Dynamic programming, treewidth and graph minors

Uriel Feige
Department of Computer Science and Applied Mathematics
The Weizman Institute
Rehovot 76100, Israel
uriel.feige@weizmann.ac.il

1 Introduction

We shall touch upon the theory of *Graph Minors* by Robertson and Seymour. This theory gives a very general condition under which a graph problem has a polynomial time algorithm (though the algorithms that come out of the theory are often not practical). We shall review a small part of this theory, and illustrate how dynamic programming can be used to solve some NP-hard problems on restricted classes of graphs (those of *bounded treewidth*).

2 Dynamic programming on trees

As a motivating example to the forthcoming notion of *treewidth*, consider the maximum weight independent set problem on trees. The input is a tree $G(V, E)$ and a weight function $w : V \longrightarrow R$. (For simplicity in performing the arithmetic operations, think of the weights as nonnegative integers.) We wish to find an independent set (a set of vertices no two of which are adjacent) of maximum weight. This can be solved in polynomial time using dynamic programming as follows.

Fix an arbitrary vertex r as the root of the tree, and orient all edges away from r. The subtree rooted at v, denoted by $T(v)$, includes v and all vertices reachable from v under this orientation of edges. (Hence $T(r) = G$.) The *children* of v, denoted by $C(v)$ are all those vertices u with a directed edge (v, u). Leaves of the tree have no children (with a possible exception if we chose r to be one of the leaves of G – then it does have children).

Let $W(T(v))$ denote the maximum weight of an independent set of $T(v)$. We want to compute $W(T(r))$. Let $W^+(T(v))$ denote the maximum weight of an independent set of $T(v)$ that contains v, and let $W^-(T(v))$ denote the maximum weight of an independent set of $T(v)$ that does not contain v. Then we have

$$W^+(T(v)) = w(v) + \sum_{u \in C(v)} W^-(T(u))$$

and

$$W^-(T(v)) = \sum_{u \in C(v)} \max[W^-(T(u)), W^+(T(u))]$$

The dynamic program now works by repeating the following procedure:

1. Find a vertex v such that for all of its children u we have already computed $W^-(T(u))$ and $W^+(T(u))$.

2. Compute $W^-(T(v))$ and $W^+(T(v))$ as described above.

Eventually, output $\max[W^-(T(r)), W^+(T(r))]$. Note that as long as there are unvisited vertices, these vertices form a tree and hence have a leaf, and this leaf can be chosen in step 1 of the above procedure. Hence the algorithm never gets stuck.

Many NP-hard graph problems are polynomially solvable on trees. (There are exceptions. For example, bandwidth is NP-hard on trees.) We shall show how the class of graphs for which the above dynamic programming approach works (namely, trees) can be significantly generalized.

3 Tree decomposition

We now introduce the notion of a tree decomposition of a graph.

Definition 1 *A* tree decomposition *of a graph $G(V, E)$ is a tree T, where*

1. *Each node i of T is labeled by a subset $B_i \subset V$ of vertices of G, referred to as a* bag.

2. *Each edge of G is in a subgraph induced by at least one of the B_i,*

3. *For every three nodes $i, j, k \in T$ with j lying on the path from i to k in T, $B_i \bigcap B_k \subset B_j$.*

Remarks:

1. The condition $B_i \bigcap B_k \subset B_j$ is equivalent to the condition that for every $v \in V$, the set of nodes in T whose bag contains v is connected in T.

2. We shall assume that no two nodes $i, j \in T$ have $B_i \subset B_j$. (Otherwise, merge i and j if they are adjacent, or i and its nearest neighbor on the path to j if they are not.)

3. A graph may have several different tree decompositions. Similarly, the same tree decomposition (as we defined it) can correspond to several different graphs. We can define B_i to be not just a set of vertices but also the set of induced edges, and then a tree decomposition uniquely determines a graph.

Every graph has a trivial tree decomposition for which T has one node including all of V. This partly motivates the following definition:

Definition 2 *The* treewidth *of G is the minimum integer p such that there exists a tree decomposition G with all subsets of cardinality at most $p + 1$.*

Observe that the treewidth of a graph is equal to the maximum of the treewidths of its connected components. Another simple observation is the following:

Lemma 1 *Every graph with treewidth p has a vertex of degree at most p.*

Proof: For graph G, consider a tree decomposition T with width p. Consider a leaf l of T. Its bag includes a vertex v that is not in the bag of its neighbor in T, and hence not in any other bag either. As $|B_l| \leq p+1$, it follows that the degree of v in G is at most p. □

We can exactly characterize those graphs having treewidth $n-1$.

Proposition 2 *An n-vertex graph has treewidth $n-1$ iff it is a clique.*

Proof: If the graph is not a clique, then some edge (u, v) is missing. In this case, we can take $B_1 = V \setminus \{u\}$ and $B_2 = V \setminus \{v\}$ as a two node tree decomposition of treewidth $n-2$.

Conversely, if G is connected and has a tree decomposition with treewidth below $n-1$, then it has a vertex of degree at most $n-2$, and cannot be a clique. □

We shall be interested in graphs with small treewidth. Clearly, the only graphs with treewidth 0 are independent sets.

Proposition 3 *G has treewidth at most 1 iff it is a forest.*

Proof: Consider an arbitrary tree G. We build a tree decomposition for it with treewidth 1 as follows. Pick a root r in G, and orient the edges away from r. Hence each vertex except for r has one parent vertex. The tree decomposition T of G will have a topology identical to G, where each node of T corresponds to a vertex of G. The bag of the node contains the corresponding vertex in G and its parent vertex (if there is one). Hence all bags have two vertices, except for one bag that contains only one vertex r. Every edge is in some bag, and for every vertex v in G, the nodes of T that contain it are connected (they are the node corresponding to the vertex v and to the direct children of v). Consequently, T is a tree decomposition of G.

Conversely, assume that G has treewidth 1. Then it has a vertex of degree at most 1. Removing this vertex from G, the treewidth cannot increase. Hence we can recursively remove vertices of degree at most 1, until we exhaust all vertices in G. This implies that G is a forest. □

Definition 3 *A* series-parallel *graph is a graph obtained from an independent set using the following operations:*

1. *Add a new vertex and connect it to an existing vertex by an edge.*
2. *Add a self loop.*
3. *Add an edge in parallel to an existing edge.*
4. *Subdivide an edge by creating a vertex in the middle.*

Proposition 4 *G has treewidth at most 2 iff it is a series-parallel graph.*

Proof: Let G be a series-parallel graph. We build a tree decomposition for it inductively, without ever exceeding treewidth 2. We do this by following the inductive construction of G. The most involved step is step 4 in the construction of series parallel graphs. Suppose edge (u, v) was subdivided, introducing a vertex w. Prior to the subdivision, the tree decomposition included a node j such that $u, v \in B_j$. Create a new node labeled by $\{u, v, w\}$ and connect it to j.

Conversely, assume a tree decomposition T with treewidth at most 2. We show that this corresponds to a series-parallel graph. We do this by removing a node from the tree decomposition, show that this corresponds to removing a vertex from the graph G, and then show that G can be obtained by adding back the vertex using one of the rules of Definition 3. Again we illustrate only one case. Take a leaf ℓ of T. Suppose that it is labeled by three vertices u, v, w (it cannot be labeled by more than three vertices). Then w.l.o.g., the neighbor node of ℓ in T does not contain w. Hence the only possible neighboring vertices of w in G are u and v. Suppose that both u and v are neighbors of w in G. (If only one of them is, remove w, and it can later be rejoined using operation 1 of Definition 3.) Remove w from G and add the edge (u, v) to G, thus obtaining a graph G' (possibly with parallel edges). Remove w from the leaf ℓ, thus obtaining a tree decomposition T' of G'. G' has one less vertex than G and has treewidth at most 2, and hence by the induction hypothesis G' is series-parallel. G can be obtained from G' by operation 4 of Definition 3, and so G is series-parallel as well. □

The notion of treewidth connects us to the notion of graph minors.

Definition 4 *H is a* minor *of G if it can be obtained from G by a sequence of operations of edge deletions, vertex deletions, and edge* contractions *(merging endpoints together).*

A planar graph is a graph that can be drawn in the plane with no two edges crossing. Kuratowski showed that a graph is non-planar iff it contains either K_5 (a clique on five vertices) or $K_{3,3}$ (a complete bipartite graph on six vertices) as a minor. We observe that a graph is not a forest iff it contains K_3 as a minor. It is known (we shall not prove it) that a graph is not series-parallel iff it contains K_4 as a minor. (In particular, K_4 is a planar graph that is not series-parallel.) As forests have treewidth at most 1 and series-parallel graphs have treewidth at most 2, one may hope that graphs not containing K_5 as minors have treewidth at most 3, especially as K_5 itself has treewidth 4. However, this is false.

Proposition 5 *The treewidth of planar-graphs can be as high as $\Omega(\sqrt{n})$.*

We shall use the following important lemma in the proof of Proposition 5.

Lemma 6 *Let $G(V, E)$ be a graph of treewidth p. Suppose that the vertices $v \in V$ have nonnegative weights $w(v)$ with $\sum_{v \in V} w(v) = 1$. Then there is a set S of at most $p + 1$* separating *vertices whose removal separates G into parts $S_1, S_2, \ldots$ (possibly only one part) with the following properties:*

1. *Each part has total weight no more than $1/2$.*
2. *For any two different parts (say, S_i and S_j) , all paths from S_i to S_j must go through S.*

Proof: Let T be a tree decomposition of G with treewidth p. Observe that the removal of a non-leaf tree node decomposes the tree into subtrees with no paths connecting different subtrees. Similarly, the removal of the vertices in the corresponding bag disconnects G into components (where a component contains the contents of the bags in a subtree, but without the vertices from the separator bag) with no paths between them.

If T contains a bag of total weight at least $1/2$, this bag can serve as the separator. If every bag has weight less than $1/2$, pick an arbitrary root r in T and orient edges away from r. For a subtree of T rooted at node v, let $W(v)$ be the sum of the weights of the distinct vertices in the bags of the subtree rooted at v. Observe that $W(r) = 1$ and the weight of every leaf is less than $1/2$. Let j be a tree node with $W(j) \geq 1/2$ and $W(u) < 1/2$ for all children u of j. Let $S = B_j$. Then $|S| \leq p + 1$. Now make j the root of T. For child u of j in T, define $S(u)$ to be the vertices of G appearing in the labels of the subtree rooted at u but not in S. Then the $S(u)$ are disjoint, and any path connecting a vertex from $S(u)$ to a vertex in $S(u')$, where $u \neq u'$ and both u and u' are children of j, must go through S. Let $W'(u) = \sum_{v \in S(u)} w(v)$. Then $W'(u) < 1/2$. □

Remark. in Lemma 6 we may instead require that S separates G into two parts S_1 and S_2 (one of which may be empty), each of weight at most $2/3$. If for some u, $W'(u) \geq 1/3$, then take $S_1 = S(u)$ and $S_2 = V \setminus S(u) \setminus S$. If $W'(u) < 1/3$ for all u, then we can combine several $S(u)$ to get a set S_1 of weight between $1/3$ and $2/3$.

We now proof Proposition 5.

Proof: Consider a $\sqrt{n}$ by $\sqrt{n}$ mesh, and let its treewidth be p. Give each vertex weight $1/n$. By Lemma 6 there is a set S of $p + 1$ vertices whose removal breaks the mesh into components $S_1, S_2, \ldots$, none of which contains more than $n/2$ vertices. Assume for the sake of contradiction that $p + 1 < \sqrt{n}/2$. Then more than $\sqrt{n}/2$ columns of the mesh are *free* – they do not contain any separator vertex. Likewise, more than $\sqrt{n}/2$ rows of the mesh are free. All the free columns and rows of the mesh are in the same connected component, giving a component larger than $\sqrt{n}\frac{\sqrt{n}}{2} = \frac{n}{2}$, contradicting the assumption that $p+1 < n/2$. Hence the treewidth of the mesh is at least $\sqrt{n}/2 - 1$. □

Remark. In fact, the treewidth of a $\sqrt{n}$ by $\sqrt{n}$ mesh is $\sqrt{n}$. We shall only show the upper bound, but will not prove the tight lower bound. To get a tree decomposition in which every bag is of size $\sqrt{n} + 1$, consider the following collection of bags $B_{i,j}$ with $1 \leq i < \sqrt{n}$ and $1 \leq j \leq \sqrt{n}$. Bag $B_{i,j}$ contains the first $\sqrt{n} + 1 - j$ vertices of row i and the last j vertices of row $i + 1$. Arranging the bags in lexicographic order gives a tree decomposition of the mesh (where the tree is a path).

Treewidth is connected to planarity via the following theorem of Robertson and Seymour, whose proof is beyond the scope of the course.

Theorem 7 *Let H be a planar graph. If G has no H-minor, then the treewidth of G is bounded by some function of H, independent of G.*

Observe that indeed H must be planar in the above theorem, as for every nonplanar H, planar grids form a graph family with no H-minor and unbounded treewidth.

4 Coloring

Recall that a legal k-coloring of a graph colors the vertices with k colors such that adjacent vertices receive different colors. The *chromatic number* of a graph is the minimum k that allows a legal k-coloring. Computing the chromatic number is NP-hard.

Proposition 8 *Every graph with treewidth p has chromatic number at most $p+1$.*

Proof: The proof is via the method of *inductive coloring*. We have seen that a graph $G(V, E)$ of treewidth p has a vertex v with degree at most p. If the rest of the graph is legally colored by $p+1$ colors then we can also $(p+1)$-color G by assigning to v one of the colors not assigned to its neighbors. As the subgraph induced on $V \setminus \{v\}$ also has treewidth at most p, the proof follows by induction. □

Though planar graphs have unbounded treewidth, the method of inductive coloring does apply to them. As every planar graph has a vertex of degree at most 5, planar graphs are 6-colorable. By the fact that planar graphs do not contain K_5, it follows that they are 5-colorable. Consider the five neighbors of a vertex v of degree five. Two of them are not adjacent, and can be contracted to v while preserving planarity, giving a graph G'. A legal 5-coloring of G' can be extended to a 5-coloring of G because now two of the neighbors of v have the same color.

The famous *4-color theorem* says that every planar graph is 4-colorable. Current proofs of this theorem are complicated and use extensive computer assisted case analysis.

Hadwiger conjectured that every graph that does not contain K_{p+1} as a minor is legally p-colorable. This conjecture is still open. If true, it would generalize the 4-color theorem. A weaker form of the conjecture is known to hold. (Namely, every graph that does not contain K_p as a minor is legally $f(p)$-colorable, where $f(p)$ is a slowly growing function of p. In particular, $f(p) \leq O(p\sqrt{\log p})$. The proof is based on the fact that inductive coloring of G fails only if there is a subgraph of high minimal degree, but then this subgraph must have a large clique as a minor.)

Deciding whether a planar graph is 3-colorable is NP-hard. On the other hand, for graphs of bounded treewidth, their chromatic number can be determined in polynomial time. This can be done via dynamic programming on the tree decomposition of the graph. We use the following theorem, whose proof is deferred to Section 4.1:

Theorem 9 *For graphs that have bounded treewidth, a corresponding tree decomposition can be found in polynomial (in fact, linear) time.*

Given a tree decomposition T of treewidth p for k-colorable graph G, we legally color G with k colors as follows. Pick an arbitrary root r for T and orient edges away from r. For node j in T, let $T(j)$ denote the subtree of T rooted at j, and let $V(j)$ denote the set of vertices of G in bags of $T(j)$. We use the fact that $|B_j| \leq p+1$. For each node $j \in T$, we maintain an indicator vector c_j of length k^{p+1}. Each entry corresponds to one possible (not necessarily legal) k-coloring of the vertices of B_j. This entry is 1 if the k-coloring can be completed to a legal k-coloring of the subgraph induced on $V(j)$, and 0 otherwise. Note that the vector c_j can be completely determined from the vectors c_u of all the children u of j, and from B_j, because for a coloring of $V(j)$ to be legal, it suffices that it is legal on B_j

and on $V(u)$ for each child u of j *separately* – there is no interaction between the children of j except through edges present in B_j. Hence using dynamic programming we can eventually compute c_r, and if c_r has a nonzero entry then G is k-colorable.

The running time of the algorithm can be estimated as follows. Every tree decomposition T of graph G has at most n nodes. (A leaf l of T has in its label a vertex v of G not appearing anywhere else in T. Removing v from G, the tree T is updated by removing at most the leaf l. Continue recursively.) Hence T has at most n edges. At each edge (i, j) we look at most k^{p+1} times (once for each coloring of its parent node i). Each time we look at an edge we at most scan the whole indicator vector c_j of length k^{p+1} of the child node j. As $k \leq p + 1$, the running time is proportional to $np^{O(p)}$ (assuming that a tree decomposition is given). Note that for fixed p, this running time is linear (assuming fixed-time random access to the adjacency matrix of G).

Thus we see that dynamic programming algorithms that work on trees often extend to graphs of bounded treewidth. To appreciate this extension, observe that graphs with bounded treewidth need not "look like" trees (e.g., series-parallel graphs).

4.1 Finding a tree decomposition

We provide here part of the proof of Theorem 9. We may assume that $p \geq 2$ (as $p = 1$ corresponds to trees). For a graph on n vertices and treewidth p we show how to find a tree decomposition with treewidth at most $8p$ in time $n^{O(1)}2^{O(p)}$. For constant p (or even p logarithmic in n), this running time is polynomial in n. Given a tree decomposition of treewidth $8p$, finding a tree decomposition of optimal treewidth p can be done using dynamic programming, though details of this are omitted here.

Given a graph G, the construction of the tree decomposition is inductive. At an intermediate stage, we already have some bags connected as a partial tree decomposition (for part of G), and the following properties hold:

1. The partial tree decomposition is a legal tree decomposition for the subgraph induced on all vertices that are already in bags.

2. Each bag has at most $8p$ vertices.

3. The vertices of G not yet in bags form several connected components in G (possibly only 1). For each such connected component C there is an associated bag B (that depends on C) that contains all neighbors of vertices in C (except those already in C). Note: a vertex might be in several bags, so a neighbor of C might be in several bags, but at least one copy of it must be in B.

4. For each such connected component C, the number of neighbors of C in the corresponding B is at most $6p$.

To initiate the inductive construction, take any $6p$ vertices and let them form a bag. The above properties surely hold.

Let us now consider an inductive step. We need to add one more bag to the partial tree decomposition, where this bag contains at least one more vertex not yet in bags, while maintaining the four properties above.

Let C be an arbitrary connected component (not yet in bags) connected to bag B, and having at most $6p$ neighbors there. If C has strictly less than $6p$ neighbors in B, then create a new bag B' composed of these neighbors and one vertex of C, and connect it to B. This satisfies all four properties above.

It remains to deal with the case that C has exactly $6p$ neighbors in B. Denote this set of neighbors by X. Consider now for the sake of the argument the optimal tree decomposition for G. (This tree decomposition is unknown to the algorithm. Later we will see how the algorithm can be run despite this.) By the remark following Lemma 6 there is a bag S in this optimal tree decomposition that separates $X \setminus S$ into two parts X_1, X_2, each with at most $4p$ vertices. Let $S' = S \cap (B \cup C)$. Pick an arbitrary vertex $v \in C$, create a bag $B' = X \cup S' \cup \{v\}$ and connect it to B. Observe that $|B'| \leq 6p + (p+1) + 1 \leq 8p$ (the last inequality holds because $p \geq 2$), as desired. Observe also that the new partial tree decomposition includes at least one new vertex, and moreover, it is a tree decomposition of the subgraph induced by the vertices in its bags. The main point that remains to show is that properties 3 and 4 above hold. Let C' be an arbitrary connected component of $C \setminus B'$. Its neighbors are in B' and in no other bag. C' can have neighbors in either X_1 or X_2 but not both (as otherwise S' did not separate X_1 from X_2). Hence it has at most $4p + (p+1) + 1 \leq 6p$ neighbors in B', as desired.

The part missing from the above algorithm is the question of how to find S as above (as the optimal tree decomposition is not given). This can be done by exhaustive search. Partition X in all possible ways into three parts X_1, X_2 and $S \cap X$ (of sizes at most $4p$, $4p$ and $p+1$ respectively). There are at most 3^{6p} ways of doing so. For each such partition find the minimum cardinality set of vertices that separates between X_1 and X_2. Modifying G by unifying X_1 into one vertex s and X_2 into another vertex t, this becomes the minimum vertex (s,t)-cut problem in the modified graph. This can be solved in polynomial time using max-flow min-cut techniques. In particular Menger's theorem applies, equating the maximum number of vertex disjoint s-t paths with the size of the smallest vertex s-t cut, because s and t are not adjacent in the modified graph (there are no edges between X_1 and X_2).

5 A general polynomial time algorithm

Robertson and Seymour show a general condition under which a graph property has a polynomial time algorithm. We review here without proofs some of the main ingredients of their theory.

Define a partial order on graphs such that $H \leq G$ iff H is a minor of G (isomorphic graphs are considered identical). A family $\mathcal{F}$ of graphs is closed w.r.t. taking minors if $G \in \mathcal{F}$ implies $H \in \mathcal{F}$ for every $H < G$. There are several ways of expressing a minor closed family $\mathcal{F}$:

1. Via a graph property. For example, the family of planar graphs, the family of graphs of treewidth at most 2, etc. Sometimes such a representation is not known.

2. As an explicit list of graphs. This representation might be infinite.

3. By a list $\mathcal{O}$ of graphs not in $\mathcal{F}$. Again, this representation might be infinite.

4. Note that if $H \in \mathcal{O}$ then H cannot be a minor of a graph in $\mathcal{F}$. Hence we may equivalently think of $\mathcal{O}$ as a list of *forbidden minors* – those graphs that never appear as minors of graphs in $\mathcal{F}$.

5. The list $\mathcal{O}$ may be redundent in the sense that it might contain two graphs H_1 and H_2 with $H_1 < H_2$. Removing H_2 from $\mathcal{O}$, we can still define $\mathcal{F}$ as those graphs that do not have minors in $\mathcal{O}$. Removing graphs from $\mathcal{O}$ in this way, we remain with a list $\mathcal{O}$ of forbidden minors that is minimal – no forbidden minor is contained in another. This list is referred to as an *obstruction set*. For example, for planar graphs $\mathcal{O} = K_5, K_{3,3}$, and for graphs of treewidth at most 2, $\mathcal{O} = K_4$.

Let us study more carefully the forbidden minor representation. In what sense is it better than the explicit list of graphs representation? For planar graphs, we see an obvious advantage – the list of forbidden minors is finite (just two graphs). But is this a coincidence? This brings us to Wagner's conjecture.

A partial order over an infinite set is a *well-quasi-order* (wqo) if it does not contain an infinite *antichain*, where an antichain is a set of elements no two of which are comparable. (A wqo is required to also not have any infinite down-chain, but this condition easily holds in our context.) Wagner conjectured that the minor-induced partial order for graphs is a wqo. Robertson and Seymour proved this conjecture. It follows that for *every* minor closed family $\mathcal{F}$ of graphs, the number of minimal elements (minors) in the corresponding obstruction set is finite (because they form an antichain), and hence $\mathcal{F}$ can be represented by a finite list of forbidden minors. We note that the theorem of Robertson and Seymour does not tell us how to find the obstruction set – it just proves its existence. For planar graphs we know the obstruction set. For many other minor-closed families we do not know it.

Robertson and Seymour go a step further. They use their theory to design polynomial time algorithms for certain key problems. For example, they show that the following problem has a polynomial time algorithm for every fixed k:

Disjoint paths: given an undirected graph G and k pairs of vertices $(s_1, t_1), \ldots (s_k, t_k)$, are there vertex disjoint paths connecting s_i and t_i for every pair?

To appreciate this algorithm, note that the same problem is NP-hard on directed graphs, even when $k = 2$.

Likewise, they design an algorithm that for every fixed H checks whether H is a minor of $G(V, E)$. The running time of the algorithm is polynomial in V, $O(|V|^3)$, where the leading constant depends only on H (in a way that grows super-exponentially with H). When H is part of the input, the problem is NP-hard.

This leads to the following general theorem:

Theorem 10 *Every property of graphs that is inherited by minors (such as being embeddable on a surface of genus g, having a linkless embedding in 3-dimensional space, etc.) can be* decided *in polynomial time.*

The algorithm implied by the theorem is the list of forbidden minors for the infinite family of graphs having the graph property, coupled with the algorithm for checking minor inclusion. As remarked earlier, the theory just shows the existence of a polynomial time

algorithm, but does not necessarily produce such an algorithm. Moreover, the algorithm just answers *yes* or *no*, without necessarily supplying a witness in the usual sense (e.g., the algorithm may decide that the graph has a linkless embedding without actually finding one).