)), \\
\#outDom(x) &= \prod_{y \text{ child of } x} (\#in(y) + \#outDom(y)) \\
&\quad - \prod_{y \text{ child of } x} \#outDom(y) \\
\#outNd(x) &= \prod_{y \text{ child of } x} \#outDom(y)
\end{align*}
\]

The final result is \( \#in(r) + \#outDom(r) \)
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 $G$

\[
\begin{align*}
&\text{A, } b, c, d, e, f, h, i, j, k \\
&\text{a, b, c, d, e, f, h, i, j, k}
\end{align*}
\]
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.
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 \(K_r \subseteq \gamma(t)\).
**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
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 $F(X)$ 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 \((u, v, \ldots)\), 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
  - \(\text{inc}(u, a)\): 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 $V$”
- $3\text{COL} := \exists R \subseteq V \ \exists G \subseteq V \ \exists B \subseteq V$
  \begin{align*}
  &\text{partition}(R, G, B) \land \text{independent}(R) \land \text{independent}(G) \land \text{independent}(B)
  \\
  \text{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))
  \\
  \text{and}
  &\text{independent}(X) := \neg(\exists u \in X \ \exists v \in X \ \text{adj}(u, v))
  
\end{align*}
By Courcelle's theorem and our $3COL$ MSO formula, we have:

**Theorem 1**

3-Colouring is FPT with parameter treewidth.
A *domatic* $k$-*partition* of a graph $G = (V, E)$ is a partition $(D_1, \ldots, D_k)$ of $V$ into $k$ dominating sets of $G$.

### (sol+tw)-Domatic Partition

**Input:** graph $G$, integer $k$

**Parameter:** $k + \text{tw}(G)$

**Question:** Does $G$ have a domatic $k$-partition.

- Show that *(sol+tw)-Domatic Partition* is *FPT* using Courcelle's theorem
\[ \exists D_1 \subseteq V \ \exists D_2 \subseteq V \ \ldots \ \exists D_k \subseteq V \]

\[ \text{partition}(D_1, D_2, \ldots, D_k) \land \]

\[ \forall v \in V \ dom(v, D_1) \land \cdots \land dom(v, D_k) \]

with

\[ dom(v, X) := v \in X \lor \exists x \in X \ adj(v, w) \]
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.
CNF Formula $F = C \land D \land E \land F \land G$ where $C = (u \lor v \lor \neg y)$, $D = (\neg u \lor z \lor y)$, $E = (\neg v \lor w)$, $F = (\neg w \lor x)$, $G = (x \lor y \lor \neg z)$.

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.

Proof.

Exercise.
Lemma 4

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

Proof.

Exercise.

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**

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

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

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 $\omega(1)$.
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
Dynamic programming: primal treewidth

- 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 \(\tau : \gamma(t) \rightarrow \{0, 1\}\), define

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

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

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. \( F = \{ \{x, \neg y\}, \{\neg x, y, z\}\} \) 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:** \( \gamma(t) = \gamma(t') \cup \{x\} \).

\[
\text{sat}(t, \{x = a\} \cup \{x_i = a_i\}_i) = \text{sat}(t', \{x_i = a_i\}_i) \\
\land \#C \in F : C \subseteq \{x^{1-a}\} \cup \{x^{1-a_i}_i\}.
\]
DP: primal treewidth III

- *forget node:*

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

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

\[ \text{join node:} \]

\[ \text{sat}(t, \{ x_i = a_i \}) = \text{sat}(t', \{ x_i = a_i \}) \land \text{sat}(t', \{ x_i = a_i \}) \]

Finally:

\[ F \text{ is satisfiable iff } \exists \tau: \gamma(r) \rightarrow \{ 0, 1 \} \text{ such that } \text{sat}(r, \tau) = 1 \]

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

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

• **join node**: 

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

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

join node: 

\[
sat(t, \{x_i = a_i\}_i) = sat(t', \{x_i = a_i\}_i) \\
\land sat(t', \{x_i = a_i\}_i).
\]


- **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**: 
  
  $$
  \text{sat}(t, \{x_i = a_i\}_i) = \text{sat}(t', \{x_i = a_i\}_i) \\
  \land \text{sat}(t', \{x_i = a_i\}_i).
  $$

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
Known treewidth based algorithms for $\text{SAT}$:

- $k =$ primal tw \hspace{2em} $k =$ dual tw \hspace{2em} $k =$ incidence tw
- $O^*(2^k)$ \hspace{2em} $O^*(2^k)$ \hspace{2em} $O^*(4^k)$

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

2. Tree decompositions

3. Monadic Second Order Logic

4. Dynamic Programming over Tree Decompositions
   - Sat
   - CSP

5. Further Reading
**Constraint Satisfaction Problem**

A **CSP** (Constraint Satisfaction Problem) 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 $i \in \{1, \ldots, r\}$. 

**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 $C$?
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?
- What if we consider incidence treewidth?
Unbounded domains

Theorem 7

CSP is \(W[1]\)-hard for parameter primal treewidth.
Unbounded domains

Theorem 7

CSP is $\mathsf{W}[1]$-hard for parameter primal treewidth.

Proof Sketch.

Parameterized reduction from $\mathsf{CLIQUE}$.
Let $(G = (V, E), k)$ be an instance of $\mathsf{CLIQUE}$.
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 at most $k - 1$. 
Theorem 8

CSP is \(W[1]\)-hard for parameter incidence treewidth and Boolean domain \((D = \{0, 1\})\).

Proof.

Exercise: reduction from \textsc{Clique}.

Hints: (1) Use Boolean variables \(x_{ij}\) with \(1 \leq i \leq k\) and \(1 \leq j \leq n\) with the meaning that \(x_{ij}\) is set to 1 if the \(i\)th vertex of the clique corresponds to the \(j\)th vertex in the graph.

(2) Add \(O(k^2)\) constraints enforcing that for each \(i \in \{1, \ldots, k\}\), exactly one \(x_{ij}\) is set to 1, and whenever two \(x_{ij}, x_{i'j'}\) with \(i \neq i'\) are set to 1, then vertices \(j\) and \(j'\) are adjacent in the graph.

(3) Show that a graph with a vertex cover of size \(q\) has treewidth at most \(q\).
Exercise

<table>
<thead>
<tr>
<th>tw-INDEPENDENT SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
</tr>
<tr>
<td>Parameter:</td>
</tr>
<tr>
<td>Question:</td>
</tr>
</tbody>
</table>

Design an $O^*(2^t)$ time DP algorithm for tw-INDEPENDENT SET.

**Hint**: Proceed as for the presented SAT algorithm, storing the largest size of an independent set extending every in/out labeling of the vertices in a bag to all the vertices contained in bags in the current subtree of the tree decomposition.
Solution sketch

- Obtain a nice tree decomposition \((T, \gamma)\) of width \(t\) in polynomial time.
- Denote \(T_i\) the subtree of \(T\) rooted at node \(i\)
- Denote \(\gamma_\downarrow(i) = \{v \in \gamma(j) : j \in V(T_i)\}\)
- Denote \(G_\downarrow(i) = G[\gamma_\downarrow(i)]\)
- For each node \(i\) of \(T\), and each \(S \subseteq \gamma(i)\), compute \(\text{ind}(i, S)\), the size of a largest independent set of \(G_\downarrow(i)\) that contains all vertices of \(S\) and no vertex from \(\gamma(i) \setminus S\) by dynamic programming.
Solution sketch II

- For a leaf node $i$ with $\gamma(i) = \{v\}$:
  
  \[
  \text{ind}(i, \emptyset) = 0
  \]
  \[
  \text{ind}(i, \{v\}) = 1
  \]

- For a forget node $i$ with child $i'$ and $\gamma(i) = \gamma(i') \setminus \{v\}$:
  
  \[
  \text{ind}(i, S) = \max(\text{ind}(i', S), \text{ind}(i', S \cup \{v\}))
  \]

- For an introduce node $i$ with child $i'$ and $\gamma(i) = \gamma(i') \cup \{v\}$:
  
  \[
  \text{ind}(i, S) = \begin{cases} 
  -\infty & \text{if } G[S] \text{ contains an edge} \\
  \text{ind}(i', S \setminus \{v\}) + [1 \text{ if } v \in S] & \text{otherwise}
  \end{cases}
  \]

- For a join node $i$ with children $i'$ and $i''$:
  
  \[
  \text{ind}(i, S) = \text{ind}(i', S) + \text{ind}(i'', S) - |S|
  \]
Exercise

**tw-Dominating Set**

Input: Graph $G$, integer $k$, and a tree decomposition of $G$ of width at most $t$

Parameter: $t$

Question: Does $G$ have a dominating set of size $k$?

Design an $O^*(9^t)$ time DP algorithm for **tw-Dominating Set**. Can you even achieve an $O^*(4^t)$ time DP algorithm?

**Hint:** Use labeling (in dominating set) / (not in dominating set and needs to be dominated) / (not in dominating set but does not need to be dominated).
Obtain a nice tree decomposition $(T, \gamma)$ of width $t$ in polynomial time.

Denote $T_i$ the subtree of $T$ rooted at node $i$

Denote $\gamma \downarrow (i) = \{ v \in \gamma(j) : j \in V(T_i) \}$

Denote $G \downarrow (i) = G[\gamma \downarrow (i)]$

For each node $i$ of $T$, and each labelling $\ell : \gamma (i) \rightarrow \{in, outDom, outNd\}$, compute the smallest size of a subset $D$ of $\gamma \downarrow (i)$ such that $D \cap \gamma(i)$ is the set of vertices labelled $in$ by $\ell$, and that dominates all vertices from $\gamma \downarrow (i)$ except those that are labeled $outNd$ by $\ell$ by dynamic programming.

The running time depends on how join nodes are handled.

See Section 10.5 in [Niedermeier, '06] for details.
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


