CS 270 Algorithms **Spring 2021**

Lecture 23: Linear Programming in Combinatorial Optimization

Lecturer: Prasad Raghavendra Scribe: Mayank, Jerry Lai

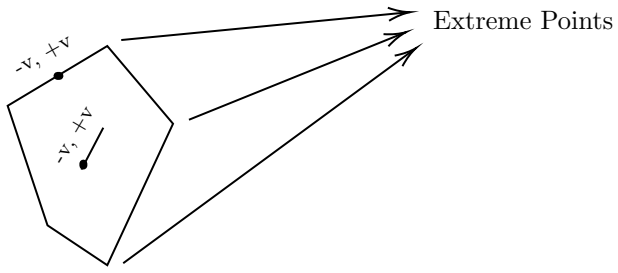
23.1 LP Relaxations for Combinatorial Optimization

There has been a lot of work on using linear programming relaxations for combinatorial optimization which spans from classical problems like computing matchings and spanning trees over graphs, to approximation algorithms for NP-complete problems. In this lecture, we will only be covering a relatively recent and widely used rounding technique from this vast literature — the Iterative Rounding Technique. We show how iterative rounding is useful in solving the following classical problems over undirected graphs:

1. Matchings or independent edge sets.
2. Spanning trees

23.2 Integral Polytopes

Definition 23.1 (Extreme Points)**.** *On a polytope P, x is an extreme point if $\nexists v \neq 0$ such that $x - v$ and $x + v \in P$. In other words, x is an extreme point if you cannot place any non-zero length line around it which is also in P.*



The above definition provides a great intuition on what extreme points are and how they look like, but in the rest of the discussion, we will be heavily using the following alternate definition of extreme points:

Definition 23.2 (Extreme Points)**.** *On a polytope $P \in \mathbb{R}^n$, every extreme point x is an intersection of n hyperplanes. For a linear program with polytope $P \in \mathbb{R}^n$, this can be interpreted as every extreme point x lies at the intersection of n linearly-independent tight constraints.*

Definition 23.3 (Integral Polytope)**.** *A polytope $P \in \mathbb{R}^n$ is integral if all of its extreme points x_i are integral, i.e., $x_i \in \mathbb{Z}^n$.*

23.3 Bipartite Matching

Consider an undirected graph $G = (V, E)$, where V is the vertex set and E is the edge set.

Matching: A matching $M \subseteq E$ for graph G is a set of edges such that $\forall e_1, e_2 \in M$, where $e_1 \neq e_2$, e_1 and e_2 have no common vertex.

Finding a matching in a bipartite graph can be cast as a network flow problem and therefore, can be solved using max-flow algorithms. However, we focus on solving this using a linear programming relaxation to be able to better appreciate the power and usefulness of this technique.

Minimum Weight Bipartite Matching: We focus on the following variant of the matching problem for the rest of the section:

Input: $G = (V_1 \cup V_2, E)$, where $|V_1| = |V_2| = n$. A weight function $w : E \to \mathbb{R}^+$ that maps each edge e to a non-negative weight $w(e)$ (alternatively also denoted by w_e).

Goal: Find the minimum weight matching, i.e., $\min_M \sum_{e \in M} w_e$.

23.3.1 LP Relaxation for Bipartite Matching

Let us look at the an LP relaxation for the minimum weight bipartite matching problem. We begin by defining variables x_e for each edge $e \in E$ as:

$$x_e = \begin{cases} 1 & \text{if } e \in M \\ 0 & \text{otherwise} \end{cases}$$

The above definition of x_e is our "intent" and doesn't necessarily reflect x_e values that are recovered by solving the LP. We will interchangeably use $x(e)$ or x_e. Let $\delta(v)$ denote the set of edges incident on vertex v.

The LP relaxation (with x_e as the variables) is[1]:

$$\begin{aligned} & \min \sum_{e \in E} x_e w_e & \\ \text{subject to} \quad & x_e \geq 0 & \forall e \in E \\ & x(\delta(v)) = \sum_{e \in \delta(v)} x_e \leq 1 & \forall v \in V_1 \cup V_2 \end{aligned}$$

The above constraints describe a polytope P corresponding to the feasible region of solutions to the LP. Since we are interested in finding integral matchings only, informally, we require that P captures integral matchings to the best extent possible. That is, the best the one can hope for is for the bipartite matching polytope P to be an integral polytope (as defined in section 23.2). We will now see some special properties of P using which we will be able to qualify the usefulness of solution returned by the LP.

[1] We will tweak our LP solver to only output extreme points as optimal solutions. This will make sure that LP doesn't output an empty set M as the trivial optimum.

23.3.2 Properties of Bipartite Matching Polytope P

We now prove the following theorem about P:

Theorem 23.4. *The bipartite matching polytope is integral.*

Assuming that the above theorem is true for a moment, note that this implies that if we instruct our LP to give an extreme point as the optimal solution, then we are sure that the point will be integral. Simplex algorithm is an example where the algorithm returns an extreme point, and in fact, for many algorithms we can make required tweaks to get extreme points as optimal solutions. This means that solving the LP will give you the exact solution that you wanted.

One way to prove the above theorem is by using the following fact: max-flow algorithms return integral flows when edge capacities on the graph are integral. We take a different route and instead of proving the above theorem directly, we prove the following *equivalent* statement:

Theorem 23.5. $\forall$ *weights* $w : E \to \mathbb{R}^+$, $\exists$ *an algorithm* $\mathcal{A}$ *that finds a matching* M *whose cost is the same as that at the LP optimum.*

The construction of algorithm $\mathcal{A}$ will be very similar to how iterative rounding method works. Therefore, another way to state the above theorem is that one can always round the LP solution to get a matching which preserves the cost at LP optimum. However, we will see in the construction that this rounding is never actually needed.

Let's start by first proving how the above two theorems are equivalent. It follows from the simple observation that if there was an extreme point x on the polytope which was not integral (was fractional), then for some choice of the weight function w, x would be the LP optimum. In that case, any rounding operation performed on x would not yield (or preserve) the exact same optimum cost. Therefore, the only way for the rounding operation to preserve the cost at LP optimum is when the rounding is never actually performed, i.e. the extreme points are all integral, implying that the polytope is integral.

Having established that the theorems are equivalent, we now present the construction of $\mathcal{A}$, which will prove theorem 23.5.

Theorem 23.5 (Restated)**.** $\forall$ *weights* $w : E \to \mathbb{R}^+$, $\exists$ *an algorithm* $\mathcal{A}$ *that finds a matching* M *whose cost is the same as that at the LP optimum.*

Proof. We have a graph $G = (V_1 \cup V_2, E)$. We perform the steps shown in algorithm 1.

We start by solving the LP to get an extreme point solution $x \in \mathbb{R}^E$. If this solution is integral, then we have the following two cases:

Case 1: $\exists e \in E$ such that $x_e = 0$. In this case, since LP is not including the edge e in the solution, we can safely remove this edge from the graph and restart. This will not affect the cost at optimum.

Case 2: $\exists e \in E$ such that $x_e = 1$. Since the LP is actually including (or matching) the edge e in the solution, the vertices (u, v) in edge e represent a matched pair. Therefore, we add e to the current matching M, remove $e = (u, v)$ from the graph, and restart.

Algorithm 1 Construction of $\mathcal{A}$

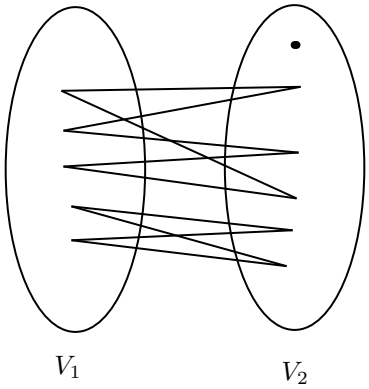
```
Initialize M = {}
Solve LP to get an extreme point solution x ∈ R^|E|
if ∃e ∈ E for which x_e = 0 then
    Set G ← G \ e
    Jump to step 2
end if
if ∃e ∈ E for which x_e = 1 then
    Update M ← M ∪ e
    Set G ← G \ e
    Jump to step 2
end if
```

If none of the above cases happen, then the only possibility is:

Case 3: (never happens) For all the edges e in the current graph, x_e is fractional. We will now prove that this case never happens.

We have that x is an extreme point with fractional $x_e, \forall e \in E$. Recall from section 23.2 that if x is an extreme point and polytope $P \in \mathbb{R}^{|E|}$, then x satisfies exactly $|E|$ linearly independent tight constraints. Since x_e are fractional, we know that the constraints $x_e \geq 0, \forall e \in E$ are not tight (because fractional value cannot be 0). This means that the only tight constraints come from $x(\delta(v)) = \sum_{e \in \delta(v)} x_e \leq 1, \forall v \in V_1 \cup V_2$. Let $W = \{v \mid x(\delta(v)) = 1\}$, i.e., W denotes the set of vertices for which the constraint is tight. Observe that for $x(\delta(v)) = \sum_{e \in \delta(v)} x_e = 1$, we need at least 2 incident edges on each vertex $v \in W$ (because for the sum to be $= 1$, at least two fractional values are needed). So, $\forall v \in W, d_v \geq 2$, where d_v is the degree of v.

We know that for any graph, $2|E| = \sum_{v \in V_1 \cup V_2} d_v$ which is $\geq \sum_{v \in W} d_v \geq 2|W|$ in our specific graph G. There can be two possibilities here: a) $|W| < |E|$ in which case the number of tight constraints are $< |E|$ which contradicts x being an extreme point; so this case doesn't happen, b) $|W| = |E|$. Let's look at the case b) in more detail. Case b) would mean that $\forall v \in W, d_v = 2$ and $\forall v \notin W, d_v = 0$. This is how the graph would look like in this case:



Sum of x_e for all edges in the graph $= \sum_{v \in V_1} x(\delta(v))$ and it is also $= \sum_{v \in V_2} x(\delta(v))$ (due to bipartite nature). Therefore, $\sum_{v \in V_1} x(\delta(v)) = \sum_{v \in V_2} x(\delta(v))$. This leads to one linear dependency in the tight constraints that

we had (because the degree of $v \notin W$ was 0, this dependency is on the tight constraints). So, the number of linearly independent tight constraints $\leq |W| - 1$. Therefore, the number of linearly independent tight constraints $\leq |E| - 1 < |E|$ which is again a contradiction to x being an extreme point. This proves that case 3 never happens. □

23.4 Spanning Trees

Spanning Tree: A spanning tree $T = (V', E')$ of a graph $G = (V, E)$ is a subgraph of G where $|V'| = |V|$ and T is tree.

We will use LP relaxation to compute a spanning tree of G. We begin by first describing the *subtour elimination LP* which defines a spanning tree polytope.

23.4.1 Subtour Elimination LP

Let us look at the following feasibility LP. We begin by defining variables x_e for each edge $e \in E$ as:

$$x_e = \begin{cases} 1 & \text{if } e \in T \\ 0 & \text{otherwise} \end{cases}$$

Let $E(S)$ denote the set of edges in the vertex set $S \subseteq V$. Subtour elimination LP finds a feasible solution $x \in \mathbb{R}^{|E|}$ subject to:

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

$$\sum_{e \in E} x_e = |V| - 1$$

Note that there are exponentially many constraints $\sum_{e \in E(S)} x_e \leq |S| - 1$. As a side note, one way (rather inefficient) to compute spanning trees is by constructing a separating oracle for these exponentially-many constraints and using ellipsoid algorithm to solve the LP. The polytope defined by the above constraints has the following interesting property:

Theorem 23.6. *The spanning tree polytope is integral.*

We won't be proving this theorem; the proof follows a similar iterative construction as used earlier. We will rather focus on a harder problem of bounded-degree spanning tree.

23.4.2 LP Relaxation for Bounded-Degree Spanning Tree

23.4.2.1 Problem Statement

Let $G = (V, E)$ be a graph with degree bounds b_v for a subset of vertices $v \in W$, where $W \subseteq V$. Similar to the LP for spanning trees, we define

$$x_e = \begin{cases} 1 & \text{if } e \in T \\ 0 & \text{otherwise} \end{cases}$$

As before, let $\delta(v)$ denote the set of edges incident on vertex v. We write a feasibility LP with the following constraints:

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

$$\sum_{e \in E} x_e = |V| - 1$$

$$\sum_{e \in \delta(v)} x_e \leq b_v \qquad \forall v \in W$$

Here, $\sum_{e \in \delta(v)} x_e \leq b_v$ is the key. Note that this is an extremely hard problem right now, because, for example:

$$\begin{aligned} deg(s) &\leq 1 \\ deg(t) &\leq 1 \\ \forall u \in V,\ deg(u) &\leq 2 \end{aligned}$$

In this case, the problem is essentially asking for a Hamiltonion Path, because the only tree where all the degrees are at most 2 is a path, and to be a spanning tree, it has to go through every vertex, making this a Hamiltonion Path Problem. This makes it a Hard problem.

23.4.2.2 Solution Algorithm

We will do something surprising. Here is a Theorem we'll show:

Theorem 23.7. *$\exists$ a poly time alg to compute a tree T s.t.:*

- $cost(T) \leq$ *LP-cost*
- $\forall v, deg_T(v) \leq b_v + 2$

In other words, we will violate the degree bounds by a little bit, but at the same cost.

For example, if we encounter the previous hamiltonian path problem, our solution will have the same cost as a hamiltonian path but the s. and t. vertices will have up to degree 3, the other vertices will have up to degree 4.

Additionally, it is possible to improve the algorithm to achieve $\forall v, deg_T(v) \leq b_v + 1$, but we will only do the $b_v + 2$ version here.

Proof. The algorithm (Leaf-finding) is similar to what we have seen in bipartite matching (1).

Algorithm 2 Leaf-finding algorithm

```
1:  while Not Done do
2:      Solve bounded degree spanning tree LP(G, W, b) & compute x ← extremal optima
3:      if x_e = 0 then
4:          Restart LP(G\e, W, b)                    ▷ Remove edge and run LP again
5:          So now there are no zero edges.
6:      else if Vertex v has deg(v) = 1 then
7:          You have to make this vertex the leaf.
8:          F ← F ⋃ (u, v)
9:          G ← G \ v
10:         b'_u ← b_u − 1
11:         Restart LP(G\v, W, b')
12:     else if Vertex v has deg(v) ≤ 3 then     ▷ Violates degree constraint by ≤ 2 which is ok
13:         Drop Σ_{e∈v} x_e ≤ b_v Constraint from LP
14:         restart LP with one fewer constraint
15:     else
16:         We will prove that this state will never be reached
17:     end if
18: end while
```

Proof by Contradiction:

There are at least $|E|$ tight constraints because x is optimal.

Unlike in Bipartite graphs, we have exponentially many constraints in:
$\sum_{e \in E(S)} x_e \leq |S| - 1 \forall$ sets $S \subseteq V$

Some of them are tight constraints. Let T be the set of tight constraints:
$T = \left\{ S \subseteq V \middle| \sum_{e \in E(S)} x_e = |S| - 1 \right\}$

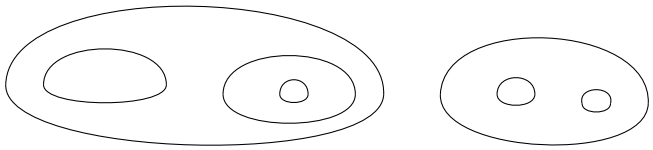
Fact:
$A, B \in T \Rightarrow A \cup B, A \cap B \in T$

This lets you do an operation called "uncrossing": Going from 2 sets A and B which are crossing, to 2

different sets $A \cup B$ and $A \cap B$ which do not cross.
This allows you to get to what's called a "Laminar Family"

Laminar family L is a collection of sets such that no pairs cross.
Definition of cross:
$A \cap B \neq \emptyset$ but $A \nsubseteq B$ and $B \nsubseteq A$

Example of laminar family:



You can think of a Laminar family as a Forest of Large sets containing smaller sets, with leaves being the smallest sets.

Lemma 23.8. *A Laminar family L can have $|L| \leq 2n - 1$*
*A Laminar family with **no singleton sets** can have $|L| \leq n - 1$.*

Lemma 23.9. *Key lemma:*

$$\exists \text{ a laminar family } L, span(L) = span(T)$$

So although there are exponentially many linear constraints, the number of linearly independent constraints is equal to the size of some Laminar Family.

$$\Rightarrow dimension(span(T)) \leq |L|$$

$$\Rightarrow dimension(span(T)) \leq |V| - 1$$

The goal of all that was to show that there are at most $|V| - 1$ tight constraints this part of the LP:

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

and

$$\sum_{e \in E} x_e = |V| - 1$$

Which leaves the degree constraints:

$$\sum_{e \in \delta(v)} x_e \leq b_v \;\; \forall v \in W$$

Which have at most $|W|$ tight constraints.

For the next part of the proof, we will assume that we have failed the 3 cases in the Leaf-finding algorithm, thus reaching the 4th case (line 15 in pseudocode), which is what we are trying to prove is impossible.

So:

$$\forall v \in W \;\; deg(v) \geq 4$$
$$\forall v \in V \setminus W \;\; deg(v) \geq 2$$

If either of the above are false, we would have gone with one of the 3 cases of the algorithm.

$$|E| \geq \frac{1}{2} \sum_{v} deg(v)$$

$$= \frac{1}{2} \left[4|W| + 2(n - |W|) \right]$$
$$|E| \geq n + |W|$$

As stated at the beginning of the proof, we need $|E|$ tight constraints, so we need $\geq n + |W|$ tight constraints.

As shown in the previous parts of the proof, our LP provides at most $n - 1$ tight constraints from the laminar families, and $|W|$ tight constraints from the degree constraints.

Therefore, in total, it has $n + |W| - 1$ tight constraints which is not enough. This is the contradiction: since we have not enough tight constraints, the LP solver would never terminate or get stuck in this position.

□

23.4.2.3 Comments on this algorithm

So, to circle back, this is quite a simple algorithm. There exists a modification that allows $\forall v, deg_T(v) \leq b_v + 1$. To get the better result, instead of "throwing away" the constraint when we get $deg(v) \leq 3$ in line 12 of the pseudocode, we can throw it away when we get $deg(v) \leq 2$.

In general, the technique used by the Leaf-finding algorithm is called the **iterative rounding technique**. More information on iterative rounding can be found in the book by Lau, Ravi and Singh. In iterative rounding, you find something that is close to integral in your solution, and you "fix" it to make it integral and resolve the LP.

In linear programs, one other natural rounding technique is called **randomized rounding**. In randomized rounding, if you look at a vertex, you choose whether or not to add it with a 1/2 probability either way, and you do it independently for each vertex.

You can actually reprove many of the classical results in combinatorial optimization using the **iterative rounding technique** and you can get approximation algorithms for some of them, such as this one. This is an approximation algorithm not in the factor sense, but in the sense that you relax the degree by at most 1. You have a tree that is slightly weaker but with the same cost.

There are problems for which you can get approximation algorithms, but in the usual sense, using the **iterative rounding technique**. One example of this is Vertex Cover. We will attempt to do this now.

23.4.3 Vertex Cover using Iterative Rounding

23.4.3.1 LP for vertex cover

$$\begin{aligned} \forall e = (u,v) \quad & x_u + x_v \geq 1 \\ & 0 \leq x_u \leq 1 \qquad\qquad \forall u \in V \\ & \min \sum w_u x_u \end{aligned}$$

23.4.3.2 Iterative Rounding Algorithm

- Solve LP (extreme point)
  1. Suppose $x_v \geq \frac{1}{2}$. for $v \in V$
     Then add s to vertex cover
     (Repeat)
     (Note: Normally, when you solve this, you will get the vertex cover in one step with no need to repeat. The repeat step is here for our proof only.)

23.4.3.3 Proof

We need to prove that we will never get "stuck" with this algorithm.

A simple proof is using the constraint $x_u + x_v \geq 1$. In order for this constraint to be satisfied, one or the other vertices have to be $\geq \frac{1}{2}$.

Alternatively, you can prove this algorithm using a dimension argument, as we did with the leaf-finding algorithm for bounded-Degree Spanning Trees.

Proof. You can prove that in an extreme point, by counting the number of tight constraints, there has to be at least one coordinate that is at least $\frac{1}{2}$. We will use proof by contradiction, and assume that all of the coordinates are less than $\frac{1}{2}$.

- We need at least $|V|$ tight constraints because that is the number of variables.
- None of the constraints in $0 \leq x_u \leq 1 \ \ \forall v \in V$ are tight.
- $\forall e = (u,v) \ \ x_u + x_v \geq 1$ also aren't tight, based on the assumption that the values are less than 1/2.

Thus, we show that if all of the values are less than $\frac{1}{2}$, we won't have any tight constraints at all. Thus, the LP wouldn't get stuck in a state where the values are all less than $\frac{1}{2}$. □

23.4.3.4 Comments on this algorithm

You can get other approximation algorithms using the same technique we used just now. One of the famous ones is Network Design problems. You can get a factor of 2 approximation by using this technique, i.e.:

- solve LP
- go to extreme point
- There will be one edge or something that at least $\frac{1}{2}$, so round that to 1. You lose a factor of 2, and you repeat.
- You use the same sort of dimension counting argument to show you're never stuck.

In the next lecture we will cover matchings in non-bipartite graphs.