Lecture 11

Minor Graph Theory and the Ellipsoid Algorithm

Nov 8, 2004
Lecturer: Kamal Jain
Notes: Atri Rudra

In the last lecture we saw how the lemma on characterizing the tree width does not generalize to graphs of tree width three and higher. In this lecture we will study tree width in some more detail and outline how tree decomposition can be used to solve NP-Complete problem on graphs with fixed tree width. We will also introduce the Ellipsoid algorithm for solving convex programs.

11.1 Minor-close Properties

We start with a definition:

Definition 11.1. A property $\mathcal{P}$ is a *minor-close* property if whenever any graph G satisfies $\mathcal{P}$, any minor of G also satisfies $\mathcal{P}$.

It is easy to see that if $\mathcal{P}_{for}$ is the set of all forests then $\mathcal{P}_{for}$ is *minor-close*. However the set of trees is not a minor-close property as deleting an edge gives a minor which is not a tree. Other examples of minor-close properties are the set of series-parallel graphs and planar graphs.

We are going to state the next theorem without proof:

Theorem 11.1. *(***Forbidden Graph Minor Theory***) If $\mathcal{P}$ is a minor-close property then there exists a finite set of graphs $\{M_1, \cdots, M_k\}$ such that a graph G satisfies $\mathcal{P}$ if and only if G does not have any M_i (for $1 \leq i \leq k$) as a minor.*

Remark. Note that Theorem 11.1 gives a coNP certificate for $\mathcal{P}$. We also point out that the Kuratowski's Theorem for graph planarity is a special case of this theorem. Finally, we remark that the proof for Theorem 11.1 is constructive.

We next state another important theorem (again without proof):

Theorem 11.2. *Given a* fixed *graph H, there is a polynomial time algorithm which tests whether a given graph G has H as it's minor or not.*

Before we state the next result, let us recollect the definitions of tree decomposition and tree width of a graph from the last lecture.

Definition 11.2. A tree decomposition of a graph G is given by a tree T and the set of "bags" $\{H_u\}_{u \in V(T)}$, where each H_u is a sub-graph of G, with the following properties–

1. $\cup_{u \in V(T)} H_u = G$
2. For all $u, v, w \in V(T)$ such that v lies on the path from u and w in T, $H_u \cap H_w \subseteq H_v$.

An equivalent definition for tree decomposition is the following:

Definition 11.3. A tree decomposition of a graph G is given by a tree T and the set $\{H_u\}_{u \in V(T)}$ where each H_u is a sub-graph of G, with the following properties–

1. $\cup_{u \in V(T)} H_u = G$
2. For all $v \in V(G)$, the set $\{i | v \in V(H_i)\}$ spans a subtree of T.

Definition 11.4. A graph G has tree width p if and only if there exists a tree decomposition $(T, \{H_u\}_{u \in V(T)})$ such that

3. For all $u \in V(T)$, $V(H_u) \leq p + 1$.

We now present the following lemma which along with Theorems 11.1 and 11.2 implies a polynomial time algorithm for checking if a graph has some fixed tree width.

Lemma 11.3. *Fix p. The set of all graphs with tree width atmost p is a minor closed property.*

Proof. Let G be a graph with tree width atmost p. We now need to show that any minor M has tree width atmost p. Recall that one obtains M from G by a sequence of two basic operations and the proof would be complete if one shows that tree width being less than p is invariant over these two inductive steps.

Say M is obtained from G by deleting an edge (i, j). Consider the tree decomposition $(T, \{H_u\}_{u \in V(T)})$ for G. It can be easily verified that it is also a valid tree decomposition (with the necessary changes made to reflect the removal of the edge (i, j)) for M and thus, M has tree width atmost p.

We now consider the case when we contract the edge (i, j) to a vertex k. Let $(T, \{H_u\}_{u \in V(T)})$ be the tree decomposition for G. We create a new tree decomposition for M $(T, \{H'_u\}_{u \in V(T)})$, where

1. If $i, j \notin V(H_u)$ then $H'_u = H_u$
2. If exactly one of i or j is in $V(H_u)$, then replace i or j with k in H'_u.
3. If $i, j \in V(H_u)$ then H'_u is obtained from H_u by contracting the edge (i, j) to vertex k.

It is easy to see that $(T, \{H'_u\}_{u \in V(T)})$ satisfies property 1 of Definition 11.2 and property 3 of Definition 11.4. To complete the proof we need to show that for any $u, v, w \in V(T)$ if v is on the path from u to w in T then $H'_u \cap H'_w \subseteq H'_v$. From rule 1 of the creation of the new tree decomposition, it is clear that for all $\ell \in V(M) - \{k\}$, if $\ell \in H'_u$ and $\ell \in H'_w$ then $\ell \in H'_v$. Finally, we do a case analysis to show that if $k \in H'_u$ and $k \in H'_w$ then $k \in H'_v$. It is obvious that the above holds if $i \in H_u$ and $i \in H_w$ or $j \in H_u$ and $j \in H_w$ hold. Thus, we have the case as shown in Figure 11.1 where $i \in H_u$ and $j \in H_w$ and we have to show that either $i \in V(H_v)$ or $j \in V(H_v)$.

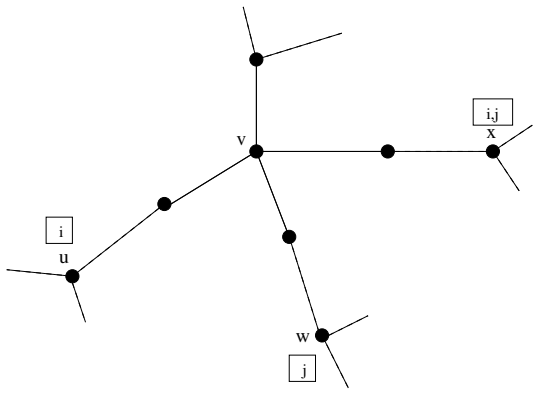

Figure 11.1: The "bad" case in the proof of Lemma 11.3. The vertices in the squares belong to the bags of the corresponding vertices in T.

Consider $x \in V(T)$ such that $i, j \in V(H_x)$. Note that the existence of such is vertex is guaranteed by property 1 of Definition 11.2. From property 2 of Definition 11.3, we have that u and x lie on a subtree of T: in particular they have (an unique) path passing through v. Finally, applying condition 2 of Definition 11.2 to $\{H_u\}_{u \in V(T)}$, we have $i \in V(H_v)$.

□

Given a fixed p, there are many NP-complete problem which become easy on graph of tree width atmost p. We look at this next.

11.2 NP-Complete problems on graphs of fixed tree width

We would be using Dynamic Programming to solve these problems. In this section we assume that G has tree width atmost p and that it's tree decomposition is given by $(T, \{H_u\}_{u \in V(T)})$.

11.2.1 Maximum Independent Set

We will briefly sketch the Dynamic Program because the ideas are similar to the one used for solving the problem on trees where $p = 0$. Consider the recursive step at the node $v \in V(T)$ as show in Figure 11.2.

Recall that $|H_v| \leq p + 1$. Thus for any vector $\boldsymbol{y} \in \{0,1\}^{p+1}$ represents all the choices we have for choosing vertices in $V(H_v)$. By $f_v^{(\boldsymbol{y})}$ we denote the size of the maximum independent set including vertices in $V(H_v)$ which are chosen if the corresponding bit in $\boldsymbol{y}$ is one. Note that for certain choices of $\boldsymbol{y}$ there cannot be an independent set and in those cases we set $f_v^{(\boldsymbol{y})} = 0$. Recall that in our Dynamic program for $p = 0$, $f_v^{(\langle 1 \rangle)} = f_v$ and $f_v^{(\langle 0 \rangle)} = g_v$. For each $\boldsymbol{y} \in \{0,1\}^{p+1}$, we define $f_v^{(\boldsymbol{y})}$ recursively in terms of the values of different choices of f for the children $u_1, \cdots, u_d$: i.e. $f_v^{(\boldsymbol{y})}$ and $f_{u_1}^{(\boldsymbol{y'})}$ can be combined if the set of vertices chosen from H_v and H_{u_1} according to $\boldsymbol{y}$ and $\boldsymbol{y'}$ can be combined to form an independent set. The rest of the argument is the same as that for the $p = 0$ case.

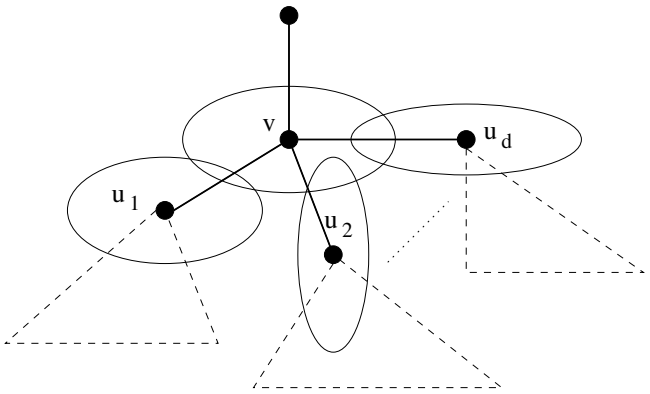

Figure 11.2: The recursive step for the Dynamic Program to solve Maximum Independent Set and Chromatic Number on graphs of tree width atmost p. The ovals represent the bags of the vertices in the tree while the dotted triangles denote the subgraphs "hanging" from the children of v.

The running time of the above dynamic program is $O(2^{p+1}n)$ which is linear for fixed p.

11.2.2 Chromatic Number

The chromatic number of a graph G (denoted by $\chi(G)$) is the minimum number of colors needed to color the vertices such that the end-points of any edge are colored differently.

We will again solve this by a Dynamic Program. Consider the recursive step of Figure 11.2. For every node in $V(H_v)$ there are $(\chi(G))^{p+1}$ possible colorings. The Dynamic Program would choose those coloring which can be extended to the coloring of it's children $u_1, \cdots, u_n$. However, for general graphs $\chi(G)$ can be as big as n so the above Dynamic Program would have a polynomial run time. However, the lemma below shows that for fixed p the run-time is actually linear in n.

Lemma 11.4. *If G has tree width atmost p then $\chi(G) \leq p+1$.*

Proof. Let $(T, \{H_u\}_{u \in V(T)})$ be the minimal tree decompositions of G. Consider any leaf $l \in V(T)$. The minimality of T implies that there is an edge (i,j) such that for any $v \neq \ell$, $(i,j) \not\subseteq H_v$. We further claim that atleast one of i or j (denote this vertex by i^*) is only present in H_ℓ, that is, $\forall v \neq \ell$, $i^* \notin V(H_v)$. If such an i^* did not exist then it would imply[1] that for some $v \neq \ell$, $(i,j) \subseteq H_v$ which is a contradiction. As $|V(H_\ell| \leq p+1$, degree of i^* is atmost p. We inductively color $G - \{i^*\}$ with $p+1$ colors and then color i^* with one of the $p+1$ colors not used. □

11.3 The Ellipsoid Algorithm

The Ellipsoid Algorithm was the first polynomial time algorithm to solve any Linear Program. Actually, it can solve any convex program. Finally, one of the most appealing feature about the Ellipsoid algorithm is

[1]The argument is similar to the one used in the proof of Lemma 11.3.

that the convex program need not be specified explicitly: any polynomial time separating oracle works fine too.

We first define an Ellipsoid.

Definition 11.5. An Ellipsoid in $\mathbb{R}^n$ is specified by an $n \times n$ matrix $\boldsymbol{A}$ and an $n \times 1$ vector $\boldsymbol{c}$ and is given by the set of points $E_{\boldsymbol{A},\boldsymbol{c}} = \{\boldsymbol{A}x + \boldsymbol{c} | x \in Ball(\mathbf{0}, 1)\}$, where $Ball(\mathbf{0}, 1)$ is the unit sphere centered at the origin. Further for any Ellipsoid E, let $E_{\frac{1}{2}}$ denote any symmetric half of E: i.e. any half obtained by "cutting" E by a hyperplane through the center of E. Also let $Vol(E)$ denote the volume of the ellipsoid E.

We have the following nice property of an Ellipsoid which we present here without proof:

Theorem 11.5. *Given any Ellipsoid E and it's symmetric half $E_{\frac{1}{2}}$, there exists another ellipsoid E' such that $E_{\frac{1}{2}} \subseteq E'$ and $Vol(E') \leq e^{\frac{-1}{3n}} Vol(E)$.*

Remark. We note that not all geometric structures have the nice property of Theorem 11.5 : for example, cuboids do not have this property. Let the original unit cube be C and let $C_{\frac{1}{2}}$ be any of the symmetric halves

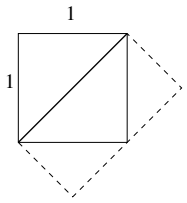

Figure 11.3: The cuboid does not have the nice property of Theorem 11.5. The example is for $n = 2$.

made by the diagonal. The two possible candidates for C' are C and the dotted cuboid (with sides of $\sqrt{2}$ and $\frac{1}{\sqrt{2}}$) and thus, in either case $Vol(C') = Vol(C) = 1$.

Given a convex set $\mathcal{C} \subseteq \mathbb{R}^n$, the Ellipsoid algorithm finds a $x \in \mathbb{R}^n$ such that $x \in \mathcal{C}$. Assume that the Ellipsoid algorithm is initially provided with an ellipsoid E such that $\mathcal{C} \subseteq E$. If the center of E is in $\mathcal{C}$ then we are done. If not we get a separating hyperplane passing the center from the separation oracle and we can use that to get an $E_{\frac{1}{2}}$ which contains $\mathcal{C}$ as shown in Figure 11.4. The next step is to invoke Theorem 11.5 to get a smaller ellipsoid E' which contains $\mathcal{C}$ as shown in Figure 11.5.

After n recursive steps, we would have an ellipsoid whose volume is atmost $(e^{-\frac{1}{3n}})^n = e^{-\frac{1}{3}}$ of the original E. Thus, the running time of the ellipsoid algorithm would be something like $p(n) \cdot n(\frac{Vol(E)}{Vol(\mathcal{C})})$ for some polynomial $p(.)$. This however, is not true. The problem is that when we compute the center S of E', S might be irrational. The solution is then to shift E' to get E'' centered at a nearby rational point S': the shifting is done by truncating bits. However, as show in Figure 11.6, E'' might not contain $\mathcal{C}$ at all.

The "solution" is to then expand E'' so that it contains $\mathcal{C}$ in it. Let $\mathcal{B}_\epsilon = Ball(\mathbf{0}, \epsilon)$ be the largest sphere contained in $\mathcal{C}$ and we always compute E'' (that is, a translated and uniformly expanded version of E') such that the running time is $p(n) \cdot n \log(\frac{Vol(E)}{Vol(\mathcal{B}_\epsilon)})$ for some polynomial $p(.)$, or in other words it is $p(n) \cdot n \log(\frac{R}{\epsilon})$ where $R = Vol(E)$.

Let us keep p bits of extra precision (we will fix p soon), that is, these p bit would be used to "round-off" the irrational S to a rational S'. Note that this implies that when expanding E to E' we would expand by a

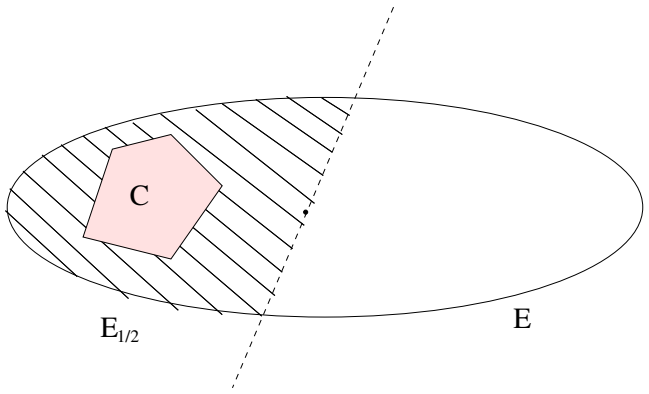

Figure 11.4: Generating $E_{\frac{1}{2}}$ from E using the Separation Oracle.

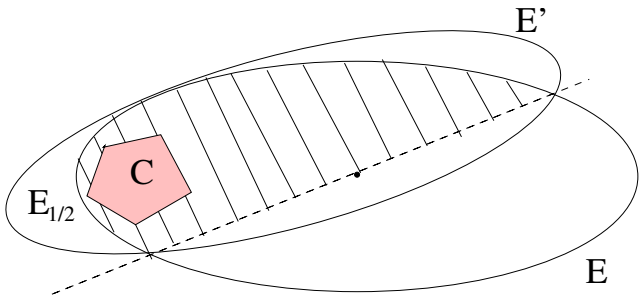

Figure 11.5: The recursive step of the Ellipsoid algorithm.

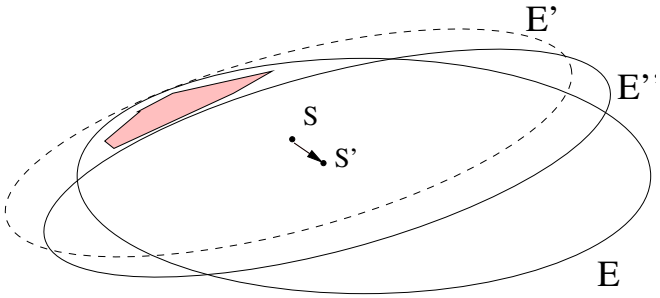

Figure 11.6: Moving the center of the new ellipsoid to a rational point.

factor of atmost $(1 + 2^{-p})$ in each dimension. By Theorem 11.5, we have that $Vol(E') \leq e^{-\frac{1}{3n}} Vol(E)$. Also by the argument in the last couple of lines, $Vol(E'') \leq Vol(E')(1 + 2^{-p})^n$, that is, $Vol(E'') \leq (e^{-\frac{1}{3}}(1 + 2^{-p})^{n^2})^{\frac{1}{n}} Vol(E)$. We need to choose p such that $e^{-\frac{1}{3}}(1 + 2^{-p})^{n^2} < 1$, which is satisfied by $(1 = 2^{-p}) < e^{\frac{1}{4n^2}}$. Using the identity $(1 + \frac{1}{x}) < e^{\frac{1}{x}}$, we get $2^p > 4n^2$. Thus, choosing logarithmic number of extra precision bits suffices.

The Ellipsoid algorithm as described in this section can run into the following problems when trying to

solve a linear program:

- If the feasible set can be a single point then $\epsilon = 0$.
- If the feasible set is not bounded then $r = \infty$.

We will tackle these issues in the next lecture.