CS 270 Combinatorial Algorithms and Data Structures, Spring 2015

Lecture 12: Linear programming

We start to move to the second part of the course and the main object of interest is linear programming.

Today, we introduce linear programming relaxations for combinatorial optimization problems, and discuss in what situations they solve the problems exactly, and the main concept is extreme point solutions.

We also mention how to solve LP efficiently, and how separation implies optimization.

Linear programs

In a linear program, we have a set of n variables $x_1, \dots, x_n$. We can add linear constraints to the variables, e.g. $x_1 + 3x_2 - 5x_3 \le 7$, $x_1 + 4x_2 = 6$, etc (but not strict inequalities), and we can optimize a linear objective function say max (or min) $3x_1 + 2x_2 - x_3$.

More compactly, if there are m constraints, we can put the constraints as rows of a matrix $A \in \mathbb{R}^{m \times n}$, and we can put the coefficients of the objective function as a vector $c \in \mathbb{R}^n$, and the right hand sides of the constraints as a vector $b \in \mathbb{R}^m$, and so the LP can be written as:

$$\max \langle c, x \rangle \qquad\qquad \max \langle c, x \rangle$$
$$\qquad\qquad\qquad \text{or}$$
$$Ax \le b \qquad\qquad\qquad Ax = b$$
$$\qquad\qquad\qquad\qquad\qquad x \ge 0$$

I think the first form is called the canonical form and the second is called standard form, but I prefer to just say the first is inequality form and the second equational form.

Anyway, they are equivalent and one can rewrite the first form as the second form and vice versa, e.g. by introducing new variables $s \in \mathbb{R}^m$ and write $Ax \le b$ as $Ax + s = b$, $s \ge 0$.

Okay, so this is the definition of a linear program.

Integer linear programs

Many combinatorial optimization problems can be easily formulated as an integer linear program, where we also have the additional constraints $x_i \in \mathbb{Z}$ or just $x_i \in \{0,1\}$.

For example, for bipartite matching, we can have one variable x_e for each edge e, and write

$$\max \sum_{e \in E} w_e x_e$$

$$\sum_{e \in \delta(v)} x_e \le 1 \quad \forall v \in V$$, where $\delta(v)$ denotes the set of edges incident to v.

$x_e \in \{0,1\} \quad \forall e \in E$

Then, the constraints say that either we pick e (by setting $x_e = 1$) or not (by setting $x_e = 0$), and the constraints $\sum_{e \in \delta(v)} x_e \le 1$ say that we can pick at most one edge for each vertex, and the objective is to maximize the total weight of the edges picked.

Similarly, we can formulate the maximum independent set problem as an integer linear program:

$$\max \sum_v c_v x_v$$

$$x_u + x_v \le 1 \quad \forall uv \in E$$

$$x_v \in \{0,1\} \quad \forall v \in V.$$

Yes, so that's very convenient, but it also says that solving integer linear programs are NP-hard in general.

Linear programming relaxations

As we will mention later, there are polynomial time algorithms to solve (ordinary) linear programs.

So, one approach to tackle combinatorial optimization problems is to write a linear program to pretend the integer linear program, and the natural way to do it is to write $x_e \in \{0,1\}$ as $0 \le x_e \le 1$ and hope for the best.

As you may imagine, this may not work so well, as now we are now optimizing a much larger domain (the set of solutions in $\mathbb{R}^n$) rather that the set of solutions in $\{0,1\}^n$.

Consider the maximum independent set problem in a complete graph and we know the integral optimal solution is one, but the LP solution $x_i = \frac{1}{2} \; \forall i \in V$ is a feasible solution to the LP solution with objective value $\frac{n}{2}$ (assuming all weights are one).

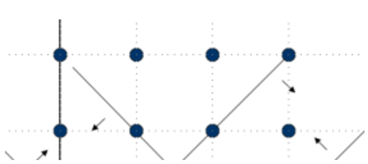
So, this approach totally fails and doesn't give any useful information to us.

Somewhat surprisingly, for most of polynomial solvable combinatorial optimization problems, we can actually extract optimal integral solutions from the LP relaxations.

In fact, it is widely accepted as a unifying algorithmic framework for polytime solvable combinatorial optimization problems.

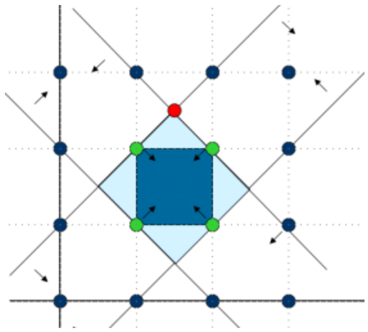
Let us first have more intuitions about why it could be the case.

Geometric interpretation



We are working over $\mathbb{R}^n$ ($\mathbb{R}^2$ in the picture).

Each constraint can be thought of as a half space.



Each constraint can be thought of as a half space.

The set of feasible solution is the intersection of the half spaces.

We are interested in the integral points in the feasible set.

Optimizing a linear objective function is to find a point further in some direction, e.g. if we optimize y, then the point on top is the optimal point.

Intuitively, given any direction to optimize, there are always some "corner points" that achieve optimality.

So, if we could show that all the corner points are integral, then we can say that the LP relaxation always has integral optimal solutions.

In fact, this is what we need to show if we are to prove that there are integral optimal solutions for every objective function, because intuitively every "corner point" is the unique optimal solution for some objective function.

So, this is what we are trying to do, to show that for some problems, every "corner point" is integral.

Corner points

Here we make the concept of corner points precise.

Consider an LP of the equational form: max $\langle c, x\rangle$ over $\underbrace{P}_{\text{(stand for polytope)}} := \{ x \mid Ax = b,\ x \geq 0 \}$.

Without loss of generality, we can assume that the rows of A are linearly independent.

There are three possible definitions of corner points.

1. Vertex solutions: A solution $x \in P$ is a vertex solution if $\nexists\, y \neq 0$ s.t. $x+y \in P$ and $x-y \in P$.

   This is to capture the geometric intuition that a corner point is not an average of two feasible points.

2. Extreme point solutions: A solution $x \in P$ is an extreme point solution if $\exists\, c$ such that x is the unique optimal solution. This is what we really want to show that they are integral.

3. Basic solutions: This definition is algebraic and is easier to deal with when we do computations.

   Let $\mathrm{supp}(x) := \{ i \mid x_i > 0 \}$ be the set of nonzero variables.

   Note that x_i corresponds to the i-th column of A.

   We say x is a basic solution if the columns correspond to $\mathrm{supp}(x)$ are linearly independent.

In the literature, the names are used interchangably (so my definitions may not be authentic), partly because of the following result.

Proposition The three definitions are equivalent.

Proof We will prove ③ ⇒ ② ⇒ ① ⇒ ③. Let A be an $m \times n$ matrix with $m \le n$ and full rank.

(③ ⇒ ②): Let $S := \mathrm{supp}(x)$. By ③, the columns of S are linearly independent.

Since A is of full rank, if $|S| < m$, we can extend S to m linearly independent columns, and without loss of generality we assume $S = \{1, 2, \ldots, m\}$.

Now consider the objective function $\min \sum_i c_i x_i$ where $c_i = 1$ if $i > m$ and $c_i = 0$ if $i \le m$.

Then, x is a solution with objective value zero, and we claim it is the unique one.

Note that any solution y with objective value zero must have $y_i = 0$ for $i > m$, as $y_i \ge 0$ by the constraints and $c_i = 1$ for $i > m$.

So, $\mathrm{supp}(y) \subseteq \{1, \ldots, m\}$. But then there is only one solution satisfying $Ay = b$ because the first m columns are linearly independent, and hence we must have $y = x$.

(② ⇒ ①): Suppose x is the unique optimal solution for objective $\min \langle c, x \rangle$.

Assume by contradiction that x is not a vertex solution, i.e. $\exists\, y \ne 0$ s.t. $x+y \in P$ and $x-y \in P$.

Note that either $c(x+y) \le cx$ or $c(x-y) \le cx$ or both, contradicting x is the unique optimum.

(① ⇒ ③): We prove the contrapositive that if x is not a basic solution, then x is not a vertex solution.

Let $S := \mathrm{supp}(x)$. If columns in S are linearly dependent, then $\exists\, y \ne 0$ with $\mathrm{supp}(y) \subseteq S$ and $Ay = 0$.

We claim that there exists a small enough $\varepsilon > 0$ such that $x + \varepsilon y \in P$ and $x - \varepsilon y \in P$, and that would imply that x is not a vertex solution, thereby completing the proof.

First, since $Ay = 0$, we have $A(x + \varepsilon y) = Ax = b$ and similarly $A(x - \varepsilon y) = Ax = b$, satisfying the equalities.

Also, since $\mathrm{supp}(y) \subseteq \mathrm{supp}(x)$, by choosing ε small enough, we have $x + \varepsilon y \ge 0$ and $x - \varepsilon y \ge 0$, satisfying the inequalities. So, both $x + \varepsilon y \in P$ and $x - \varepsilon y \in P$, and we are done. $\square$

Optimal corner points

Now, we would like to show that there is always some optimal extreme point solution.

Before we do that, I would like to state the corresponding definitions for the inequality form, as this is usually the form that we work with.

Recall the inequality form is $\min \langle c, x \rangle$, $Ax \le b$. Let $P := \{x \mid Ax \le b\}$ where $A \in \mathbb{R}^{m \times n}$.

The corresponding definitions are:

① <u>Vertex solutions</u>: x is a vertex solution if $\nexists y \neq 0$ s.t. $x+y \in P$ and $x-y \in P$.

② <u>Extreme point solutions</u>: x is the unique optimal solution for some objective function c.

③ <u>Basic solutions</u>: Let $x \in P$. We say a constraint A_i (the i-th row of A) is <u>tight</u> if $\langle A_i, x \rangle = b_i$.

Given $x \in P$, let $A^=$ be the submatrix of A formed by the tight constraints.

We say x is a basic solution if $\text{rank}(A^=) = n$ where n is the number of variables.

We usually use ① and ③ in proofs. For completeness, we prove that they are equivalent.

<u>Proposition</u> ① and ③ are equivalent.

<u>proof</u> We will prove $\neg③ \Rightarrow \neg① \Rightarrow \neg③$. Given $x \in P$, let $A^=$ be the set of rows that are tight.

$(\neg③ \Rightarrow \neg①)$: Suppose x is not a basic solution. By definition, the row rank of $A^=$ is less than n.

This implies that the column rank of $A^=$ is less than n, i.e. the columns can not be all linearly independent.

So, $\exists y \neq 0$ such that $A^= y = 0$. Hence, $A^=(x+y) = b = A^=(x-y)$, i.e. tight constraints remain tight.

Therefore, by choosing ε small enough, the non-tight (straight inequality) won't be violated, and thus

we have $A(x+\varepsilon y) \leq b$ and $A(x-\varepsilon y) \leq b$, hence x is not a vertex solution.

$(\neg① \Rightarrow \neg③)$: Suppose x is not a vertex solution. By definition, $\exists y \neq 0$ s.t. $A(x+y) \leq b$ and $A(x-y) \leq b$.

Let $A^=$ be the tight rows of x, i.e. $A^= x = b^=$, where $b^=$ is the corresponding right hand side.

Since $A(x+y) \leq b$ and $A(x-y) \leq b$, we must have $A^=(x+y) \leq b^=$ and $A^=(x-y) \leq b^=$, and this implies

that $A^= y \leq 0$ and $A^=(-y) \leq 0$, and hence $A^= y = 0$.

This implies that the columns of $A^=$ are linearly dependent, and so $\text{rank}(A^=) < n$, thus not basic. □

Okay, now we show that there is always an optimal basic feasible solution.

(From now on, I may also use these terminology interchangably.)

We say the polytope P is <u>bounded</u> if there exists a positive integer M such that if $x \in P$,

then $|x_i| \leq M$ for all $1 \leq i \leq n$. Note that it holds for most combinatorial optimization problems as $0 \leq x_i \leq 1$.

<u>Proposition</u> For bounded P, for any c, $\exists$ basic feasible x such that $cx \leq cy$ for all $y \in P$.

<u>Proof</u> The idea is simple. We move in the tight space until we hit a corner.

If x is not basic, then $\text{rank}(A^=) < n$ and thus the columns of $A^=$ are not linearly independent.

So, $\exists y \neq 0$ such that $A^= y = 0$

Consider $x+\varepsilon y$ and $x-\varepsilon y$. Both solutions have the current tight constraints still tight as $\bar{A}y=0$.

Also, either $c(x+\varepsilon y) \leq cx$ or $c(x-\varepsilon y) \leq cx$ or both, say $c(x+\varepsilon y) \leq cx$.

Set ε to be the maximum value so that $A(x+\varepsilon y) \leq b$.

Since $y \neq 0$ and P is bounded, we know that $\varepsilon < \infty$, in which case we hit a new tight constraint.

So, we can find a solution $x' = x+\varepsilon y$ with one more tight constraint if x is not basic.

This process cannot repeat forever (since we have a finite number of constraints), and hence eventually we will stop at a basic solution x^* with $cx^* \leq cx$ for any x.

In particular, we can set x to be an optimal solution and get an x^* which is optimal and basic. □

Perfect bipartite matching

Let's see a simple example how to show an LP has integral optimal solutions.

Consider the perfect bipartite matching LP:

$$\max \sum_{e \in E} c_e x_e$$

$$\sum_{e \in \delta(v)} x_e = 1$$

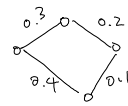
$$0 \leq x_e \leq 1.$$

We claim that a vertex solution must be integral.

Consider a fractional solution x with some $0 < x(uv) < 1$

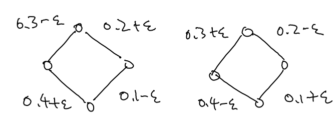
By the constraints $\sum_{e \in \delta(v)} x_e = 1$, we can find vw such that $0 < x(vw) < 1$, and so on.

Continuing until we have a "strictly" fractional cycle



This cycle is even as the graph is bipartite.

Now, we can construct two fractional solutions $x+y \in P$ and $x-y \in P$ by setting and keep the remaining variables unchanged.



Therefore, a vertex solution must be integral.

Next time, we will use the rank argument to prove other results.

Note that the same LP is not integral for perfect matching in general graphs.

Algorithms for solving linear programs

There are three main types of algorithms: Simplex methods, ellipsoid methods, and interior point methods.

We just very briefly mention these algorithms, but highlight a feature of the ellipsoid method.

Simplex methods: Work in the equational form.

Start from an arbitrary basic feasible solution.

Move to a "neighboring" basic solution by removing one column and adding one column.

If there is an "improving" neighbor solution, pick one and go there.

If there is no improving neighbor, stop and return the current solution. This is correct because a local optimal solution in a convex optimization problem is a global optimal solution.

There are many different rules in choosing which improving neighbor to go (some may run into infinite loop in degenate cases). For many natural rules, there are counterexamples showing that they need exponential time (most recently some bad examples were discovered even for randomized rules).

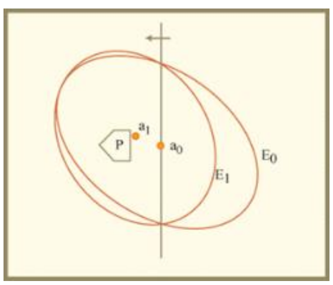
It is still a major open problem whether there are polytime simplex algorithms. Actually, one necessary condition is whether every polytope's diameter is bounded by a polynomial, and it is still wide open.

Should point out that many combinatorial algorithms (e.g. augmenting path algorithms) can be interpreted as a simplex algorithm with a specific rule.

In practice, simplex algorithms seem to be very competitive, and there is a theory of smoothed analysis developed to explain this phenomenon.

<u>Ellipsoid algorithms:</u> It is the first known polynomial time algorithm, discovered by Khachiyan.

First, notice that the optimization problem can be reduced to a decision problem of checking whether P is empty.



Initially, we find a large enough ellipsoid to guarantee to contain the whole polytope P.

At each iteration, check if the center a_i of the current ellipsoid is in P.

If yes, we are done.

If not, the algorithm requires a hyperplane that separates a_i from P.

Given such a hyperplane H, the algorithm would find a smallest ellipsoid E_{i+1} that contains $E_i \cap H$, still guaranteeing that $P \subseteq E_{i+1}$.

Repeat until $\mathrm{vol}(E_i)$ has become too small.

The key of the analysis is to show that the volume of the ellipsoids decreases significantly.

A relatively simple analysis shows that $\mathrm{vol}(E_{i+1})/\mathrm{vol}(E_i) \le e^{-\frac{1}{2(n+1)}}$, so that $O(n)$ iterations will bring it down by a constant factor. Repeat until the ellipsoid is too small to contain P if it is nonempty.

We need to show an upper bound of the initial ellipsoid and a lower bound of the final ellipsoid, details omitted....

Interior point methods : It is another class of polynomial time algorithms for solving LP, first discovered by Karmarkar.

Very roughly, the linear program is reduced to an unconstrained optimization problem using so-called "barrier functions".

Then, we start from the "center" of the polytope and moving toward an optimal point, where in each iteration we use numerical methods to a system of equations (e.g. Newton method).

This is competitive with simplex methods in practice.

In theory, there are recent exciting progress showing fast interior point algorithms for combinatorial optimization problems.

Optimization via separation

A key feature of the ellipsoid method is that it only requires a "separation oracle" for it to work, that is, an algorithm to decide whether $x \in P$; if not, return a separting hyperplane.

In particular, the ellipsoid method does not require us to write down the LP explicitly, and in principle we could solve an exponential size LP as long as there is a polynomial time separation oracle.

Let's see a nontrivial example of solving an exponential size LP.

Consider the spanning tree polytope.

$$\min \sum_{e \in E} c_e x_e$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \quad \forall S \subseteq V$$ where $E(S)$ denotes the set of edges with both vertices in S.

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

$$x_e \geq 0 \qquad \forall e \in E$$

It is not difficult to see that it is a relaxation of the minimum spanning tree problem: for any integral solution, the first class of constraints say that any subset S cannot contain more than $|S|-1$ edges, and this is used to forbid any cycles. The second constraint says we need to choose exactly $n-1$ edges.

So, an acyclic subgraph with $|V|-1$ edges must be a spanning tree.

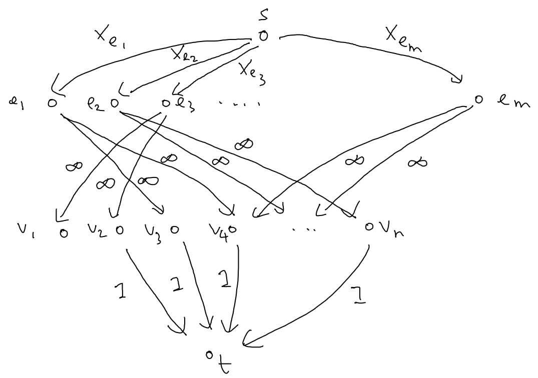
To solve this exponential size LP, we need to design a polynomial time algorithm so that if $x \notin P$, then the algorithm must return a violating constraint.

The constraint $\sum_{e \in E} x_e = |V| - 1$ is easy to check, so we focus on the class $\sum_{e \in E(S)} x_e \leq |S| - 1 \quad \forall S \subseteq V$.

It is nontrivial, but it is not very difficult to understand.

Given an LP solution $x \in \mathbb{R}^{|E|}$, we will run $|V|$ min-cut algorithms, each to check whether there is a violating set containing a specific vertex v_i.

The construction is shown in the following picture:



one vertex for each edge e_i, the capacity of the arc se_i is $x(e_i)$.

one vertex for each vertex v_i, say $e_3 = (v_1, v_2)$, then we add two edges e_3v_1 and e_3v_2, each has capacity ∞.

except for the special vertex, in this case v_1, we have an edge from v_i to t with capacity one.

<u>Claim</u> There is a violating set containing v_1 if and only if the s-t flow value is strictly less than $|V|-1$.

~~proof~~ If there is a violating set S containing v_1 say $S=\{v_1, v_2, \ldots, v_s\}$ such that $\sum_{e \in E(S)} x_e > |S|-1$.

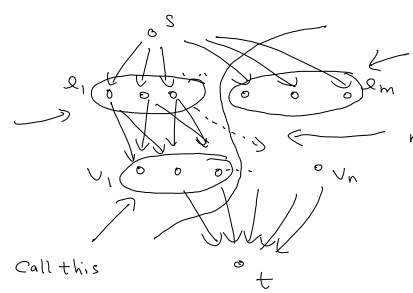
Note that each edge $e \in E(S)$ must send its flow to S, but $\sum_{e \in E(S)} x_e > |S|-1$, and S has only $|S|-1$ capacity to t, and so the capacity in $E(S)$ cannot be sent to the sink t, and thus the flow value would be strictly smaller than $|V|-1$ (recall $\sum_{e \in E} x_e = |V|-1$).

The other direction is more interesting.

If the s-t flow value is strictly less than $|V|-1$, then by the max-flow min-cut theorem, there is a s-t cut of value strictly less than $|V|-1$.

Consider such a cut. It can not cut any edge of capacity ∞. Also, the cut cannot only contain all top edges (since $\sum_{e \in E} x_e = |V|-1$) and cannot contain all bottom edges (since there are $|V|-1$ edges).

So, the cut should look like this:



So, all edges in E_1 must have both vertices in S, and hence $\sum_{e \in E(S)} x_e \geq \sum_{e \in E_1} x_e$

We claim that $\sum_{e \in E_1} x_e > |S|-1$, and this would imply that S is a violating set containing v_1.

To see this, just notice that $|V|-1 >$ s-t cut value

$$= |S|-1 + \sum_{e \in E_2} x_e$$

$$= |S|-1+|V|-1-\sum_{e\in E_1} x_e \quad \left(\text{since } |V|-1=\sum_{e\in E} x_e=\sum_{e\in E_1} x_e+\sum_{e\in E_2} x_e\right)$$

Therefore, $\sum_{e\in E(S)} x_e \geq \sum_{e\in E_1} x_e > |S|-1$, and this completes the proof. □

So, to do this construction by trying every special vertex v_i, if all flow values are $|V|-1$, then x is feasible; otherwise if some flow value is $<|V|-1$, we find a violating constraint.

So, we can use ellipsoid algorithms to solve this exponential size LP in polynomial time.

References

I try to come up with simpler proofs but there may be mistakes, for correct proofs you could take a look at the book "Iterative methods in combinatorial optimization" chapter 2 and chapter 4.

An excellent introduction to LP is the book "Understanding and using linear programming" by Matousek and Gartner. (I actually highly recommend all the books written by Matousek.)

An encyclopedia of combinatorial optimization with an emphasis on polyhedral methods is the book "Combinatorial optimization: polyhedra and efficiency" by Schrijver. It is a 3-volume set with more than 1800 pages, but the proofs are very concise and sometimes with tiny fonts.