Rounding Techniques in Approximation Algorithms

Lecture 10: Convex Geometry, Sparsity, and Scheduling

Lecturer: Nathan Klein

1 Some Basic Convex Geometry

We have already seen some of the power of understanding the structure of the extreme point solutions of linear programs. However, our understanding has for the most part been fairly coarse: either we assume no structure at all, or we prove that the extreme points are integral.

A natural question is whether there is anything in-between. So far, we have seen just one example of this when we proved on the first homework that the vertex cover LP is half-integral. It turns out that although half-integrality is rare, extreme points often have robust structure (other than integrality) which can lead to powerful algorithms. Before we dive into these ideas, we will review some basic convex geometry.

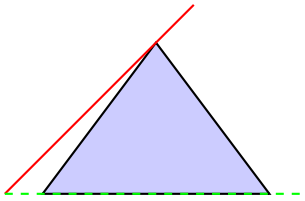

Figure 1: Two supporting hyperplanes. The solid hyperplane intersected with P is a vertex, and the dashed hyperplane intersected with P is an edge.

Hyperplanes, Faces, Vertices, and Edges

Definition 1.1 (Supporting Hyperplane)**.** *A hyperplane $H = \{x \in \mathbb{R}^n \mid a^T x = b\}$ is called a supporting hyperplane of a polyhedron $P \subseteq \mathbb{R}^n$ if $P \cap H \neq \emptyset$ and P is fully contained on one side of H, i.e. either $P \subseteq H^{\leq} = \{x \in \mathbb{R}^n \mid a^T x \leq b\}$ or $P \subseteq H^{\geq} = \{x \in \mathbb{R}^n \mid a^T x \geq b\}$.*

Definition 1.2 (Face, Vertex, and Edge)**.** *A **face** of a polyhedron is P itself or the intersection of P with any supporting hyperplane. A **vertex** of P is a 0-dimensional face of P, and an **edge** is a 1-dimensional face.*

See Fig. 1 for a simple 2-dimensional example. Also, note that a face of a polyhedron is a polyhedron.

Let's also recall the definition of extreme point we have used up until this point:

Definition 1.3 (Extreme Point)**.** *Let $P \subseteq \mathbb{R}^n$ be a polyhedron. Then $x \in P$ is an **extreme point** if there is no non-zero direction $d \in \mathbb{R}^n$ such that $x + d \in P$ and $x - d \in P$.*

We have previously mentioned that this is equivalent to the definition of a vertex. Let's prove it.

Lemma 1.4. *Let $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ be a polyhedron for $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Then, the following are equivalent given a point $y \in P$:*

1. *y is a vertex of P.*
2. *y is an extreme point of P.*
3. *There is a subsystem $\tilde{A}, \tilde{b}$ of n linearly independent constraints defining P such that y is the unique solution to $\tilde{A}x = \tilde{b}$.*

Proof. Suppose y is a vertex. Then, there is a supporting hyperplane $H = \{x \in \mathbb{R}^n \mid a^T x = b\}$ such that $H \cap P = \{y\}$. Now, suppose $y + d \in P$ and $y - d \in P$ for some direction $d \neq \mathbf{0}$. Since $y, y - d$ and $y + d$ are all on the same side of H and $y \in H$, we must have $a^T d = 0$. But then $y + d \in H \cap P$, contradiction. So, (1) implies (2).

Suppose (3) does not hold, and we will show (2) does not hold. Consider the set of tight constraints at y, and consider any maximal linearly independent family of these $\tilde{A}, \tilde{b}$. There must be fewer than n, as otherwise y would be the unique solution to this subsystem. But then, $\tilde{A}x = \tilde{b}$ has a nontrivial kernel, i.e. there exists $d \in \mathbb{R}^n$ such that $\tilde{A}d = 0$. This implies that for some $\epsilon > 0$ we have $y + \epsilon d, y - \epsilon d \in P$ since both points satisfy all tight constraints at y and by taking ϵ small enough we do not violate any of the constraints that were not already tight. So, (2) implies (3).

Finally, suppose y is the unique solution to $\tilde{A}x = \tilde{b}$ for a subsystem of n linearly independent tight constraints. Setting $c = \tilde{A}^T \mathbf{1}$, for any $x \in P$ we have that $c^T x = \mathbf{1}^T \tilde{A} x \geq \mathbf{1}^T \tilde{b}$. This is an equality if and only if $\tilde{A}x = \tilde{b}$. So $c^T x = \mathbf{1}^T \tilde{b}$ is a supporting hyperplane H with $H \cap P = \{y\}$. Therefore, (3) implies (1). □

Notice that the lemma is false if we do not require $y \in P$. While (1) and (2) are equivalent here (as they imply $y \in P$) (3) is not, because (3) can hold for points outside of P. For example, if we have an LP with constraints $0 \leq x_i \leq 1$ for all $1 \leq i \leq n$, then a set of n linearly independent constraints is $0 \leq x_i$ for all i, and (3) would simply be all zeros, which is not in P for any problem we have discussed in this class.

Combined with other facts we have discussed in this course (and not necessarily proved, as our focus is elsewhere), we obtain the following:

Finding a Vertex in Polynomial Time

For any linear program over n variables with constraints $0 \leq x_i \leq 1$ for all i and a polynomial time separation oracle, we can find an optimal solution y in polynomial time that is also the unique solution to a subsystem of n linearly independent constraints met with equality, $\tilde{A}x = \tilde{b}$.

Note this can also easily be generalized to situations where we do not have lower or upper bounds on the variables. Also, it's easy to see that this implies that the *number of fractional coordinates* at a vertex is at most $\text{rank}(\tilde{A}) \leq \text{rank}(A)$ where $\tilde{A}$ is the subset of tight constraints of A, the non-trivial constraints (where a constraint is trivial if it is of the form $0 \leq x_i$ or $x_i \leq 1$). This is sometimes known as the **rank lemma**.

2 Minimizing the Makespan on Unrelated Parallel Machines

The rank lemma is often used to show that *few variables take non-integer values.* We will use this to give a 2-approximation to the problem of minimizing the makespan on unrelated parallel machines, work of Lenstra, Shmoys, and Tardos [LST90].

As input, we are given n machines and m jobs. Each machine i takes time p_{ij} to process job j. We now want to process all jobs in the minimum amount of time T. This is called minimizing the makespan. Notice that we cannot split a job up between different machines.

Let's design an LP. Here's a first guess, where x_{ij} is the relaxed indicator variable of whether machine i process job j:

$$\begin{aligned} \min \quad & T \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_{ij} = 1 && \forall 1 \le j \le m \\ & \sum_{j=1}^{m} p_{ij} x_{ij} \le T && \forall 1 \le i \le n \\ & 0 \le x_{ij} \le 1 && \forall 1 \le i \le n, 1 \le j \le m \end{aligned}$$

The first constraint says every job should be processed once. The second constraint says no machine should run for more than T steps, the objective function.

Unfortunately, this LP has an integrality gap of n. If our input is a single job, and all machines have $p_{ij} = 1$, we can split up the job n ways to obtain $T = 1/n$. But clearly the optimal T is 1. To fix this, we ensure that no single job is larger than T, to obtain the following feasibility problem:

$$P_{UPM}^{T} = \begin{cases} \sum_{i=1}^{n} x_{ij} = 1 & \forall 1 \le j \le m \\ \sum_{j=1}^{m} p_{ij} x_{ij} \le T & \forall 1 \le i \le n \\ x_{ij} = 0 & \forall i, j \mid p_{ij} \ge T \\ 0 \le x_{ij} \le 1 & \forall 1 \le i \le n, 1 \le j \le m \end{cases}$$

Of course, we don't know T off the bat. So, we will run binary search to find the smallest T, T^*, for which P_{UPM}^{T} is non-empty.

It turns out that this new LP has an integrality gap of 2, and leads to a 2-approximation. Let's prove it using sparsity. First, let's obtain a vertex x of $P_{UPM}^{T^*}$ for our optimal T^*. The following is our main sparsity fact, which is a consequence of the fact that there are far more variables than "nontrivial" constraints. It says that most variables must be set to 0 or 1!

Fact 2.1. *Any vertex $x \in P_{UPM}^{T}$ has at most $n + m$ variables for which $0 < x_{ij} < 1$.*

Proof. We have nm variables, so by Lemma 1.4 (or the Rank Lemma), there exists a collection of nm constraints met with equality uniquely defining x. There are $n + m$ constraints that are not of the form $x_{ij} = 0$ or $x_{ij} = 1$. Therefore, $nm - n - m$ constraints must be setting variables to 0 or 1, giving the claim. □

Now consider a bipartite graph G with lefthand side $[n]$, the machines, and righthand side $[m]$, the jobs. Create an edge between machine i and job j for every $0 < x_{ij} \le 1$. We may assume that the graph is a single connected component, as otherwise our LP splits into two disjoint instances

that we can solve separately.[1] But now, we have $n + m$ vertices, at most $n + m$ edges, and the graph is connected. In other words:

Fact 2.2. *Any connected component of G is a tree plus at most one edge.*

Now, delete every job with $x_{ij} = 1$, i.e. delete all the leaves which are jobs. The remaining jobs have degree at least 2. So, all leaves are machines. We will now assign jobs to machines so that every machine gets *at most one* additional job. This would prove that we have a makespan of $2T^*$, since every machine so far has only been assigned jobs with $x_{ij} = 1$ and every edge that exists has $p_{ij} \leq T^*$.

Iteratively delete any machine which is a leaf as well as the job assigned to it. In this way, we can never create a leaf which is a job, as jobs either are deleted or their degree does not change via this process. Ultimately, we are left with no leaves, and we have some $n' + m'$ vertices and at most $n' + m'$ edges since we always delete two vertices and at least two edges. The only possibility is that there are no vertices left (in which case we are done) or the graph is a cycle, and the length of that cycle is even since G is bipartite. Find a perfect matching in this cycle to assign every job to one machine (and vice versa). This demonstrates that our makespan is at most $2T^*$.

Unfortunately, this LP cannot do better than 2.

Fact 2.3. *The integrality gap of this LP is 2.*

Proof. Create one "personal" job for each machine i, assigning $p_{ij} = n - 1$ for each such job and $p_{i'j} = 100n$ for all other machines i'.

Finally, create one "big" job b with $p_{bj} = n$ on all machines.[2] In this way there are $n + 1$ jobs. A feasible LP solution for $T^* = n$ is to assign all $n - 1$ personal jobs to their machines, and then split the final job b up with $x_{bj} = 1/n$ for all machines j. In this way the LP has makespan n.

However, clearly OPT is $2n - 1$, since every personal job must be assigned to that machine to have makespan below $100n$, and the big job must go somewhere. □

The LP solution here is indeed one tree, and after deleting all the jobs with degree 1 (the "personal" jobs), we have a star centered around b.

A major open question is whether a better-than-2 approximation can be designed for this problem.

References

[LST90] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. "Approximation algorithms for scheduling unrelated parallel machines". In: *Mathematical Programming* 46.1 (1990), pp. 259–271. ISSN: 1436-4646. DOI: 10.1007/BF01585745 (cit. on p. 3).

[1] By restricting the LP to each connected component we can show each component forms an extreme point solution to the resulting restricted problem.

[2]