Fixed-Parameter Algorithms, IA166

Sebastian Ordyniak

Faculty of Informatics
Masaryk University Brno

Spring Semester 2013

Outline

1 Treewidth
- Dynamic Programming on Tree Decompositions
- Courcelle's Theorems
- Treewidth Reduction Algorithms

Some Notation

Let (T, X) be a tree decomposition of a graph G. In the following we will assume that T is rooted in some arbitrary node r and hence parent and child relationships between nodes in T are well defined. We will also use the following notations. Let $t \in V(T)$, $U \subseteq V(T)$:

- $T(t)$ denotes the subtree of T rooted at t;
- $X(U)$ denotes the set $\bigcup_{t \in U} X(t)$;
- $V(t)$ denotes the set $X(V(T(t)))$;
- $G(t)$ denotes the graph $G[V(t)]$.

Nice Tree Decompositions

Definition

A tree decomposition (T, X) is nice if $X(r) = \emptyset$ and every node $t \in V(T)$ has one of the following 4 types:

- **Leaf Node:** no children and $|X(t)| = 1$;
- **Introduce Node:** 1 child t' and $X(t) = X(t') \cup \{v\}$ for some $v \in V(G)$;
- **Forget Node:** 1 child t' and $X(t) = X(t') \setminus \{v\}$ for some $v \in V(G)$;
- **Join Node:** 2 children t_1 and t_2, and $X(t) = X(t_1) = X(t_2)$;

Nice Tree Decompositions

Due to their restricted structure nice tree decomposition make the design of dynamic programming algorithms much easier. Furthermore, as the following Proposition shows, the overhead to obtain a nice tree decomposition from a tree decomposition is neglectable.

Proposition (NT)

A tree decomposition of width w and n nodes can be turned into a nice tree decomposition of width w and $O(wn)$ nodes in time $O(w^2 n)$.

The General Approach

The general approach to design bottom-up dynamic programming algorithms is the following:

(1) Find out what is the essential information that we need to know about partial solutions of the problem.

(2) Design the records to store that information.

(3) Can we obtain the solution for the problem from the set of records stored at the root of the tree decompositon?

(4) Can we efficiently compute the records for every of the four types of nodes of a nice tree decomposition?

The Party Problem on Treewidth

Recall:

The Party Problem is the problem to find a maximum weighted independent set of a vertex-weighted graph. We have seen that it can be solved on trees in polynomial time.

We will show next how the dynamic programming approach used to solved the Party Problem on trees can be generalized to graphs of bounded treewidth.

The Party Problem on Treewidth

w-Party Problem — **Parameter: w**

Input: A graph vertex-weighted graph G, a natural number w and a nice tree decomposition (T, X) of G of width at most w.
Question: Compute a the weight of a maximum weight independent set of G.

We will show the following:

Theorem

w-Party Problem can be solved in time $O(2^w w^2 |V(G)|)$ and hence is fixed-parameter tractable.

The Party Problem on Treewidth

As for trees we will use a dynamic programming bottom-up approach that computes sets of records for each node of the tree decomposition.

Let (T, X) be a nice tree decomposition of a vertex-weighted graph G (with weight function w) and let $t \in V(T)$. This time a record is a pair (I, w) where $I \subseteq X(t)$ and w is a real value.

The semantics of a record is as follows:
$(I, w) \in \mathcal{R}(t)$ iff w is the maximum weight of any independent set S of $G(t)$ such that $S \cap X(t) = I$.

Clearly, the solution for the Party Problem can be easily obtained from $\mathcal{R}(r)$ as the real number w such that $(\emptyset, w) \in \mathcal{R}(r)$.

The Party Problem on Treewidth

Let (T, X) be a nice tree decomposition of a vertex weighted graph G with weight function w and let $t \in V(T)$.

t is a leaf node with $X(t) = \{v\}$

$\mathcal{R}(t) := \{ (\emptyset, 0), (X(t), w(v)) \}$.

The Party Problem on Treewidth

Let (T, X) be a nice tree decomposition of a vertex weighted graph G with weight function w and let $t \in V(T)$.

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$

$$\mathcal{R}(t) := \{\, (S \cup \{v\}, w + w(v)) : (S, w) \in \mathcal{R}(t') \text{ and } S \cup \{v\} \text{ is an independent set of } G[X(t)] \,\} \cup \mathcal{R}(t')$$

The Party Problem on Treewidth

Let (T, X) be a nice tree decomposition of a vertex weighted graph G with weight function w and let $t \in V(T)$.

t is a forget node with child t' and $\{v\} = X(t') \setminus X(t)$

$$\mathcal{R}(t) := \{\, (S, \max\{w_1, w_2\}) : S \cap \{v\} = \emptyset \text{ and } (S, w_1) \in \mathcal{R}(t') \text{ and } (S \cup \{v\}, w_2) \in \mathcal{R}(t') \,\}$$

The Party Problem on Treewidth

Let (T, X) be a nice tree decomposition of a vertex weighted graph G with weight function w and let $t \in V(T)$.

t is a join node with children t_1 and t_2

$$\mathcal{R}(t) := \{\, (S, w_1 + w_2 - w(S)) : (S, w_1) \in \mathcal{R}(t_1) \text{ and } (S, w_2) \in \mathcal{R}(t_2) \,\}$$

Run-Time Analysis

Given a nice tree decomposition (T, X) of G the total time required by the above dynamic programming algorithm to solve the PARTY PROBLEM is the number of nodes of T times the maximum time spend on any of the four types of nodes of (T, X).

Because of Proposition (NT) the number of nodes of (T, X) is at most $\mathrm{tw}(G)|V(G)|$.

Question

What is the maximum time spend on any of the four types of nodes of the nice tree decomposition?

The Party Problem on Treewidth

Let (T, X) be a nice tree decomposition of a vertex weighted graph G with weight function w and let $t \in V(T)$.

t is a leaf node with $X(t) = \{v\}$

$\mathcal{R}(t) := \{ (\emptyset, 0), (X(t), w(v)) \}$.

We spend constant time at a leaf node!

The Party Problem on Treewidth

Let (T, X) be a nice tree decomposition of a vertex weighted graph G with weight function w and let $t \in V(T)$.

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$

$$\mathcal{R}(t) := \{ (S \cup \{v\}, w + w(v)) : (S, w) \in \mathcal{R}(t') \text{ and } S \cup \{v\} \text{ is an independent set of } G \} \cup \mathcal{R}(t')$$

Because we have to go over all of the at most 2^w records in $\mathcal{R}(t')$ and for each record we need time $O(w)$ to check whether we again obtain an independent set the total time spend on an introduce node is $O(2^w w)$.

The Party Problem on Treewidth

Let (T, X) be a nice tree decomposition of a vertex weighted graph G with weight function w and let $t \in V(T)$.

t is a forget node with child t' and $\{v\} = X(t') \setminus X(t)$

$$\mathcal{R}(t) := \{\, (S, \max\{w_1, w_2\}) : S \cap \{v\} = \emptyset \text{ and } (S, w_1) \in \mathcal{R}(t') \text{ and } (S \cup \{v\}, w_2) \in \mathcal{R}(t') \,\}$$

Because we have to go over all of the at most 2^w records in $\mathcal{R}(t')$ and for each record we need only constant time the total time spend on a forget node is $O(2^w)$.

The Party Problem on Treewidth

Let (T, X) be a nice tree decomposition of a vertex weighted graph G with weight function w and let $t \in V(T)$.

t is a join node with children t_1 and t_2

$$\mathcal{R}(t) := \{\, (S, w_1 + w_2 - w(S)) : (S, w_1) \in \mathcal{R}(t_1) \text{ and } (S, w_2) \in \mathcal{R}(t_2) \,\}$$

Using appropiate data structures to store the records, e.g., hastables, the total time required for a join node is $O(2^w)$.

Run-Time Analysis

Question

What is the maximum time spend on any of the four types of nodes of the nice tree decomposition?

Answer

$O(2^w w)$.

Theorem

Given a nice tree decomposition (T, X) of a graph G the PARTY PROBLEM can be solved in time $O(2^w w^2 |V(G)|)$.

An Algorithm for 3-COLORING

w-3-COLORING **Parameter: w**

Input: A graph G, a natural number w and a nice tree decomposition (T, X) of G of width at most w.
Question: Does G have a vertex coloring with at most 3 colors?

We will show the following:

Theorem

w-3-COLORING can be solved in time $O(3^w w^2 |V(G)|)$ and hence is fixed-parameter tractable.

An Algorithm for 3-COLORING

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$. This time a record is a 3-vertex coloring $c : X(t) \to \{1, 2, 3\}$ of $X(t)$.

The semantics of a record is as follows:

$c \in \mathcal{R}(t)$ iff c is a 3-vertex coloring of the vertices in $X(t)$ that can be extended to a valid 3-vertex coloring of $G(t)$.

Clearly, the solution for the w-3-COLORING problem can be easily obtained from $\mathcal{R}(r)$ by checking wether $\mathcal{R}(r) \neq \emptyset$.

An Algorithm for 3-COLORING

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a leaf node with $X(t) = \{v\}$

$\mathcal{R}(t) := \{ c : c(v) \in \{1, 2, 3\} \}$.

An Algorithm for 3-COLORING

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a leaf node with $X(t) = \{v\}$

$\mathcal{R}(t) := \{\, c : c(v) \in \{1, 2, 3\} \,\}$.

We spend constant time at a leaf node!

An Algorithm for 3-COLORING

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$

$$\mathcal{R}(t) := \{\, c : X(t) \to \{1,2,3\} : c[X(t')] \in \mathcal{R}(t') \text{ and } c(v) \neq c(w) \text{ for every } w \in N_{G[X(t)]}(v) \,\}$$

An Algorithm for 3-COLORING

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$

$$\mathcal{R}(t) := \{\, c : X(t) \to \{1,2,3\} : c[X(t')] \in \mathcal{R}(t') \text{ and } c(v) \neq c(w) \text{ for every } w \in N_{G[X(t)]}(v) \,\}$$

Because we have to go over all of the at most 3^w records in $\mathcal{R}(t')$ and for each record we need time $O(w)$ to obtain the possible colors for the vertex v the total time spend on an introduce node is $O(3^w w)$.

An Algorithm for 3-Coloring

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a forget node with child t' and $\{v\} = X(t') \setminus X(t)$

$$\mathcal{R}(t) := \{\, c[X(t)] : c \in \mathcal{R}(t') \,\}$$

An Algorithm for 3-Coloring

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a forget node with child t' and $\{v\} = X(t') \setminus X(t)$

$$\mathcal{R}(t) := \{\, c[X(t)] : c \in \mathcal{R}(t') \,\}$$

Because we have to go over all of the at most 3^w records in $\mathcal{R}(t')$ the total time spend on a forget node is $O(3^w)$.

An Algorithm for 3-COLORING

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a join node with children t_1 and t_2

$$\mathcal{R}(t) := \mathcal{R}(t) \cap \mathcal{R}(t')$$

Using appropriate data structures to store the records, e.g., hastables, the total time required for a join node is $O(3^w)$.

Run-Time Analysis

Because the maximum time spend at any of the four types of the nice tree decomposition is $O(3^w w)$ we obtain:

Theorem

w-3-COLORING can be solved in time $O(3^w w^2 |V(G)|)$ and hence is fixed-parameter tractable.

An Algorithm for w-k-Coloring

w-k-Coloring **Parameter: w**

Input: A graph G, a natural number w and a nice tree decomposition (T, X) of G of width at most w.
Question: Does G have a vertex coloring with at most k colors?

By generalizing the above algorithm for w-3-Coloring to w-k-Coloring in the natural way, we obtain:

Theorem

w-k-Coloring can be solved in time $O(k^w w^2 |V(G)|)$ and hence is fixed-parameter tractable.

An Algorithm for w-k-COLORING

Theorem

w-k-COLORING can be solved in time $O(k^w w^2 |V(G)|)$ and hence is fixed-parameter tractable.

Question

The above algorithm only works for the k-COLORING, i.e., if k is fixed and not part of the input. Can we solve the more general w-COLORING problem?

An Algorithm for w-COLORING

w-COLORING — **Parameter: w**

Input: A graph G, 2 natural numbers w and k, and a nice tree decomposition (T, X) of G of width at most w.
Question: Does G have a vertex coloring with at most k colors?

Proposition

Let G be a graph of treewidth at most w. Then G can be colored with at most $w + 1$ colors.

Corollary

w-COLORING can be solved in time $O(w^w w^2 |V(G)|)$ and hence is fixed-parameter tractable.

An Algorithm for w-COLORING

Proposition

Let G be a graph of treewidth at most w. Then G can be colored with at most $w + 1$ colors.

Proof:

We first show that every graph G with $\mathrm{tw}(G) \leq w$ has a vertex of degree at most w. Hence, let (T, X) be a small tree decomposition of G of width at most k and let $l \in V(T)$ be a leaf of T. Because (T, X) is small there is a $v \in X(l)$ that only occurs in l. Hence, using Property T1 we obtain that all neighbors of v in G must be contained in $X(l)$. Hence, v has degree at most w in G.

An Algorithm for w-COLORING

Proposition

Let G be a graph of treewidth at most w. Then G can be colored with at most $w + 1$ colors.

Proof, continued:

Using this fact and the fact that treewidth is closed under taking subgraphs we obtain that every graph of treewidth at most w is w-degenerate and hence can be colored by a simple greedy algorithm with at most $w + 1$ colors. □

An Algorithm for w-Hamiltonian Cycle

w-Hamiltonian Cycle **Parameter: w**

Input: A graph G, a natural number w and a nice tree decomposition (T, X) of G of width at most w.
Question: Does G have a hamiltonian cycle?

We will show the following:

Theorem

w-Hamiltonian Cycle can be solved in time $O((2^w w^{w/2})^2 w^3 |V(G)|)$ and hence is fixed-parameter tractable.

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$. This time a record is a pair (U, R) where $U \subseteq X(t)$ and $R \subseteq [X(t) \setminus U]^2$.

The semantics of a record is as follows:

$(U, R) \in \mathcal{R}(t)$ iff the graph either $U = X(t)$ and $G(t)$ has a hamiltonian cycle or $(G(t) \setminus X(t)) \cup (U \cup \bigcup_{r \in R} r)$ has a partition $\mathcal{P} := \{P_1, \ldots, P_l\}$ into paths such that:

- the endpoints of each path P_i are in $X(t)$;
- U consists of all vertices in $X(t)$ that have degree 2 with respect to $\mathcal{P}$;
- $\mathcal{P}$ contains a path between 2 vertices $u, v \in X(t)$ iff $\{u, v\} \in R$.

An Algorithm for w-Hamiltonian Cycle

$(U, R) \in \mathcal{R}(t)$ iff the graph either $U = X(t)$ and $G(t)$ has a hamiltonian cycle or $(G(t) \setminus X(t)) \cup (U \cup \bigcup_{r \in R} r)$ has a partition $\mathcal{P} := \{P_1, \ldots, P_l\}$ into paths such that:

- the endpoints of each path P_i are in $X(t)$;
- U consists of all vertices in $X(t)$ that have degree 2 with respect to $\mathcal{P}$;
- $\mathcal{P}$ contains a path between 2 vertices $u, v \in X(t)$ iff $\{u, v\} \in R$.

Clearly, the solution for the w-Hamiltonian Cycle problem can be easily obtained from $\mathcal{R}(r)$ by checking wether $\mathcal{R}(r) \neq \emptyset$.

An Algorithm for w-Hamiltonian Cycle

$(U, R) \in \mathcal{R}(t)$ iff the graph either $U = X(t)$ and $G(t)$ has a hamiltonian cycle or $(G(t) \setminus X(t)) \cup (U \cup \bigcup_{r \in R} r)$ has a partition $\mathcal{P} := \{P_1, \ldots, P_l\}$ into paths such that:

- the endpoints of each path P_i are in $X(t)$;
- U consists of all vertices in $X(t)$ that have degree 2 with respect to $\mathcal{P}$;
- $\mathcal{P}$ contains a path between 2 vertices $u, v \in X(t)$ iff $\{u, v\} \in R$.

Clearly, the solution for the w-Hamiltonian Cycle problem can be easily obtained from $\mathcal{R}(r)$ by checking wether $\mathcal{R}(r) \neq \emptyset$.

In the following we denote by $V(R)$ the set $\bigcup_{r \in R} r$!

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a leaf node with $X(t) = \{v\}$

$\mathcal{R}(t) := \{ (\emptyset, \emptyset) \}$.

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a leaf node with $X(t) = \{v\}$

$\mathcal{R}(t) := \{ (\emptyset, \emptyset) \}$.

We spend constant time at a leaf node!

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$

The $\mathcal{R}(t)$ consists of the following sets:

- $\mathcal{R}(t')$, i.e., v has degree 0;
- $\{ (U, R \cup \{n, v\}) : (U, R) \in \mathcal{R}(t') \wedge n \in N_{G[X(t)]}(v) \wedge n \notin U \cup V(R) \}$, i.e., v has degree 1 and is connected to a previously isolated vertex;
- $\{ (U \cup \{n\}, (R \setminus \{u, n\}) \cup \{u, v\}) : (U, R) \in \mathcal{R}(t') \wedge n \in N_{G[X(t)]}(v) \wedge \{u, n\} \in R \}$, i.e., v has degree 1 and is connected to a previous endpoint;

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$, cont.

The $\mathcal{R}(t)$ consists of the following sets:

- $\{ (U \cup \{v\}, R \cup \{n_1, n_2\}) : (U, R) \in \mathcal{R}(t') \wedge n_1, n_2 \in N_{G[X(t)]}(v) \wedge n_1, n_2 \notin U \cup V(R) \}$, i.e., v has degree 2 and is connected to 2 previously isolated vertices;
- $\{ (U \cup \{v, n_1\}, (R \setminus \{\{u, n_1\}\}) \cup \{n_2, u\}) : (U, R) \in \mathcal{R}(t') \wedge n_1, n_2 \in N_{G[X(t)]}(v) \wedge \{n_1, u\} \in R \wedge n_2 \notin U \cup V(R) \}$, i.e., v has degree 2 and is connected to 1 previous endpoint and 1 previously isolated vertex;

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$, cont.

The $\mathcal{R}(t)$ consists of the following sets:

- $\{(U \cup \{v, n_1, n_2\}, (R \setminus \{\{u_1, n_1\}, \{u_2, n_2\}\}) \cup \{u_1, u_2\}) : (U, R) \in \mathcal{R}(t') \wedge n_1, n_2 \in N_{G[X(t)]}(v) \wedge \{u_1, n_1\}, \{u_2, n_2\} \in R\}$, i.e., v has degree 2 and is connected to 2 previous endpoints from different paths;
- $\{(U \cup \{v, n_1, n_2\}, \emptyset) : (U, R) \in \mathcal{R}(t') \wedge n_1, n_2 \in N_{G[X(t)]}(v) \wedge \{n_1, n_2\} \in R \wedge U = X(t') \setminus \{n_1, n_2\}\}$, i.e., v has degree 2 and is connected to 2 previous endpoints from the same path (in this case we get a hamilton cyle for $G(t)$);

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is an introduce node with child t' and $\{v\} = X(t) \setminus X(t')$, cont.

Can be computed in time $O(w^2|\mathcal{R}(t')|)$.

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a forget node with child t' and $\{v\} = X(t') \setminus X(t)$

$$\mathcal{R}(t) := \{ (U \setminus \{v\}, R) : (U, R) \in \mathcal{R}(t') \text{ and } v \in U \}$$

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a forget node with child t' and $\{v\} = X(t') \setminus X(t)$

$$\mathcal{R}(t) := \{ (U \setminus \{v\}, R) : (U, R) \in \mathcal{R}(t') \text{ and } v \in U \}$$

Can be computed in time $O(|\mathcal{R}(t')|)$.

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a join node with children t_1 and t_2

$$\begin{aligned}\mathcal{R}(t) := \{ (U, R) : \\ & (U_1, R_1) \in \mathcal{R}(t_1) \text{ and } (U_2, R_2) \in \mathcal{R}(t_2) \\ & U_1 \cap U_2 = R_1 \cap R_2 = \emptyset \text{ and} \\ & R := \{ \{u, v\} : \text{ the graph } P = ((X(t), R_1 \cup R_2) \text{ has a} \\ & \text{path from } u \text{ to } v \text{ and } u \text{ and } v \text{ have degree 1 in P } \} \\ & U = U_1 \cup U_2 \cup \{ u : \exists_{u', u''} \{\{u, u'\}, \{u, u''\}\} \subseteq R_1 \cup R_2 \} \\ \}\end{aligned}$$

An Algorithm for w-Hamiltonian Cycle

Let (T, X) be a nice tree decomposition of a graph G and let $t \in V(T)$.

t is a join node with children t_1 and t_2

$$\begin{aligned}\mathcal{R}(t) := \{ (U, R) : \\ & (U_1, R_1) \in \mathcal{R}(t_1) \text{ and } (U_2, R_2) \in \mathcal{R}(t_2) \\ & U_1 \cap U_2 = R_1 \cap R_2 = \emptyset \text{ and} \\ & R := \{ \{u, v\} : \text{ the graph } P = ((X(t), R_1 \cup R_2) \text{ has a} \\ & \text{path from } u \text{ to } v \text{ and } u \text{ and } v \text{ have degree 1 in P } \} \\ & U = U_1 \cup U_2 \cup \{ u : \exists_{u', u''} \{\{u, u'\}, \{u, u''\}\} \subseteq R_1 \cup R_2 \} \\ \}\end{aligned}$$

Can be computed in time $O(w^2 |\mathcal{R}(t_1)||\mathcal{R}(t_2)|)$.

Run-Time Analysis

It follows that the maximum time spend at any of the four types of the nice is $O(w^2|\mathcal{R}(t)|^2)$.

Question

What is the maximum number of records?

Run-Time Analysis

- $|\mathcal{R}(t)| \leq 2^{w+1}M$, where M is the number of possible matchings of $X(t)$;
- An easy upper bound for M is $(w+2)!$, which corresponds (assymtotically) to $\sqrt{2\pi(w+2)}(\frac{w+2}{e})^{w+2} = O(\sqrt{w}(w)^w)$ using Stirling's formula;
- A better upper bound can be obtained by realizing that M is equal to the $(w+1)$-th involution number[1], giving us $(\frac{w+1}{e})^{(w+1)/2}\frac{e^{\sqrt{w+1}}}{(4e)^{1/4}} = O(w^{w/2}e^{\sqrt{w}})$;
- Hence, $|\mathcal{R}(t)|$ is at most $O(2^w w^{w/2})$.

[1] en.wikipedia.org/wiki/Telephone_number_(mathematics)

Run-Time Analysis

It follows that the maximum time spend at any of the four types of the nice is $O(w^2|\mathcal{R}(t)|^2) = O(w^2(2^w w^{w/2})^2)$.

Theorem

w-HAMILTONIAN CYCLE can be solved in time $O((2^w w^{w/2})^2 w^3 |V(G)|)$ and hence is fixed-parameter tractable.

Summary: Algorithms on Treewidth

- The majority of NP-hard problems can be solved in polynomial time on graphs of bounded treewidth!
- Using nice tree decompositions helps a lot, at almost no cost.
- The main challenge is finding out which information to store for tree nodes; when this is done, proving formulas for forget, introduce and join nodes is tedious but mostly straightforward.
- Another research subject is finding faster dynamic programming strategies: sometime it may be possible to store and compute fewer combinations.

Outline

1 Treewidth
- Dynamic Programming on Tree Decompositions
- Courcelle's Theorems
- Treewidth Reduction Algorithms

Courcelle's Theorems

Theorem (Decision)

Let Φ be a MSOL formula of lenght k and G be a (directed) graph on n vertices with $\mathrm{tw}(G) \leq w$. Then in time $f(k,w)O(n)$ it can be decided whether G satisfies Φ.

Theorem (Optimization)

Let $\Phi(S)$ be a MSOL formula of lenght k, G be a (directed) graph on n vertices with $\mathrm{tw}(G) \leq w$, and let m be a natural number. Then in time $f(k,w)O(n)$ it can be decided whether there exists an S for G with $|S| \leq m$ (resp. $|S| \geq m$) that satisfies $\Phi(S)$.

A Graph as a Relational Structure

An (undirected) graph G can be represented by a relational structure as follows:

- $U = V(G) \cup E(G)$ (The universe);
- Vx is a unary relation on U that is satisfied if $x \in V(G)$;
- Ex is a unary relation on U that is satisfied if $x \in E(G)$;
- Ixy is a binary relation on U that is satisfied if $x \in V(G)$, $y \in E(G)$, and $x \in y$.

Unary relations are also called sets and instead of Vx we also write $x \in V$.

A Digraph as a Relational Structure

A directed graph G can be represented by a relational structure as follows:

- $U = V(G) \cup E(G)$ (The universe);
- Vx is a unary relation on U that is satisfied if $x \in V(G)$;
- Ex is a unary relation on U that is satisfied if $x \in E(G)$;
- $I^- xy$ ($I^+ xy$) is a binary relation on U that is satisfied if $x \in V(G)$, $y \in E(G)$, and $y = (x, z)$ ($y = (z, x)$) for some $z \in V$.

Monadic Second Order Logic

Monadic Second Order Logic has the following form:

- An infinite set of variables (lower case letters) and relation symbols (upper case letters) is given.
- Atoms are of the form $x = y$ for 2 variables x and y, or $Rx_1 \dots x_k$ for a k-ary relation R.

These are combined into syntactically correct logical formulas using:

- The logical connectives $\neg$, $\vee$, and $\wedge$;
- Universal and existential quantification over variables $\forall x$ and $\exists x$;
- Universal and existential quantification over unary relations (sets) $\forall S$ and $\exists S$.

We also use $\rightarrow$ and $\leftrightarrow$ as logical connectives.

Monadic Second Order Logic

- A variable x or a relation S that is not in the scope of a quantifier binding x or S is called free. We use the convention $\Phi(x_1, \ldots, x_k, S_1, \ldots, S_l)$ to denote that $x_1, \ldots, x_k$ are free variables in Φ and $S_1, \ldots, S_l$ are free relation symbols.
- Because we will apply these formulas to graphs (and digraphs) and view V, E, and I as constants instead of free set variables, the above formula will be called Φ instead of $\Phi(V, E, I)$.
- We employ the usual semantic interpretation of MSOL formulas, e.g., $\Phi \wedge \phi$ is satisfied iff both Φ and ϕ are satisfied, and $\forall x \phi(x)$ is satisfied if for all values of the variable x, $\phi(x)$ is satisfied.

Example

G is bipartite

$$\Phi(V, E, I) := \exists S \exists T((\forall x(Vx \rightarrow (Sx \vee Tx))) \wedge$$
$$\forall x \forall y((\neg(x = y) \wedge \exists z(Ixz \wedge Iyz)) \rightarrow \neg((Sx \wedge Sy) \vee (Tx \wedge Ty))))$$

Simplifying the notations

The example formula expressed a very simple property but was already hard to read. Therefore, in addition to $\rightarrow$, we introduce some more macros and standard formulas that will often be used:

- $S \subseteq T$ stands for $\forall x(Sx \rightarrow Tx)$;
- $\forall x \in S\ \phi(x)$ stands for $\forall x(Sx \rightarrow \phi(x))$, and $\exists x \in S$ is defined analogously;
- $\forall S \subseteq T\ \phi(S)$ stands for $\forall S((\forall x \in S Tx) \rightarrow \phi(S))$, and $\exists S \subseteq T$ is defined analogously;
- $\mathrm{adj}_V(x, y)$ stands for $\neg(x = y) \wedge \exists e \in E(Ixe \wedge Iye)$;
- $\mathrm{adj}_E(e, f)$ stands for $\neg(e = f) \wedge \exists x \in V(Ixe \wedge Ixf)$;
- $\forall xy \in E\ \phi(x, y)$ stands for $\forall x \in V\ \forall y \in V\ \mathrm{adj}_V(x, y) \rightarrow \phi(x, y)$.

More examples

S is a vertex cover

$$\mathrm{VC}(S) := S \subseteq V \wedge \forall e \in E\ \exists s \in S\ I s e$$

S is a dominating set

$$\mathrm{DS}(S) := S \subseteq V \wedge \forall v \in V\ \exists s \in S\ \mathrm{adj}_V(v, s)$$

More examples

G is 3-vertex colorable

$$\begin{gathered}\mathrm{Col}_3 := \exists X_1 \subseteq V\ \exists X_2 \subseteq V\ \exists X_3 \subseteq V \\ (\forall x \in V\ ((X_1 x \vee X_2 x \vee X_3 x) \wedge \\ \neg(X_1 x \wedge X_2 x) \wedge \neg(X_1 x \wedge X_3 x) \wedge \neg(X_2 x \wedge X_3 x)) \wedge \\ \forall xy \in E(\neg(X_1 x \wedge X_1 y) \wedge \neg(X_2 x \wedge X_2 y) \wedge \neg(X_3 x \wedge X_3 y)))\end{gathered}$$

Clearly, k-vertex colorability can be expressed this way for any k (Col_k). However, the formula has length $O(k^2)$.

Some More Macros

- $\exists^{\geq 2} x\ \phi(x)$ stands for $\exists x \exists y\ \phi(x) \wedge \phi(y) \wedge \neg(x = y)$, and $\exists^{\geq k} x \phi(x)$ is defined analogously (observe that the formula length grows with k).
- $\exists^{=k} x\ \phi(x)$ stands for $\exists^{\geq k} x\ \phi(x) \wedge \neg \exists^{\geq k+1} x\ \phi(x)$.
- $S = \emptyset$ stands for $\neg \exists x S x$.
- $\exists S \subsetneq V$ stands for $\exists S \subseteq V \wedge \exists x V x \wedge \neg S x$, and $\forall S \subsetneq V$ is defined analogously.

Some More Examples

The set T of edges induces a connected and spanning subgraph

$$\mathrm{CS}(T) :=$$
$$T \subseteq E \wedge \forall S \subsetneq V(\neg S = \emptyset \rightarrow \exists xy \in T(Sx \wedge \neg Sy))$$

G contains a hamiltonian cycle

$$\mathrm{HC} :=$$
$$\exists T \subseteq E(\mathrm{CS}(T) \wedge \forall x \in V \exists^{=2} e \in T\ Ixe)$$

An Example for Digraphs

C induces a set of vertex disjoint (directed) cycles

$$\mathrm{DC}(C) := C \subseteq E \wedge$$
$$\forall x \in V(\exists e \in C(I^+xe \vee I^-xe) \rightarrow (\exists^{=1} e \in C\ I^+xe \wedge \exists^{=1} e \in C\ I^-xe))$$

S is a (directed) feedback vertex set

$$\mathrm{DFVS}(S) := S \subseteq V \wedge$$
$$\forall C \subseteq A\,(DC(C) \rightarrow \exists x \in S\ \exists e \in CI^+xe)$$

Algorithms for treewidth: Summary

- Many (di)graph problems can be solved in linear time for graphs of bounded treewidth. This includes (almost) all problems treated in this lecture series.
- More precisely, there are FPT algorithms for the general problems parameterized by tree width. (Since an FPT algorithm for deciding tree width exists, we may be sloppy here.)
- For designing an algorithm, dynamic programming is necessary. To decide whether such an algorithm exists, monadic second order logic is a strong tool.

Outline

1 Treewidth
- Dynamic Programming on Tree Decompositions
- Courcelle's Theorems
- Treewidth Reduction Algorithms

Introduction

So far the use of treewidth has be restricted to bounded treewidth graphs. However, as we will see, by using structural properties about a problem, treewidth reduction algorithms can be applied even to instances with unbounded treewidth.

Example: Vertex Cover

Proposition (1)

k-VERTEX COVER can be solved in linear time on graphs of bounded treewidth (Because of Courcelle's Theorem).

Propostion (2)

Let $\mathcal{I} := (G, k)$ be an instance of k-VERTEX COVER such that $\mathrm{tw}(G) > k$. Then $\mathcal{I}$ is a NO instance.

Example: Vertex Cover

Using Proposition (1) and (2) we obtain the following algorithm to solve an instance $\mathcal{I} := (G, k)$ of k-VERTEX COVER:

- Decide whether $\mathrm{tw}(G) \leq k$ and if so compute a tree decomposition $\mathcal{T} = (T, X)$ of G of width at most k.
- If $\mathrm{tw}(G) \leq k$, then use the tree decomposition $\mathcal{T}$ of G to decide $\mathcal{I}$ (Proposition (1)).
- If $\mathrm{tw}(G) > k$ return NO.

Example: Vertex Cover

Using Proposition (1) and (2) we obtain the following algorithm to solve an instance $\mathcal{I} := (G, k)$ of k-VERTEX COVER:

- Decide whether $\mathrm{tw}(G) \leq k$ and if so compute a tree decomposition $\mathcal{T} = (T, X)$ of G of width at most k.
  Running time $O(f(k)|V(G)|)$
- If $\mathrm{tw}(G) \leq k$, then use the tree decomposition $\mathcal{T}$ of G to decide $\mathcal{I}$ (Proposition (1)).
  Running time $O(g(k)|V(G)|)$
- If $\mathrm{tw}(G) > k$ return NO.

Example: Vertex Cover

Using Proposition (1) and (2) we obtain the following algorithm to solve an instance $\mathcal{I} := (G, k)$ of k-VERTEX COVER:

- Decide whether $\text{tw}(G) \leq k$ and if so compute a tree decomposition $\mathcal{T} = (T, X)$ of G of width at most k.
  Running time $O(f(k)|V(G)|)$
- If $\text{tw}(G) \leq k$, then use the tree decomposition $\mathcal{T}$ of G to decide $\mathcal{I}$ (Proposition (1)).
  Running time $O(g(k)|V(G)|)$
- If $\text{tw}(G) > k$ return NO.

This gives an FPT algorithm for k-VERTEX COVER.

Example: Vertex Cover

It remains to show Proposition (2):

Propostion (2)

Let $\mathcal{I} := (G, k)$ be an instance of k-VERTEX COVER such that $\mathrm{tw}(G) > k$. Then $\mathcal{I}$ is a NO instance.

Proof:

It suffices to show that if G has a k-VC $C \subseteq V(G)$ then G has treewidth at most k. Becaus C is a vertex cover it follows that $G \setminus C$ is an independent set and hence has treewidth 0. Thus, let (T, X) be a tree decomposition of $G \setminus C$ of width 0. Then it is straighforward to check that (T, X') with $X' := X \cup C$ is a tree decomposition of G of width at most k. □

Example: Feedback Vertex Set

k-Feedback Vertex Set — **Parameter:** k

Input: A graph G and a natural number k.
Question: Does G have a feedback vertex set of size at most k, i.e., is there a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G \setminus S$ is acyclic?

Question

Is k-Feedback Vertex Set fixed-parameter tractable?

Example: Feedback Vertex Set

Proposition (1)

k-FEEDBACK VERTEX SET can be solved in linear time on graphs of bounded treewidth (Because of Courcelle's Theorem).

Propostion (2)

Let $\mathcal{I} := (G, k)$ be an instance of k-FEEDBACK VERTEX SET such that $\text{tw}(G) > k + 1$. Then $\mathcal{I}$ is a NO instance.

Example: Feedback Vertex Set

Propostion (2)

Let $\mathcal{I} := (G, k)$ be an instance of k-FEEDBACK VERTEX SET such that $\text{tw}(G) > k + 1$. Then $\mathcal{I}$ is a NO instance.

Proof:

It suffices to show that if G has a k-FVS $S \subseteq V(G)$ then G has treewidth at most $k + 1$. Because S is a feedback vertex set it follows that $G \setminus S$ is a forest and hence has treewidth at most 1. Thus, let (T, X) be a tree decomposition of $G \setminus S$ of width 1. Then it is straighforward to check that (T, X') with $X' := X \cup S$ is a tree decomposition of G of width at most $k + 1$. □

Example: Feedback Vertex Set

Question

Is k-FEEDBACK VERTEX SET fixed-parameter tractable?

Answer

Yes, and this time it is not as obvious as for k-VERTEX COVER.

Example: Longest Cycle

k-Long Cycle **Parameter:** k

Input: A graph G and a natural number k.
Question: Does G have a cycle of length at least k?

The above problem is clearly NP-hard (reduction from Hamiltonian Cycle) and not easily seen to be FPT.

Example: Longest Cycle

Proposition (1)

k-LONG CYCLE can be solved in linear time on graphs of bounded treewidth (Because of Courcelle's Theorem).

Propostion (2)

Let $\mathcal{I} := (G, k)$ be an instance of k-LONG CYCLE such that $\text{tw}(G) \geq k - 2$. Then $\mathcal{I}$ is a YES instance.

Example: Longest Cycle

Propostion (2)

Let $\mathcal{I} := (G, k)$ be an instance of k-LONG CYCLE such that $\text{tw}(G) \geq k - 2$. Then $\mathcal{I}$ is a YES instance.

Proof:

Assume to the contrary that G has no cycle of length at least k and let T be a spanning tree of G obtained by **Depth First Search** of G starting in $r \in V(G)$. Because T is obtained via Depth First Search all edges in $E(G) \setminus E(T)$ are between vertices v, w such that w is on the unique path from v to r in T (or vice versa).

Example: Longest Cycle

Propostion (2)

Let $\mathcal{I} := (G, k)$ be an instance of k-LONG CYCLE such that $\mathrm{tw}(G) \geq k - 2$. Then $\mathcal{I}$ is a YES instance.

Proof, cont.:

Furthermore, because G contains no cycle of length at least k all these vertices v and w have distance at most $k - 2$ in T. For $t \in V(T)$ let $A(t)$ be the set of the first $k - 2$ vertices on the unique path from t to r in T. Then (T, X) with $X(t) := \{t\} \cup A(t)$ is a tree decomposition of G of width at most $k - 2$. □

Example: Maximum Leaf Spanning Tree

k-Maximum Leaf Spanning Tree **Parameter: k**

Input: A graph G and a natural number k.
Question: Does G have a spanning tree with at least k leaves?

Previous algorithm used Kernelization!

Example: Maximum Leaf Spanning Tree

Proposition (1)

k-MAXIMUM LEAF SPANNING TREE can be solved in linear time on graphs of bounded treewidth (Because of Courcelle's Theorem).

Propostion (2)

???

Example: Maximum Leaf Spanning Tree

Let G be a graph with spanning tree T. We denote by $L(T)$ the set of leaves of T and by $N^3(T)$ the set of vertices that have at least 1 neighbor with degree at least 3 in T.

Proposition (2A)

Let G and T be as above and let $\{u, v\} \in E(G) \setminus E(T)$ where $u \notin L(T)$ and $v \notin N^3(T)$. Then there is an edge $\{v, w\} \in E(T)$ such that $T - \{v, w\} + \{u, v\})$ is a spanning tree with more leaves than T.

Example: Maximum Leaf Spanning Tree

Proposition (2A)

Let G and T be as above and let $\{u, v\} \in E(G) \setminus E(T)$ where $u \notin L(T)$ and $v \notin N^3(T)$. Then there is an edge $\{v, w\} \in E(T)$ such that $(V(T), (E(T) \setminus \{\{v, w\}\}) \cup \{\{u, v\}\})$ is a spanning tree with more leaves than T.

Proof:

$T + \{u, v\}$ contains a unique cycle, which contains the edge $\{u, v\}$. Let $\{v, w\}$ be the other edge incident to v on that cycle. Clearly, $T + \{u, v\} - \{v, w\}$ is again a spanning tree. Because $v \notin N^3(T)$, w becomes a leaf and because u was not a leaf in T all leaves of T remain leaves. □

Example: Maximum Leaf Spanning Tree

Proposition (2B)

Let T be a spanning tree of G such that every edge of $E(G) \setminus E(T)$ has at least 1 end point in $L(T) \cup N^3(T)$. Then $\text{tw}(G) \leq |L(T) \cup N^3(T)| + 1$.

Proof:

Let (T, X) be a tree decomposition of T of width 1. Then (T, X') with $X' := X \cup L(T) \cup N^3(T)$ is a tree decomposition of G of the required width. □

Example: Maximum Leaf Spanning Tree

Proposition (2C)

For any tree T, it holds that $|N^3(T)| \leq 3|L(T)| - 6$

Proof:

Let H contain the vertices of degree at least 3. Then:

$$|N^3(T)| \leq \sum_{v \in H} d(v) \leq 3 \sum_{v \in H} (d(v) - 2) =$$

$$3(\sum_{v \in V} (d(v) - 2) + |L(T)|) = 3(2|E(T)| - 2|V(T)| + |L(T)|) =$$

$$3(|L(T)| - 2)$$

□

Example: Maximum Leaf Spanning Tree

An Algorithm for k-MLST

(1) Construct a spanning tree T of G
(2) While there is an edge $\{u, v\} \in E(G) \setminus E(T)$ with $u \notin L(T)$ and $v \notin N^3(T)$ do
$T := T + \{u, v\} - \{v, w\}$
(3) If T has at least k leaves then return YES
(4) Use T to construct a tree decomposition (T, X) of G of width at most $4k$.
(5) Answer the problem using dynamic programming over (T, X).

Example: Maximum Leaf Spanning Tree

Analysis of the Algorithm

- Clearly, the algorithm returns the correct answer.
- Because every iteration in Step (2) increases the number of leaves (Proposition (2A)), Step (2) stops after at most $|V(G)|$ iterations.
- Because of Propositions (2B) and (2C) the tree decomposition of width $4k$ used for Step (4) can actually be found.
- Hence, all steps apart from (5) take polynomial time. However, because of Proposition (1), Step (5) can be executed in time $f(4k)O(n)$.

Example: Maximum Leaf Spanning Tree

Analysis of the Algorithm

- Clearly, the algorithm returns the correct answer.
- Because every iteration in Step (2) increases the number of leaves (Proposition (2A)), Step (2) stops after at most $|V(G)|$ iterations.
- Because of Propositions (2B) and (2C) the tree decomposition of width $4k$ used for Step (4) can actually be found.
- Hence, all steps apart from (5) take polynomial time. However, because of Proposition (1), Step (5) can be executed in time $f(4k)O(n)$.

The above algorithm is an FPT algorithm for k-MLST!