2WO08: Graphs and Algorithms **Lecture 17** **Date: 11/5/2012**
Instructor: Nikhil Bansal **Scribe: Kim Starmans**

1 Spanning Tree Polytope

In the last lecture we have seen the following linear program for the spanning tree problem:

$$\min \sum_e c_e x_e$$

$$\sum_e x_e = n - 1$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \forall S \subseteq V$$

$$0 \leq x \leq 1$$

We also saw the following theorem:

Theorem 1 *The LP above always has an optimum integer solution.*

In the previous lecture, we gave a proof of this theorem using tight sets. Now we will see another proof based on a counting argument.

Proof: Take some optimum fractional solution x. If $x_e = 0$, we delete this edge and never consider it. So, we can assume that $x_e > 0$ for all edges in the graph. We claim that there exists a vertex v_i in our graph that is adjacent to only one edge in our LP-solution. Let us first see why this is enough to show that the spanning tree polytope has an integer optimal solution, and later on we will prove this claim itself.

Consider such a vertex v_i with one incident edge. We show that $x_e = 1$. We know that x_e satisfies all above constraints. In particular,

$$\sum_{e \in E(S)} x_e \leq |S| - 1$$

for any S, so also for $S = V \setminus \{v_i\}$. This set has size $n-1$, and therefore this constraint says for this particular set:

$$\sum_{e \in E(S)} x_e \leq n - 1 - 1 = n - 2$$

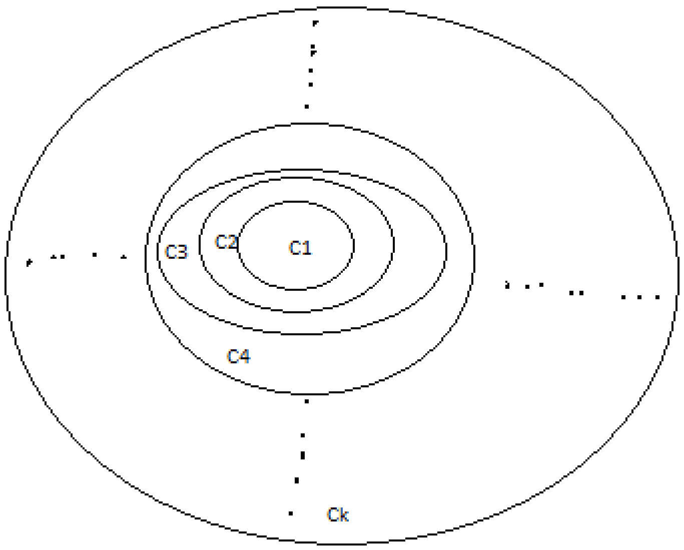
. As the total sum of x_e is $n-1$, this x_e we were looking at must be 1.

Now we look at $G \setminus e$, where e is this previous edge. We now claim that the residual LP (which is the previous LP without x_e) is still feasible for $G \setminus e$. For our constraints, the only interesting case is of course when our edge was in the set S we look at. Removing v and e makes the size of S equal to $|S| - 1$. As the lefthandside also decreases by 1 (we removed $x_e = 1$), the constraint still holds. As the original LP on G had cost C, and our residual LP on $G \setminus e$ has cost $C - c_e$, this is still ok. We can now repeat the procedure with our residual graph, and keep doing that until we have no edges left.

This leaves us with our original claim. Before we can prove this, we first have to prove some other things. We will continue this proof later on. □

We are now going to look at our basic LP-solution. If $x_e = 1$, we already know what to do by the above argument. If $x_e = 0$, we learned above that we can just throw it out. So the only interesting cases left are those where $0 < x_e < 1$. We will now only look at those. We have some solution x consisting of x_e which satisfies $Ax = b$. We now look at a linearly independent subset of tight constraints. Any basic LP-solution is determined by tight linearly independent constraints. $0 \leq x_e \leq 1$ is not a tight constraint (if it would be, it would be only 0 or 1), so it must be determined by some subset of the other constraints, $\sum_e x_e = n - 1$ and $\sum_{e \in E[S]} x_e \leq |S| - 1$. We now claim the following:

Claim 2 *There is a chain $C_1 \subseteq C_2 \subseteq ..... \subseteq C_k$ that determine V. So in particular: any tight con-*

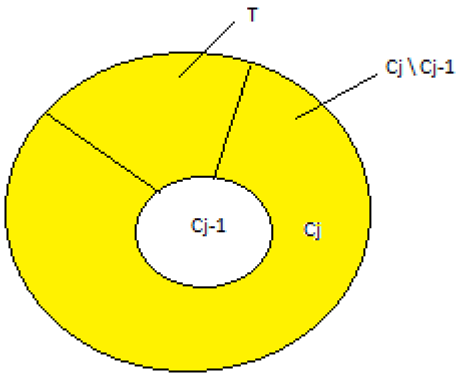


straint lies in span$(C_1, C_2, ..., C_k)$.

Proof: Look at all tight sets (or constraints, sets correspond to constraints!) at x^*. Let X be a collection of those sets and let $C_1, C_2, ...C_k$ be some maximal chain in X. Assume now that some set T is not in the span of $C_1, C_2, ..., C_k$. We have two possibilities:

- T is not a subset of C_k. Look at $T \cup C_k$. If both T and C_k are tight, then $T \cup C_k$ is also tight (see [1]). Because $T \cup C_k$ bigger than C_k and includes C_k, we can just add this set to our chain: $C_1, C_2, ..., C_k, T \cup C_k$. This obviously contradicts maximality of our original chain, and therefore we know T was already in the span of $C_1, C_2, .., C_k$.

- $T \subset C_k$. Let us prove that T is not a subset of $\bigcup_{j \in I} c_j \setminus c_{j-1}$. Let's see why. Let X_{c_j} be the incidence vector of set C_j. This is a vector with ones in certain places and zeros in the others. We can now write the incidence vector X_T as $\sum_{i \in I} X_{c_j} - X_{c_{j-1}}$ and therefore we can write all constraints for T in terms of constraints for the C_j. This proves that T cannot be in the union mentioned above. As it is not in this union, T must look like this:



Now look at $Y = (C_j \cap T) \cup C_{j-1}$. $(C_j \cap T)$ is a tight set because both C_j and T are tight sets, and as C_{j-1} is also a tight set, the union of both is also tight (see again [1]). We now have a set that is tight, bigger than and including C_{j-1} and smaller than and is included by C_j. Therefore we can add it to our chain: $C_1, C_2, ...C_{j-1}, Y, C_j, ...C_k$. This again contradicts maximality of the original chain, and proves that T was already spanned by this chain.

□

Let us now return to the proof of our original theorem:

Proof:(continued) We now have an LP-solution generated by a chain. C_1 must have at least 2 vertices in it, because it has to have an edge. C_k can have at most n vertices, because there are no more in the graph. So the length of our chain can be at most $n-1$. This means that we can have at most $n-1$ variables x_e such that $0 < x_e < 1$. This is because they are completely determined by our chain. Assume all vertices had at least 2 edges adjacent to them in the LP-solution. Then the degrees would sum up to something bigger than $2n$ and therefore there would be at least $2n/2 = n$ edges. This contradicts our fact that there are at most $n-1$. So our assumption was wrong: there must exist at least one vertex that has only 1 edge adjacent to it in the LP-solution. This concludes the proof of the original theorem: Every optimal LP-solution can be written as integer. □

2 Degree-bounded Spanning Tree

We now look at the problem of finding a degree-bounded spanning tree. This means that, in addition to finding a min-cost spanning tree in the, we also have to satisfy certain bounds on the vertices. Let's say a vertex v has degreebound d_v, that is, there can only be d_v edges adjacent to v. To see that this is an NP-hard problem, we set $d_v = 2$. Finding such a spanning tree is the same as finding a Hamiltonian path, from which we knew it was NP-hard[2]. We will now show that there is a polynomial time algorithm if we allow ourselves to relax the degree constraints a little bit. More precisely, let OPT denote cost of the min-cost spanning tree subject to the degree bounds. We will give a polynomial time algorithm that finds a spanning tree of cost at most OPT, but with possibly a +2 violation of the degree constraints.

Let us first set up our LP:

$$min \sum_e c_e x_e$$

$$\sum_e x_e = n - 1$$

$$\sum_{e \in E[S]} x_e \leq |S| - 1 \forall S \subseteq V$$

$$\sum_{e \in \delta(V)} x_e \leq d_v \forall v$$

$$0 \leq x_e \leq 1$$

Note that it is mostly the same as our original spanning tree LP, but the fourth line adds the degree-bounds. We now assume there is a solution, but it may be fractional. We propose an algorithm that rounds the fractional solution to an integer one, without increasing the cost, but possibly violating degrees a bit.

Algorithm: The algorithm proceeds iteratively in several steps, where we make some progress towards in integer solution in each step. If $x_e = 0$ or $x_e = 1$, we already can make progress. If zero, we can drop the edge, and if one, we can pick it in our solution (once and for all, and lower the degree bounds by 1 for the vertices adjacent to e).

So, let us assume that $0 < x_e < 1$ for all edges e. We know that solution x is determined by the tight constraints, and in fact by some linearly independent set of constraints that span these tight constraints. So the number of variables x_e is the number of linearly independent tight constraints (we have seen that in the last section). We write this as: |support| =|tight linearly independent set of constraints|.

Lemma 3 *1. $\exists v$ such that only one edge is adjacent to it, or*
2. $\exists w$ such that it has a degree bound and only ≤ 3 edges are adjacent to it.

If we can prove this lemma, it implies our algorithm:
if 1.: We can prove the value of this edge is 1, as we have seen for regular spanning trees. We then continue iteratively.
if 2.: We drop the degree constraint and resolve the simpler LP and continue iterative. The worst case would be: it had 3 edges, and we could pick all 3 of them eventually, and get degree (almost) 3 on the vertex. The degreebound could have been 1 (which is the smallest value possible) and we have a violation of +2 as desired. All other cases are better.

Proof: Let T be the set of type constraints and W the set of tight degree bounds. There exists a chain with the same span as T, as seen before. We can now construct a linearly independent family by adding the degree bounds. If we come across a degree bound which is already spanned by the chain, we of course don't add it. Now let C be the chain, and D the collection of degree bounds we added. We know $|supp| = |C| + |D| \leq n - 1 + |D|$ because $|C| \leq n - 1$. Now suppose all vertices have at least 2 edges adjacent to them and those with degree bounds have at least 4 edges. There are at most $n - |D|$ vertices without degree bounds. The sum of degrees will be $2(n-|D|)+4|D| = 2(n+|D|)$. The number of edges in the support now is at least $2(n+|D|)/2 = n+|D|$. This is bigger than $|C|+|D|$, which contradicts what we computed before. This means our assumption is wrong, so either there is a vertex with only one edge adjacent to it, or some vertex with a degree bound has at most 3 edges adjacent to it, and our lemma is proven. □

3 Matroids

Definition 4 *Let U be a collection of elements and let $\mathcal{I} \subseteq 2^U$. A matroid now is defined by the following 2 properties:*
1. If $I \in \mathcal{I}$ and $A \subset I$, then $A \in \mathcal{I}$.
2. If $I \in \mathcal{I}$ and $I^* \in \mathcal{I}$ such that $|I^*| > |I|$, $\exists x \in I^* \setminus I$ such that $I \cup \{x\} \in \mathcal{I}$.

The sets I in the collection $\mathcal{I}$ are called independent sets.

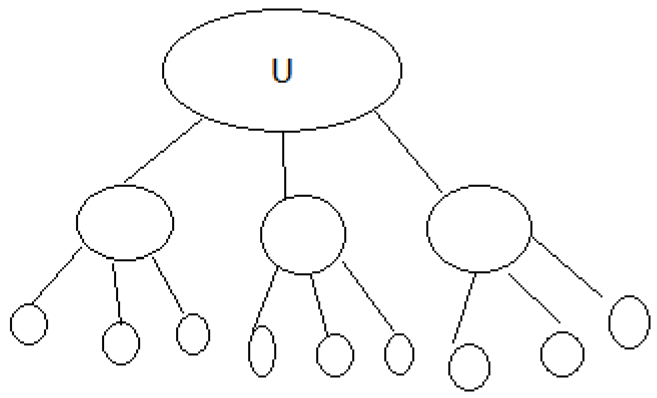
Example 1: Graphic Matroid
In the graphic matroid on a graph G, U is the set of all edges, so $U = E$. $\mathcal{I}$ is all subsets of edges that form a forest. Property 1 is obvious, because a subset of a forest is always a forest itself. For the second property, it is required to show that adding an edge from a bigger forest to a smaller one does not yield any cycles (this is an easy exercise).

Example 2: Vectors of $\mathbb{R}^n$
Let U be vectors $\{v_1, ..., v_k\}$ in $\mathbb{R}^n$. A subset I of those vectors is an independent set in the matroid if it has only linearly independent vectors. Property 1 again is easy, as every subset of independent vectors is by definition independent. For property 2, look at 2 sets of linearly independent vectors, one bigger than the other. Now find a vector x in I^* that is not already in the span of I, and add it to I.

Example 3: Uniform Matroid
Let U be $\{1, 2, ..., n\}$. Now I is an independent set if $|I| \leq k$ where k is some number chosen in advance. Property 1 can be seen as follows: If x is a subset of I, then of course the size of x is smaller than the size of I. As for property 2: If I^* is bigger than I, there must be an element in I^* which is not in I and we can add that and still have an independent set. This is always possible, because if both I and I^* are $\leq k$, and I^* is bigger than I, I must be strictly smaller than k and therefore we can add an element.

Example 4: Patrition Matroid
Again let U be $\{1, 2, ..., n\}$. Now let $p_1, p_2, ..., p_k$ be a partition of U, and let $h_1, h_2, ..., h_k$ be some numbers corresponding to the partitions. Now I is an independent set if $|I \cap p_i| \leq h_i \forall i = 1, 2, ..k$. Property 1 follows immediately, as x will be a subset of I, and then of course its intersection with a p_i can never get bigger than it was before. We now look at property 2. If $I \cap p_i = h_i$, we cannot pick any elements in I^* that are in this p_i. So now we look at the p_i for which $I \cap p_i$ was strictly smaller than h_i. In at least one of them I^* must have an element that is not in I (otherwise it would not be bigger!). We take that element as our x.

Example 5: Laminar Matroid
$U = \{1, ...m\}$. We partition U as in the following picture:



We have sets S_i for which either $S_i \cap S_j = \emptyset$ or $S_i \subset S_j$. We also have again some numbers h_i. Now I is an independent set if $|I \cap S_i| \leq h_i$. Property 1 follows again from the fact that x is a subset of I and therefore the intersection with some S_i can never be bigger than before. For property 2, we look again at those S_i for which $|S_i \cap I|$ was strictly smaller than h_i. I^* again must have some x that was not in I, or else it would not be bigger. Take that x. Adding it to I does not violate the constraints, because the size of one of the $|S_i \cap I|$ only increases by 1, and we took only those that were strictly smaller than h_i.

Example 6: Traversal Matroid
Let $G = (V_1, V_2)$ be a bipartite graph. Let U be V_1. I is independent if it can be covered by a matching.

Definition 5 *The base of a matroid is an independent set of maximal cardinality.*

Proposition 6 *All bases have the same size.*

Proof: Take two bases, B and B^*, where $|B^*| \geq |B|$. By property 2 of matroids, we can extend B such that it is still an independent set, but now has bigger cardinality. This says that B was not a basis in the first place, and it can made into a basis of same size as B^*. $\square$

Definition 7 *The size of a basis is called the rank of the matroid.*

Note that, in case of the graphic matroid, finding a basis is just finding a spanning tree! We can generalize this idea into finding a max-weight basis for a matroid.

Theorem 8 *The greedy algorithm always works. Recall the greedy algorithm: Order the weights in decreasing order. Take w_1 first, look at w_2. If you still get an independent set, take it, otherwise go to w_3, and so on.*

Proof: Suppose that greedy does not produce an optimum solution. Take some optimal solution and order the elements $t_1 \geq t_2 \geq ..... \geq t_k$ according to non-increasing values. Let the solution found by the greedy algorithm be $s_1 \geq s_2 \geq ..... \geq s_k$. Since greedy is sub-optimum there is some point i where $s_1 + .. + s_i < t_1 + ... + t_i$. Take the point such that it is the first time that the s falls below the t. Why did we add s_i at this step? We know that $w(t_i) > w(s_i)$ (otherwise i would not be the earliest point). So, $w(t_j) > w(s_i)$ for all $j = 1, \ldots, i$. Moreover, the set $\{t_1....t_i\}$ is independent. So when we were at s_{i-1}, at least one of $\{t_1....t_i\}$ is still available and greedy should have picked t_i instead of s_i. $\square$

Next time we will look at matroid intersections. That is, let's have $M_1 = (U, \mathcal{I}_1)$ and $M_2 = (U, \mathcal{I}_2)$. Find a max weight set $S \subset U$ such that S is independent in both M_1 and M_2. This can be applied in for example bipartite matching, where U is the collection of edges, M_1 is the partition matroid where p_i is the edges on vertex i at the left side, and M_2 does the same on the right side. A common independent set is a matching. We will prove this next time. Also we will look at the so-called Arborescense. This is a directed tree where all edges go down from a root.

References

[1] Reint den Toonder. Lecturenotes of lectures 15 and 16. June 8, 2012

[2] Mereke van Garderen. Lecturenotes of lecture 2. February 9, 2012