Treewidth and Tree-Decompositions

Presenter: Kimberly Crosbie

November 28, 2015

What Is Treewidth?

Intuitive Idea:

- Treewidth generalizes the property of having small seperators
- A graph with small treewidth can be recursively decomposed into small subgraphs that have small overlap
- Treewidth is a measure of how "tree-like" a graph is
- A tree should have small treewidth and a "fat-tree" should also have small treewidth

Why is treewidth interesting?

- Many graph problems that are NP-hard in general can be solved in polynomial time on graphs with small treewidth

What Is Treewidth?

Intuitive Idea:

- Treewidth generalizes the property of having small seperators
- A graph with small treewidth can be recursively decomposed into small subgraphs that have small overlap
- Treewidth is a measure of how "tree-like" a graph is
- A tree should have small treewidth and a "fat-tree" should also have small treewidth

Why is treewidth interesting?

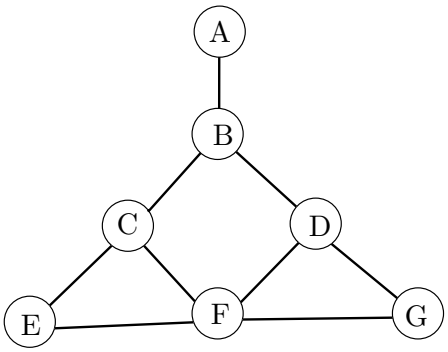
- Many graph problems that are NP-hard in general can be solved in polynomial time on graphs with small treewidth

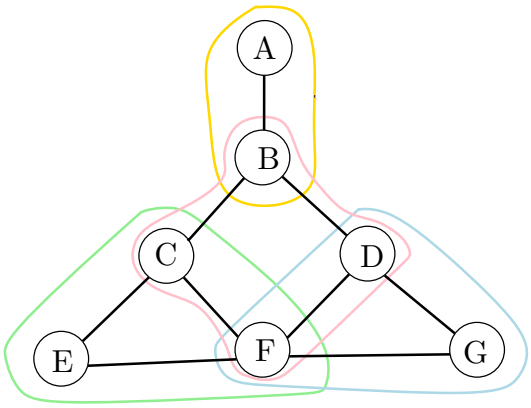
Tree-Decomposition

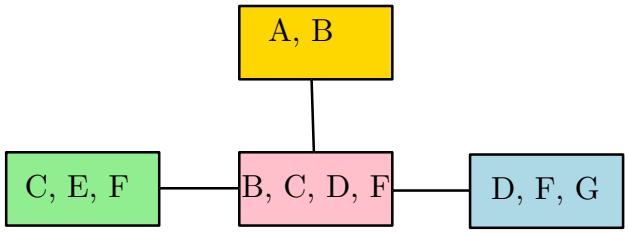
Definition: A *tree decomposition* of a graph $G = (V, E)$ is a tree $T = (I, F)$, where each node $i \in I$ has a label $X_i \subseteq V$ such that:

- $\bigcup_{i \in I} X_i = V$, or equivalently, every vertex $v \in V$ is contained in at least one label
- for any edge $(u, v) \in G$, there exists an $i \in I$ with $u, v \in X_i$
- for any $v \in V$, the vertices containing v in their label form a connected subtree of T, or equivalently, $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$

An Example





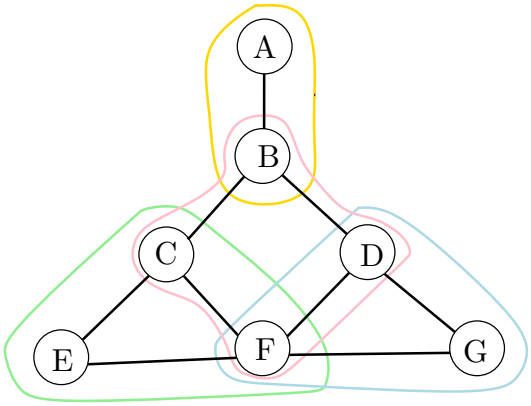


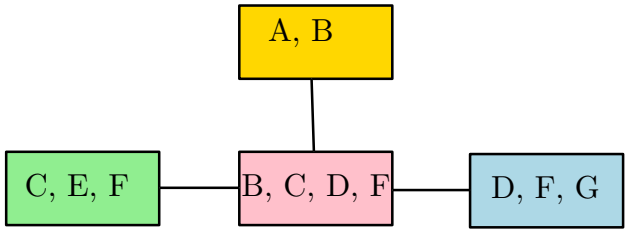
Width and Treewidth

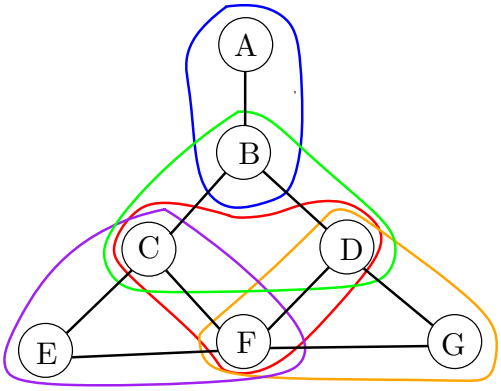
The ***width*** of a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ is $max_{i \in I}|X_i| - 1$.

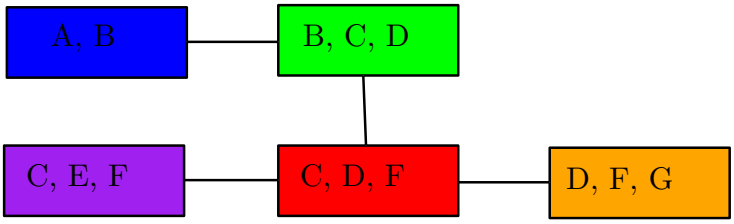
The ***treewidth*** of a graph G is the minimum width over all possible tree decompositions of G.

Further Examples









Maximum Independent Set

Given a graph $G = (V, E)$, we are looking for the maximum size of a set $W \subseteq G$ such that for all $u, v \in W$, the edge $uv \notin E$.

In the general case, this problem is NP-complete, but if we're given a tree-decomposition with bounded treewidth we can do better!

Set-up

Given a tree-decomposition of the graph G with treewidth k, it is easy to form another tree-decomposition, with treewidth k, that is a rooted binary tree.

Suppose we have such a tree-decomposition $\{X_i \mid i \in I\}, T = (I, F)$ where r is the root of T.

For each $i \in I$, we define:
$Y_i = \{v \in X_j \mid j = i$ or j is a descendent of $i\}$.
Let $G[Y_i]$ be the subgraph of G induced by the vertices of Y_i.

How do we approach it?

Key Observation:
When we have an independent set W of $G[Y_i]$ and want to extend that to an independent set of the full graph G we only need to know what vertices of X_i that belong to W. We need not consider what vertices of $Y_i - X_i$ are in W, we only need to know their number.

Our Friend Dynamic Programming

For $i \in I$, $Z \subseteq X_i$, define $S_i(Z)$ to be the maximum size of an independent set W in $G[Y_i]$ with $W \cap X_i = Z$. If no such set exists, we set $S_i(Z) = -\infty$.

We compute all tables of S_i for all nodes $i \in I$ in a bottom-up fashion.

The Algorithm

Leaf node i:
Compute all $2^{|X_i|}$ values in the table S_i by using the following formula:

$$S_i(Z) = \begin{cases} |Z| & \text{if } \forall\, u, v \in Z : uv \notin E \\ -\infty & \text{if } \exists\, u, v \in Z : uv \in E \end{cases}$$

The Algorithm

For an internal node i with children j and k, we compute S_i using the following formula:

If $\forall\ u, v \in Z : uv \notin E$:
$S_i(Z) = max\{S_j(Z') + S_k(Z'') + |Z \cap (X_i - X_j - X_k)| - |Z \cap X_j \cap X_k|\}$, when $Z \cap X_j = Z' \cap X_i$ and $Z \cap X_k = Z'' \cap X_i$

If $\exists\ u, v \in Z : uv \in E$:
$S_i(Z) = -\infty$

The Algorithm

- For each node $i \in I$ we compute the table S_i in bottom-up order until we have computed the table S_r.
- Find the size of the maximum independent set of the graph G by taking $max_{Z \subseteq X_r} S_r(Z)$.
- Therefore, we can solve the problem in $O(2^{3k} n)$ time.

The Algorithm

- For each node $i \in I$ we compute the table S_i in bottom-up order until we have computed the table S_r.
- Find the size of the maximum independent set of the graph G by taking $max_{Z \subseteq X_r} S_r(Z)$.
- Therefore, we can solve the problem in $O(2^{3k}n)$ time.

Hard problems made easy!

Claim: Any problem that can be stated in monadic second order logic can be solved in linear time on graphs with bounded treewidth.

Monadic Second Order Logic

Monadic second order logic is a language to describe graph properties that uses the following constructions:

- Logic operations ($\wedge, \vee, \neg, \Rightarrow$)
- membership tests (e.g, $e \in E$, $v \in V$)
- quantifications over vertices, edges, sets of vertices, sets of edges (e.g., $\forall v \in V$, $\exists e \in E$, $\exists I \subseteq V$, $\forall F \subseteq E$)
- adjacency tests (u is an endpoint of e)
- and some extensions that allow for things such as optimization.

More Monadic Second Order Logic

The 3-colouring problem for graphs can be expressed in monadic second order logic as follows:
$\exists W_1 \subseteq V : \exists W_2 \subseteq V : \exists W_3 \subseteq V : \forall v \in V : (v \in W_1 \vee v \in W_2 \vee v \in W_3) \wedge \forall v \in V : \forall w \in W : (v, w) \in E \Rightarrow (\neg(v \in W_1 \wedge w \in W_1) \wedge \neg(v \in W_2 \wedge w \in W_2) \wedge \neg(v \in W_3 \wedge w \in W_3))$

- Given a graph $G = (V, E)$ and an integer k, the problem to determine if the treewidth of G is at most k had be proven to be NP-complete.
- On special graphs (e.g. tree, circle graphs) it can be solved in constant or polynomial time.
- Open problem: what about planar graphs?

Bibliography


Bodlaender, H.L.
A Tourist Guide through Treewidth
Acta Cybernetica, 11(1-2):1, 1994.