CS762: Graph-Theoretic Algorithms
Lecture 14: Bounded Treewidth
February 6, 2002

Scribe: Michael Laszlo

Abstract

We show that any graph of treewidth k is a partial k-tree, establishing one direction of an equivalence anticipated in the previous lecture. Given a fixed upper bound on treewidth, many NP-hard graph problems become tractable; here, we discuss the application of dynamic programming to a special tree-decomposition in order to solve a variety of such problems in linear time. We illustrate the approach with an algorithm for INDEPENDENT SET.

1 Introduction

1.1 Motivation

Treewidth is a relatively recent addition to the bestiary of graph theory, but one that has enjoyed widespread interest thanks to its theoretical and practical utility. In 1983, Robertson and Seymour introduced the idea of tree-decompositions [RS83]; from the tree-decompositions of a graph we derive the measure called treewidth, which has a constant bound in certain graph classes. Forests have treewidth 1, for example, and series-parallel graphs, which we are due to encounter in a forthcoming lecture, have treewidth 2 [Bod88]. We shall see that from a graph G of fixed treewidth, we can construct a special kind of tree-decomposition that allows generally exponential or polynomial problems to be solved on G in linear time.

Bodlaender summarizes a number of industrial applications involving treewidth [Bod93]. In the GATE LAYOUT PROBLEM of VLSI theory, we are given gates $G_1, \ldots, G_m$ (which can be imagined on a macro scale as the pins of a chip), nets $N_1, \ldots, N_n$ (akin to segments of copper track used to join such pins), and an $n \times m$ matrix M where m_{ij} specifies that N_i must be in contact with G_j. The problem of finding a gate permutation that minimizes the number of tracks necessary to make all connections is equivalent to the problem of determining minimum pathwidth, a close relative of treewidth [Moh90]. The property of bounded treewidth has been exploited in artificial intelligence: certain expert systems can be modelled by graphs of small treewidth, which lead to efficient algorithms for reasoning within the system [LS88]. Empirical findings in natural language processing suggest that dependency graphs of the syntactic relations in a sentence tend to have pathwidth 6 or less [KT90].

1.2 Outline

In the previous lecture, we stated our desire to use k-trees as an equivalent characterization of graphs of treewidth k. On this occasion, we prove that graphs of treewidth k are partial k-trees, leaving the reverse implication as an easy exercise for the reader. We are, as usual, interested in the complexity of generally intractable problems. When restricted to members of this graph class, are they subject to solution by algorithms of polynomial cost or better? It turns out that if we are given a tree-decomposition of the kind constructed in our proof, a wide range of NP-hard problems can be solved in linear time. We describe a general approach, using dynamic programming, that takes advantage of these narrow tree-decompositions. As a concrete example, we begin an explanation of how to find the maximum independent set in linear time.

2 Definitions

A ***k*-tree** is a chordal graph that has a perfect elimination order $(v_1, \ldots, v_k, v_{k+1}, \ldots, v_n)$ in which vertices $v_{k+1}, \ldots, v_n$ have exactly k predecessors each. A **partial *k*-tree** is a graph to which edges can be added (without disturbing the set of vertices) so as to form a k-tree.

A **tree-decomposition** of a graph $G = (V, E)$ is given by a tree $T = (I, F)$ and a family of subsets of V denoted X_i, corresponding to each vertex $i \in I$, where

1. $\bigcup_{i \in I} X_i = V$, that is, every vertex of G is assigned to some X_i;
2. for every edge $(v, w) \in E$, there is some X_i containing both v and w;
3. for each vertex $v \in V$, the subgraph of T induced by $\{i \in I \mid v \in X_i\}$ is connected.

We shall henceforth refer to the vertices $i \in I$ of T as *nodes* in order to distinguish them from the vertices $v \in V$ of G.

Treewidth is defined for tree-decompositions and for graphs. The treewidth of a tree-decomposition is given by $\max_{i \in I} |X_i| - 1$. Since a tree-decomposition can always be obtained by assigning all vertices of G to a single subset X_1, the maximum treewidth over all tree-decompositions of a graph is $|V| - 1$. It is not so easy to ascertain the minimum treewidth over all tree-decompositions of a graph; this is what we call the treewidth of the graph.

3 Treewidth *k* implies a partial *k*-tree

Our broad aim is to show that the following is true.

Theorem 1 A graph G has treewidth k if and only if G is a partial k-tree.

We shall prove the more difficult of the two implications in this equivalence, leaving the other as a diversion for the reader.

3.1 Construction in three stages

Claim 1 If a graph G has treewidth k, then G is a partial k-tree.

Proof: By the definition of treewidth, G has at least one tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ where the greatest size of any subset X_i is $k + 1$. Let us take such a tree-decomposition.

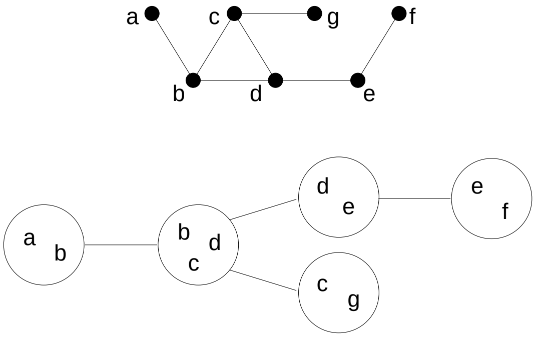

Figure 1: A graph and its tree-decomposition, both of treewidth 2.

3.1.1 Attaining uniform node size

We now form subsets $X'_1, \ldots, X'_{|I|}$ by assigning a sufficient number of additional vertices to each of the subsets $X_1, \ldots, X_{|I|}$ to ensure that $|X'_i| = k+1$ for all $i \in I$. Adding vertices to the X_i cannot possibly break the first two properties of a tree-decomposition: every vertex is already assigned to some subset, and the endpoints of every edge can already be found together in some subset. Care must be taken only to preserve the third property, requiring that all occurrences of a vertex in T form a contiguous set of nodes. To do so, we supplement each X_i only with vertices found in X'_j such that $j \in \mathsf{Adj}(i)$. This is readily accomplished by augmenting the subsets during a traversal of T, either breadth-first or depth-first, starting from a root corresponding to a subset already of size $k+1$.

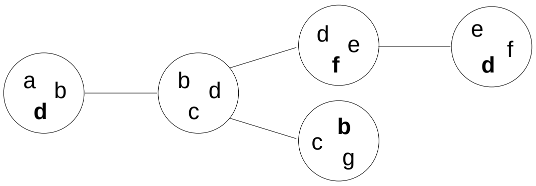

Figure 2: After augmentation of subsets.

3.1.2 Attaining uniform node difference

We add a sufficient number of nodes to I, forming I', and make $k+1$ vertex assignments to each new corresponding subset $X'_{|I|+1}, \ldots, X'_{|I'|}$, to bring about a state of affairs where $|X'_i - X'_j| \leq 1$ for all $(i,j) \in F'$. In other words, neighboring nodes in $T' = (I', F')$ must always designate subsets that are equal in size and that differ by at most one vertex. Again, this can be done by breadth-first or depth-first traversal. Before descending to a child j of node i, we consider whether $|X'_i - X'_j| > 1$. If so, we pick vertices v, w such that $((v \in X'_i) \wedge (v \notin X'_j)) \wedge ((w \notin X'_i) \wedge (w \in X'_j))$; augment I with a node i' corresponding to the new subset $X'_{i'} = (X'_i - \{v\}) \cup \{w\}$; add edges $(i, i'), (i', j)$ to F; and remove edge (i, j).

The introduction of this intermediate node not only preserves the properties of tree-decomposition, but narrows the gap between X'_i and X'_j, since $|X'_i - X'_{i'}| = 1$ and $|X'_{i'} - X'_j| = |X'_i - X'_j| - 1$. If a new child i' was indeed inserted, we descend from i to i'; otherwise, if no such maneuver was necessary, we descend directly to j. Upon completion of the traversal, we have a new tree-decomposition

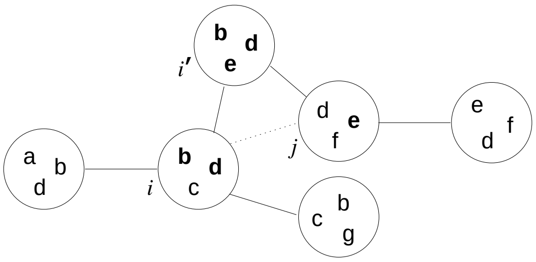

Figure 3: Bridging the gap one vertex at a time.

$(\{X'_1, \ldots, X'_{|I'|}\}, T' = (I', F'))$ such that no pair of subsets (X'_i, X'_j) corresponding to $(i,j) \in F'$ can differ by more than one vertex.

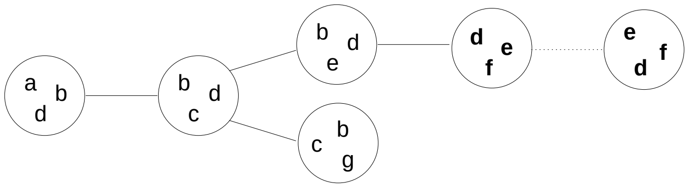

Figure 4: Contracting edges whose endpoints correspond to equal subsets.

It is possible that we had $X'_i = X'_j$ for adjacent i, j at the outset of our traversal, a circumstance unaffected by the ensuing transformation. We locate all such cases and contract the edges (i, j), so that we are finally left with a tree-decomposition where the difference between every pair of subsets (X'_i, X'_j) corresponding to adjacent nodes of T' is exactly one: $(\forall (i,j) \in I')(|X'_i - X'_j| = 1)$.

3.2 The special tree-decomposition implies a k-tree

Consider the fullest possible graph implied by our final tree-decomposition, *i.e.*, the graph $G' = (V, E')$ whose vertex set is that of G but whose edges are $E' = \{(v, w) \mid (\exists i \in I')(v, w \in X'_i)\}$, noting for future reference that $E' \supseteq E$. It is not immediately clear that this implied graph must be chordal, since our construction allows us to add new edges between members of V and thus, perhaps, to form new cycles $C_{l>3}$. However, we propose that G' is not merely chordal but a k-tree.

Lemma 1 The fullest graph $G = (V, E)$ having tree-decomposition $(\{X_1, \ldots, X_{|I|}\}, T = (I, F))$, with properties $(\forall i \in I)(|X_i| = k+1)$ and $(\forall (i,j) \in I)(|X_i - X_j| = 1)$, is a k-tree.

Proof:

We proceed by induction on the number of nodes in T.

Base

If $|I| = 1$, then the sole subset X_1, necessarily of size $k+1$, contains all vertices of G, and E must include every possible edge. G is therefore the complete graph K_{k+1}, for which any vertex ordering $(v_1, \ldots, v_{k+1})$ is a perfect elimination order such that v_{k+1} has k predecessors.

Step

Assume that for fixed $n \geq 2$, our proposition is true in all cases where the given tree-decomposition satisfies $|I| < n$. Consider a case where $|I| = n$.

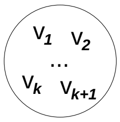

Figure 5: A tree-decomposition of treewidth k, with $|I| = 1$.

With T rooted at an arbitrary node, pick a leaf j with parent i. In our special tree-decomposition, $|X_i - X_j| = 1$, so let v_n be the vertex such that $X_j - X_i = \{v_n\}$. Since v_n does not appear in the subset X_i corresponding to the only neighbor i of j, it can appear nowhere besides X_j. Now consider the tree-decomposition $(\{X_1, \ldots, X_{|I|}\} - \{X_j\}, T' = (I' = I - \{j\}, F'))$ that results from deleting node j and the corresponding subset X_j. Compared to G, the fullest graph G' implied by this tree-decomposition lacks v_n and all associated edges. Since the subsets apart from X_j have not been disturbed, neither of the special properties required by our proposition can have been broken, and $|I'| = n - 1$ implies that the fullest graph G' having this tree-decomposition is a k-tree.

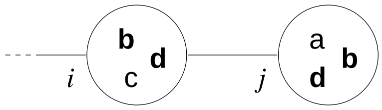

Figure 6: If we delete node j, only vertex a is lost.

Then, by definition, G' has a perfect elimination order $(v_1, \ldots, v_{n-1})$ with $|\mathsf{Pred}(v_i)| = k$ for all $i > k$. We can use this as the basis of a particular kind of perfect elimination order for G itself. The order $(v_1, \ldots, v_{n-1}, v_n)$ accounts for every vertex of G, and the predecessors of v_1 through v_{n-1} remain the same as in G'; the vertices $X_j - \{v_n\}$ already appear together in X_i. As for v_n, it occurs only in the leaf j and is therefore adjacent to precisely the k other vertices assigned to subset X_j. Thus, $|\mathsf{Pred}(v_n)| = k$, and, indeed, $(\forall v \in V)(|\mathsf{Pred}(v)| = k)$ in our perfect elimination order; G is therefore a k-tree. □

From the result just shown, we know that the tree-decomposition T' earlier constructed from a graph G is a k-tree. But the graph G' of which T' is a tree decomposition has exactly the vertices of G, and must include every edge found in G. Since G can be transformed into a k-tree by addition of edges, it is a partial k-tree. And since T' can be constructed from any graph of treewidth k, all such graphs are partial k-trees. □

4 Linear-time algorithms for graphs of bounded treewidth

Tree-decompositions obtained by the foregoing construction lend themselves to efficient algorithms for a variety of problems that are, in general, NP-hard. Before seeing one such algorithm, let us introduce some terminology in the context of a graph $G = (V, E)$ with treewidth k and tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ where $|X_i - X_j| = 1$ for all $(i, j) \in F$. It is necessarily true that $|X_i| = k + 1$ for all $i \in I$. An arbitrary node r is employed as the root of T.

For a given node i, let $Y_i = \{v \in X_j \mid (j = i) \vee (j \in \mathsf{descendants}(i))\}$. In other words, Y_i is the set of all vertices found in nodes of the subtree of T rooted at i. Let $G[Y_i]$ denote the subgraph of G induced by Y_i. In the arguments to follow, we shall use *configuration* to mean some selection or arrangement of vertices—and possibly of edges, depending on the problem at hand—made in some portion of a graph in an effort to solve a problem.

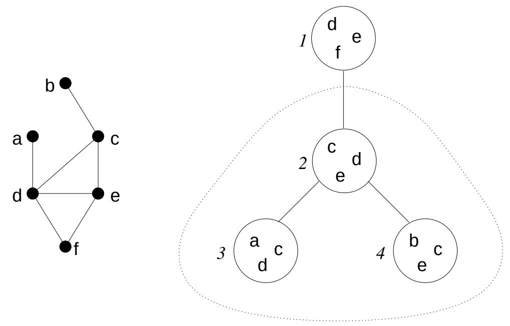

Figure 7: $Y_2 = \{a, b, c, d, e\}$.

4.1 The general approach

Consider a node $j \in I$ and its parent i. Observe that if a vertex $v \in Y_j$ has been assigned to some X_m where m is not a descendant of i, then, by the third property of tree-decompositions, v must have been assigned to X_i. Thus, of all the vertices in Y_j, only those that are also in X_i can affect the configuration of the graph lying beyond $G[Y_i]$. And of all the vertices lying outside Y_j, only those that are also in X_j can affect the configuration of $G[Y_j]$. One might say that the choice of configurations for G narrows to a bottleneck at each pair of adjacent nodes. In the search for a globally optimal configuration, the traffic between $G[Y_i]$ and $G - G[Y_j]$ is limited to the corridor between X_i and X_j.

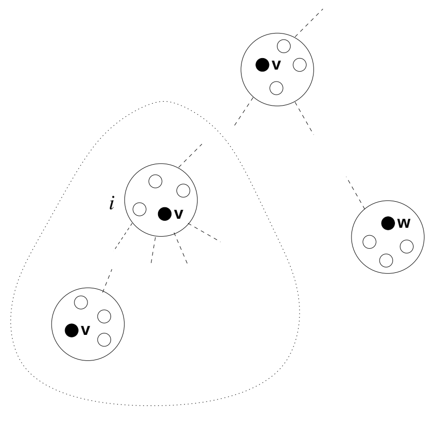

Figure 8: If v and w occur outside Y_i, then $(w \notin X_i) \Rightarrow (w \notin Y_i)$ and $(v \in Y_i) \Rightarrow (v \in X_i)$.

This happy circumstance inspires us to perform dynamic programming on the tree-decomposition of any graph G with bounded treewidth. Typically, in solving a problem on G, we will want to begin at the lowest level of T and ascend to its root. Upon arriving at a node i, we will have computed tables of optimal solutions for the $G[Y_j]$ corresponding to each of its children j. Then, for every possible configuration of X_i, we look up the best solution in $G[Y_j]$ consistent with that configuration. The number of configurations may be exponential relative to $k + 1$, but with fixed k, the cost of the procedure at each node is bounded by a constant, if perhaps a rather large one. Thus, the total cost is proportional to the number of nodes in the tree-decomposition.

4.2 Application to INDEPENDENT SET

To clarify the idea, let us apply it to the problem of finding the maximum independent set in a graph whose tree-decomposition conforms to the above description. (In the general case, this problem is NP-complete.) For a subset $Z \subseteq X_i$ of the vertices corresponding to node i, let $\mathsf{is}_i(Z)$ be the size of the greatest independent set $W \subseteq G[Y_i]$ that includes every vertex in Z but no other member of X_i. To put it another way, $W \cap X_i = Z$. If there is no such W, then set $\mathsf{is}_i(Z) = -\infty$.

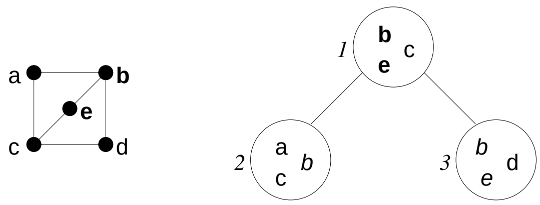

Figure 9: With $Z = \{b, e\}$, there is no independent set in $G[Y_1]$.

To arrive at the size of the maximum independent set in G, we proceed in the bottom-up fashion suggested above, computing $\mathsf{is}_i(Z)$ for each of the $2^{|X_i|} = 2^{k+1}$ possible values of Z at each node i. At the leaves, we have simply

$$\mathsf{is}_i(Z) \quad = \quad \begin{cases} -\infty \text{ if } Z \text{ is not an independent set of vertices} \\ |Z| \text{ otherwise.} \end{cases}$$

Notice that if V is non-empty, then $(\exists Z \in X_i)(|Z| = 1)$, so that we always have $\mathsf{is}_i(Z) \geq 1$. The value of $-\infty$ ensures that non-independent sets in $G[X_i]$ are barred from future use. The next lecture will treat the question of how to compute $\mathsf{is}_i(Z)$ if i is an internal node.

5 Further Study

The perceptive reader will have remarked that when we apply our dynamic-programming approach to the maximization problem of INDEPENDENT SET, it is an especially straightforward task to enumerate the configurations of $G[X_i]$ at each node i of the tree-decomposition. We have only to choose a subset Z of the vertices in X_i, and all edges with endpoints in Z are understood. The proceedings become stickier if we tackle a problem such as HAMILTONIAN CYCLE, where each configuration entails not merely a choice of vertices in $G[X_i]$, but a selection among its edges. Although our approach is clear in principle, its application to specific graph problems, especially NP-complete ones such as HAMILTONIAN CYCLE and STEINER TREE, invites investigation.

Because the treewidth of a graph is a valuable piece of information from the perspective of many important problems, methods to compute treewidth are of interest. The problem of determining whether an unrestricted graph has treewidth no greater than variable k is NP-complete [ACP87], but certain graph classes are amenable to polynomial-time algorithms. For classes with constant treewidth, such as forests, SP-graphs, and outerplanar graphs [Bod93], the question is immediately answered. Among the graph classes in which treewidth can be found in polynomial time are chordal graphs (whose treewidth is one less than the maximum clique size), permutation graphs [BKK92], and circle graphs [Klo93].

In our description of the linear-time approach using dynamic programming, we assumed that we had at our disposal a special tree-decomposition corresponding to the problem instance. But if our graph algorithm is to take linear time from beginning to end, we cannot ignore the cost of constructing such a tree-decomposition. Indeed, until 1992, the best known construction took time in $O(n \log n)$ [Ree92]. Those inclined to worry about the matter will want to peruse Bodlaender's linear-time algorithm, which results in a tree-decomposition whose size is proportional to that of the original graph [Bod92].

References

[ACP87] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[APS90] S. Arnborg, A. Proskurowski, and D. Seese. Monadic second order logic, tree automata and forbidden minors. *Proceedings of the 4th Workshop on Computer Science Logic, CSL90*, pages 1–16, Springer Verlag, Lecture Notes in Computer Science, vol. 477, 1990.

[BKK92] H. L. Bodlaender, T. Kloks, and D. Kratsch. Treewidth and pathwidth of permutation graphs. Technical Report RUU-CS-92-30, Department of Computer Science, Utrecht University, The Netherlands. 1992.

[Bod88] H. L. Bodlaender. Some classes of graphs with bounded treewidth. *Bulletin of the EATCS*, 36:116–126, 1988.

[Bod92] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. Technical Report RUU-CS-92-27, Department of Computer Science, Utrecht University, The Netherlands. 1992.

[Bod93] H. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[Klo93] T. Kloks. Treewidth of circle graphs. Manuscript. 1993.

[KT90] A. Kornai and Z. Tuza. Narrowness, pathwidth, and their application in natural language processing. Manuscript. Submitted *Disc. Appl. Math.*, 1990.

[LS88] S. J. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *The Journal of the Royal Statistical Society. Series B (Methodological)*, 50:157–224, 1988.

[Moh90] R. H. Mohring. Graph problems related to gate matrix layout and PLA folding. *Computational Graph Theory, Computing Supplement 7*, pages 17–51, 1990.

[Ree92] B. Reed. Finding approximate separators and computing tree-width quickly. *Proceedings of the 24th Annual Symposium on Theory of Computing*, pages 221–228, 1992.

[RS83] N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *J. Comb. Theory Series B*, 35:39–61, 1983.