A Theorist's Toolkit **(CMU 18-859T, Fall 2013)**

Lecture 17: Treewidth

November 4, 2013

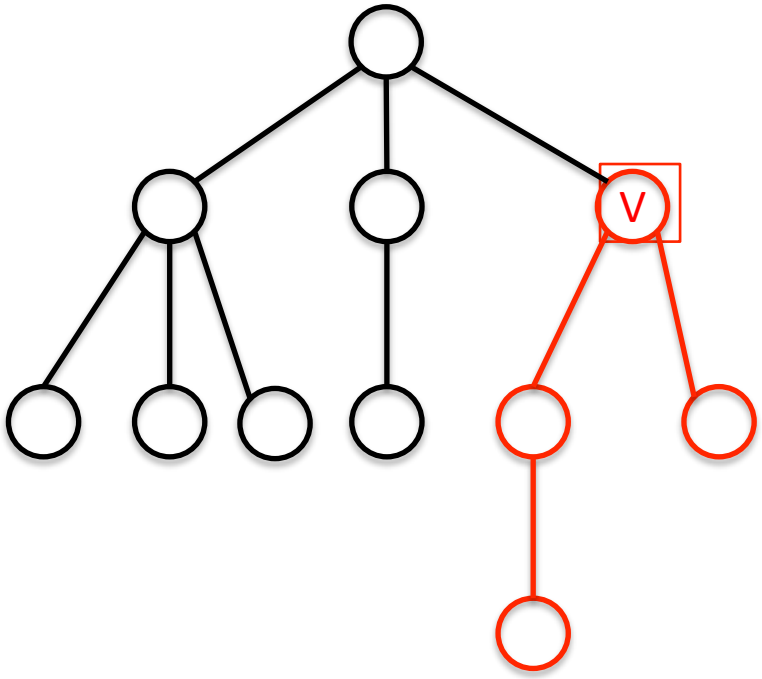
Lecturer: Ryan O'Donnell *Scribe: Anonymous*

1 NP-Hard Problems on Easy Graphs

Intuitively, much of the difficulty in NP-Hard problems lies in untangling the web of connections between the constraints. But if we happen to be working in a situation where there are fewer or more organized constraints, some of these problems become much easier, and are often solvable in polynomial time by a dynamic programming algorithm. In this section, we examine the particular case where the "constraint graph" is a tree.

Example. Suppose we are trying to find the largest independent set of a tree T. First, we choose an arbitrary vertex R to be the root, and represent T as a rooted tree.

For any vertex V, define $T_V$to be the subtree of T consisting of V and all vertices of T directly below it. For instance, on the following tree with labeled vertex V, $T_V$is highlighted in red:



For each vertex V, we compute the following two quantities:

- $M^+[V]$, the size of the largest independent set of T_V containing V
- $M^-[V]$, the size of the largest independent set of T_V not containing V

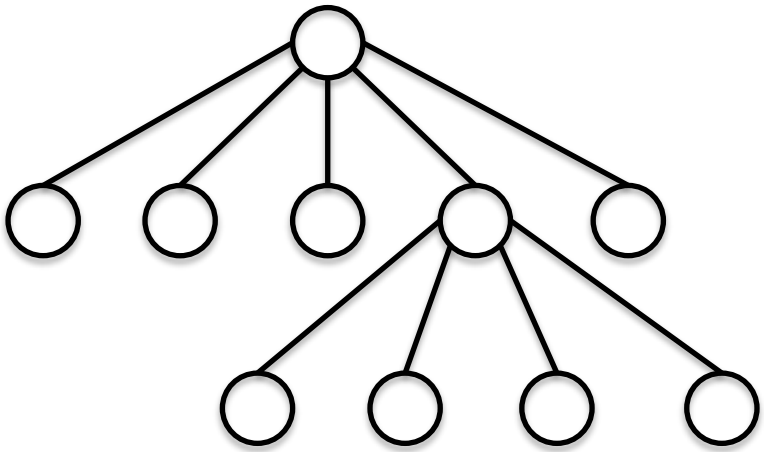
In the tree above, we have and $M^+[V] = M^-[V] = 2$, and if \$V\$ is a leaf, we have that $M^+[V] = 1$ and $M^-[V] = 0$. For a vertex V with children $w_1, ..., w_d$, we can compute these quantities recursively:

$$M^+[V] = 1 + \sum_{i=1}^{d} M^-[w_i]$$

$$M^-[V] = \sum_{i=1}^{d} \max\{M^+[w_i], M^-[w_i]\}$$

Then the size of the largest independent set is $\max\{M^+[R], M^-[R]\}$. This algorithm computes the largest independent set of tree in linear time.

Remark. The natural greedy algorithm is to examine every other layer of the rooted tree, taking the vertices either in all odd rows or in all even rows. This fails badly for the following graph:



A similar result holds for an arbitrary Constraint Satisfaction Problem, although we will need the following concept:

Definition. Given an instance of a CSP with variables V, let G be the graph with vertex set $V$and an edge connecting any two elements of V that appear together in a constraint. We call G the *primal graph* of the CSP.

Claim. Given a CSP instance whose primal graph is a tree (this necessarily means that all constraints contain at most two variables), we can maximize the number of satisfied constraints in linear time.

Proof. As before, we take our primal graph to be a tree T rooted at R. For each variable v and element a of the domain Ω, let $E[v, a]$ denote the maximum number of constraints in T_v that can be satisfied if $v = a$. Let the children of v be $w_1, ..., w_d$, and let $C_i(a, b)$ be the indicator of whether the constraint linking v and w_d is satisfied when$v = a$ and $w_d = b$. Then we can compute $E[v, a]$ recursively by the equation:

$$E[v, a] = \sum_{i=1}^{d} \max\{C_i(a, b) + E[w_i, b] | b \in \Omega\}$$

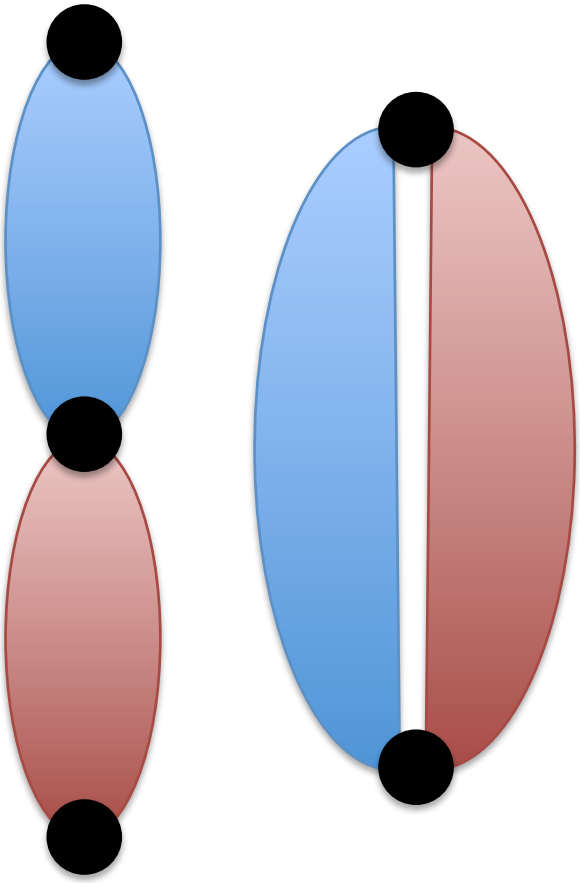
Intuitively, this is tracking the optimal number of constraints we can satisfy for each choice of a. Once we have reached to root, we can easily determine the maximum number of satisfiable constraints as the maximum of $E[R, a]$, as a ranges over all values in Ω. $\square$

While trees may be a very special case, there are other families of graphs for which a similar process works.

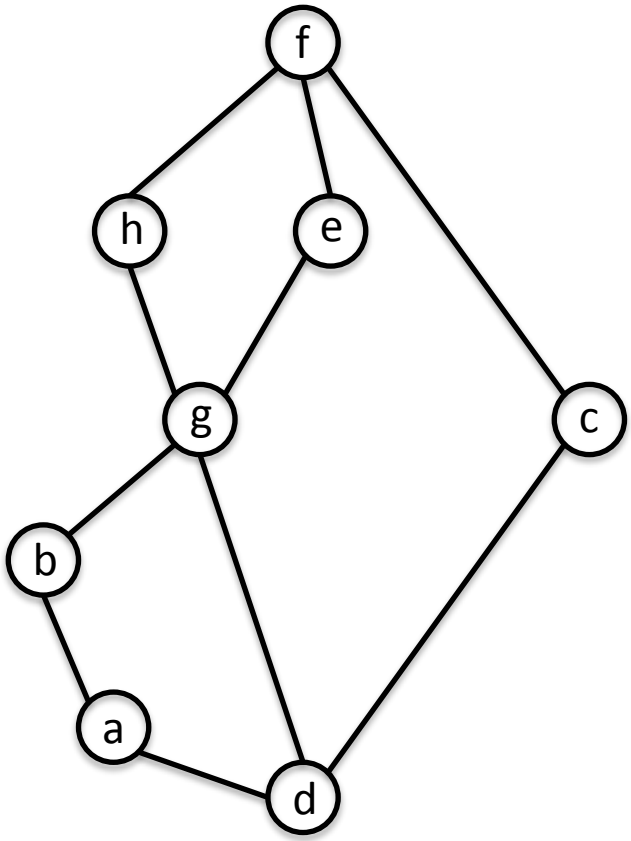
Definition. Let G be a graph with two vertices labeled s and t. We say that G is a *series/parallel graph* if any of the following hold:

- G contains only one edge, from s to t.
- G is the result of sticking two series/parallel graphs together, end to end
- G can be split into two series/parallel graphs, both leading from s to t, whose only common vertices are s and t.

The following diagram illustrates the two ways that two series/parallel circuits may be combined:



An example of a series/parallel graph is below:



As before, we can solve NP-hard CSPs fairly easily with a dynamic programming algorithm if their primal graphs are a subgraph of a series/parallel graph. For example, in finding the maximum independent set of a series/parallel graph, we need to keep track of four values at each step, corrosponding to the maximum independent set that contains both s and t, the one that contains s but not t, etc.

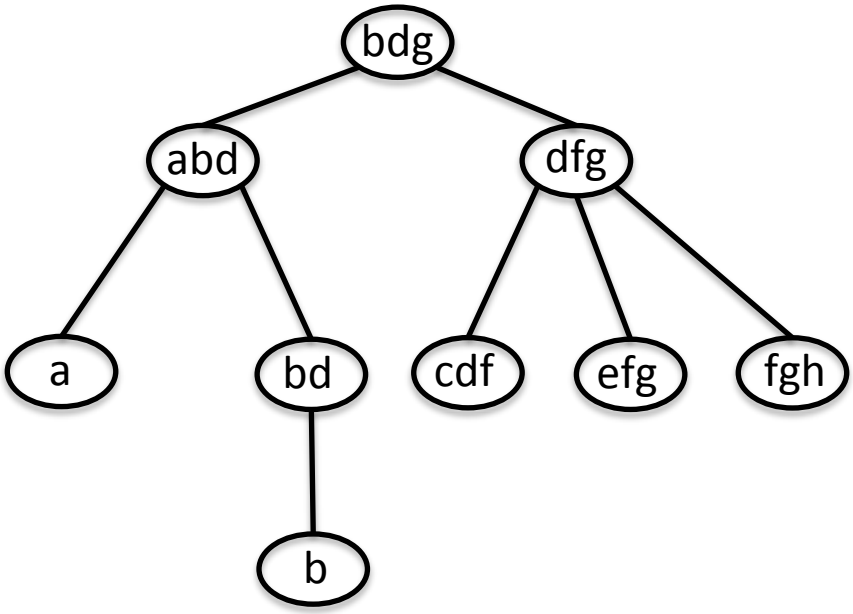
2 Tree Decomposition

While NP-hard problems may be easy on trees and series/parallel graphs, these are extremely special cases, and we would like a generalization to other families of sparse graphs. The folowing idea gives us exactly that.

Definition. Let G be a graph. Let T be a tree whose nodes are "bags" of vertices of G. Then we call T a *tree decomposition* of G if the following hold:

- Each edge is contained in a bag; that is, for any vertices u, v of G, there is some node of T whose bag contains both u and v.
- The subtree of T consisting of all bags containing u is connected, for all vertices u in G.

As an example, here is a tree decomposition of the series/parallel graph from the previous section:



Note that such a decomposition is not unique. In the case above, the the bags containing a, bd, and b are not necessary. One valid tree decomposition contains all vertices of the graph in a single bag. However this is ineffective for the purpose of solving NP-hard problems, so we define the following:

Definition. The *width* of a tree decomposition is one less than the maximum bag size (in the above example, the width is 2).

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

We subtract 1 because we want trees to have treewidth 1. The following facts hold about tree decompositions. None of these are very difficult, and I provide a sketch of a proof.

Claim. The treewidth of G is 1 if and only if G is a forest.

It's easy to tree decompose a tree; we can simply let the bags be edges of a tree and connect certain bags that represent incident edges. It's also easy to see that a cycle in G forces a cycle in its decomposition.

Claim. The treewidth of G is at most 2 if and only if G is a subgraph of a series/parallel graph.

We can tree decompose a series/parallel graph by grouping into bags the start, end, and middle of each series combination. One can also work backwards from a tree decompositon to obtain a series/parallel graph that must contain G.

Claim. No minor of G has treewidth exceeding the treewidth of G.

We can keep the same tree decomposition when deleting or contracting an edge of G, and the width can only decrease.

Claim. Any clique must be in its own bag.

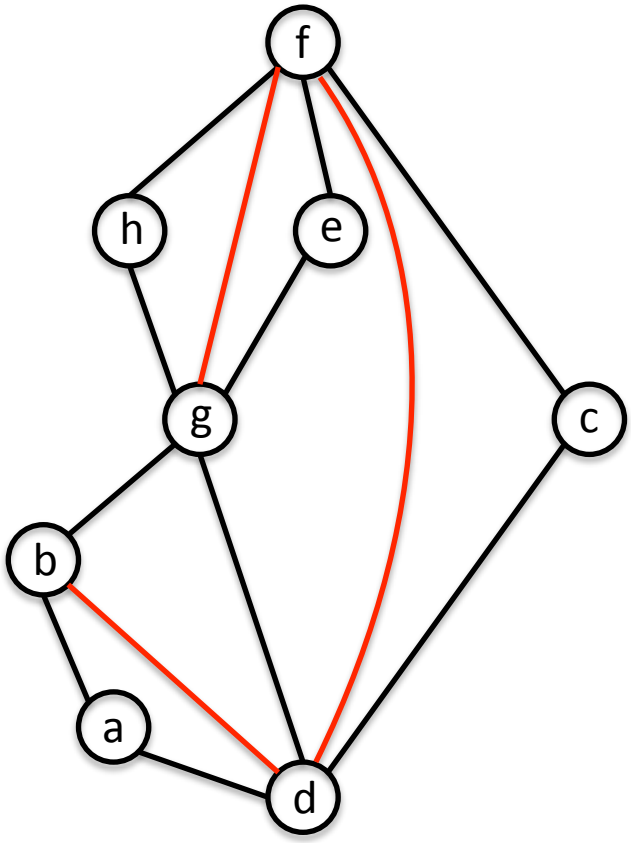
One consequence of this is that any graph that has K_n as a minor has tree width at least $n-1$. In particular, K_n itself has treewidth equal to $n-1$.

3 Chordal Graphs

As a general measure of "sparseness" of a graph, treewidth has many equivalent definitions. This section explores a few of these.

Definition. A graph G is *chordal* if every cycle on at least four vertices has at least one chord. We can add edges to a non-chordal graph to make it chordal; we call this process *triangulation*.

A tree decompositon provides a natural method of triangulation; namely, we can draw all edges between vertices in each bag that don't already exist. Here is an example of this process on the graph above:



Claim. A graph G has treewidth at most k if and only if there is a triangulation of G with no $k+2$ clique.

The forward implication is gaurunteed by the triangulation of bags. The following concept and result give a means for dealing with the converse:

Definition. A *perfect elimination ordering* is an ordering of the vertices of G such that the higher indexed neighbors of any vertex form a clique. That is, if we order the vertices $V_1, V_2, ..., V_n$, then for all $i, j, k \in [n]$ satisfying $i < j < k$ for which V_i is adjacent to both V_j and V_k, V_j is adjacent to V_k.

Theorem. *[FG] G is chordal if and only if there exists a perfect elimination ordering of the vertices of G.*

It follows that any graph with treewidth at most k has a ordering of its vertices such that each vertex has at most k higher indexed neighbors (and is thus k+1-colorable). The converse, however, is not true.

The following theorem provides another definition of treewidth:

Theorem. *[ST] The treewidth of a graph G is at most k if and only if $k+1$ cops can win the cops and robbers game on G.*

In the cops and robbers game, a robber attempts to escape capture by hiding out in a graph G. The cops may either station themselves at a vertices of G or "hover" above the graph, and at any time, may take off or land at a vertex. The robber is free to move the edges of G instantaneously, but is blocked by vertices occupied by cops. When a cop begins to "descend" onto a vertex, the robber is able to see this, and may choose to move wherever he desires before the cop lands. The cops win if they can trap the robber at a vertex, while the robber wins if he can escape capture indefinately.

The tree decomposition of a graph naturally leads to a winning strategy. First, they station themselves at the vertices in the root of the decomposition (in our example they would land at b, d, and g). Depending on where the robber hides, the cops change their positions to an appropriate subtree (if they can't immediately trap the robber at a, the cop at b moves to f). The cops can proceed to work down the tree until they reach a leaf and catch the robber.

The other direction is much more difficult, and is the substance of the theorem

4 Using Tree Decompositions on NP-Hard Problems

Many NP-Hard problems can be solved polynomial time on graphs with constant treewidth. First, we need to find a tree decomposition we can use. The following theorems gauruntee that we can do this:

Theorem. *[ACP] In a graph on n vertices with treewidth k, there is an algorithm that will return a tree decomposition with width k in time $n^{k+O(1)}$.*

When n is big, this can become extremely unweildy, so in some cases, the following is more useful:

Theorem. *[Bodlaender] Same as above, but in time $2^{\tilde{O}(k^3)}n$.*

Sometimes, it is advantageous to allow larger width in exchange for shorter running time:

Theorem. *[BDDFLP] Will return a tree of width $5k+4$ in time $2^{O(k)}n$.*

This last theorem removes the dependence in the running time from k:

Theorem. *[FHL] Will return a tree of width $O\left(k\sqrt{\log k}\right)$ in poly(n) time.*

While determining the exact treewidth of a graph is NP-Hard, there are algorithms that can determine whether the treewidth is at most k in time $O(n^k)$. One way to see do this is to attempt all robber hunting strategies for $k+1$ cops on the graph.

Theorem. *[Courcelle] Let F be a formula in extended monadic second order logic. On graphs with constant treewidth, F can be decided in polynomial time.*

The algorithm provided in this theorem is horribly inefficient for most problems, but it provides a guideline as to which NP-hard problems can be decided in polynomial time on a low treewidth graph G. Here are some examples of such problems:

- Deciding SAT, where G is the the primal graph of the SAT instance
- Deciding colorability of G
- Searching for a Hamiltonian cycle in G
- The traveling salesman problem restricted to G
- Finding the largest clique of G
- Finding the largest vertex disjoint path of G

Most algorithms work best when we add extra constraints to the tree before beginning computation.

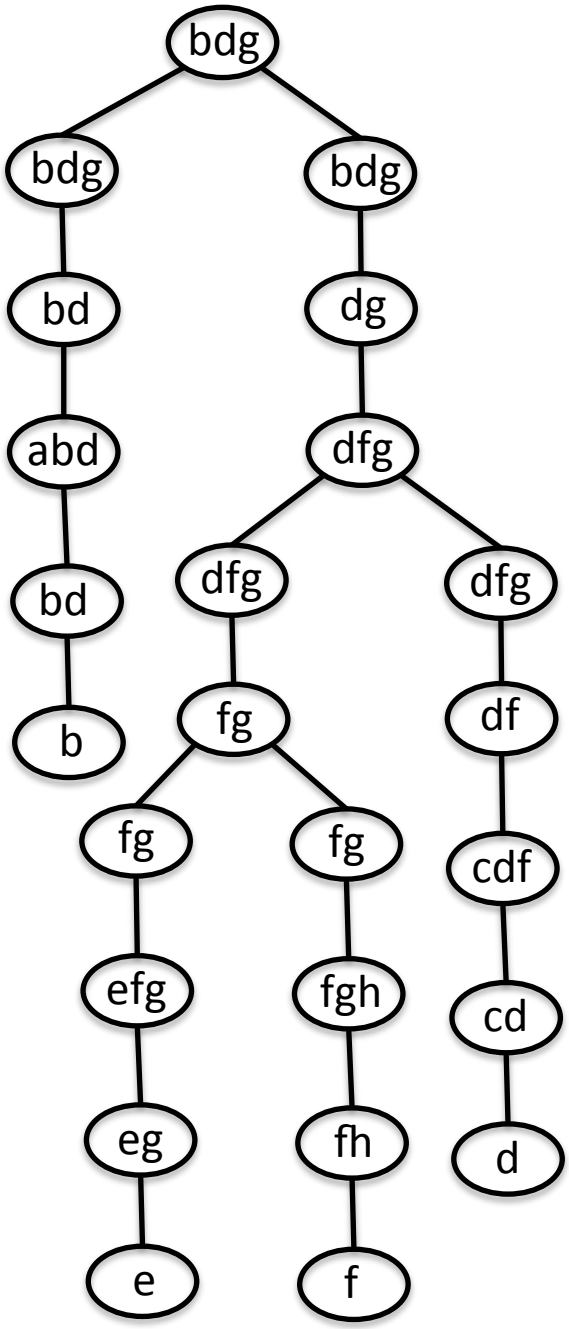
Definition. A tree decomposition is *smooth* if all bags have $k+1$ vertices, adjacent bags share k vertices.

For instance, the example tree decompositon in section 2 can be smoothed by deleting the three bags with two or fewer vertices. A smooth tree decomposition must necessarily have $n-k$ bags. In a sense, it is the most concise tree decomposition. However, for many purposes, the following simplification is more useful:

Definition. A *nice* tree decomposition is a rooted binary tree with four different types of nodes:

- Leaf nodes have no children and bag size 1.
- Introduce nodes have one child. The child has the same vertices as the parent with one deleted.
- Forget nodes have one child. The child has the same vertices as the parent with one added.
- Join nodes have two children, both identical to the parent.

A decomposition of this form can make it easy to recursively solve the problem with a dynamic programming algorithm. There is an algorithm that will convert a tree decomposition into a nice one with $O(nk)$ bags in time $O(nk^2)$. Below, I have made nice the example tree we used in section 2:



As an example of how nice trees can be used, I will describe a dynamic programming algorithm for determining colorability on a graph given a nice tree decomposition.

For a node X in the tree decomposition and a coloring c of the vertices contained in X, we define the boolean variable $E[X, c]$ to be true if and only if c can be extended to a valid coloring on all vertices in X or children of X. We want to know if $E[R, c]$ is true for any coloring c, where R is the root of the tree. We can compute $E[X, c]$ recursively as follows:

- If X is a leaf: All colorings are valid; i.e., $E[X, c]$ is true for all c.
- If X introduced the vertex v: For each coloring c, we must verify that it is valid among it's child and that the new vertex doesn't violate any new edge in X. Ex-

plicitly, $E[X, c]$ is false if v is the same color as one of its neighbors in X; otherwise, $E[X, c] = E[Y, c']$, where Y is X with v removed and c' is the coloring c restricted to Y.

- If X forgot the vertex v: We need to check if we can extend c to v; i.e., $E[X, c]$ is true if and only if there exists a coloring c' extending c to $Y = X \cup \{c\}$ such that $E[Y, c']$ is true.
- If X joined the nodes Y and Z: The coloring must be valid on both Y and Z; i.e., $E[X, c] = E[Y, c] \cap E[Z, c]$.

As we are often dealing with planar graphs, it would be fortunate if such graphs had low treewidth. Unfortunately, this is not the case. In particular, the graph of an $m \times m$ grid has treewidth m. (One way to see this is with the cops and robbers definition of treewidth: with m cops, the robber can ensure that he has acess to one full row and column whenever at least one cop is hovering.) Here are some results on treewidth on planar graphs:

Theorem. *[Eppstein] A planar graph with diameter d has treewidth at most $3d - 2$.*

The following two theorems are useful for approximation algorithms to NP-hard problems on planar graphs:

Theorem. *[Baker] Let G be planar, and let k be at least 2. Then there exists a polynomial time algorithm that partitions the edges of G into k sets such that when any one of the sets is deleted, the resulting graph has treewidth at most $O(k)$.*

There is also a good variant of this result for graphs with constant genus.

Theorem. *[DHM] Replace "deleted" with "contracted" above.*

References

[1] Stefan Arnborg, Derek G. Corneil, and Andzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, April 1987.

[2] Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, January 1994.

[3] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[4] Hans L. Bodlaender, Pål G. Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshanov, and Michał Pilipczuk. A $o(c^k n)$ 5-approximation algorithm for treewidth. *CoRR*, abs/1304.6321, April 2013.

[5] Bruno Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, February 2001.

[6] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Bojan Mohor. Approximation algorithms via contraction decomposition. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 278–287, 2007.

[7] D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3-4):275–291, January 2000.

[8] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.

[9] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

[10] P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–23, May 1993.