Advanced Algorithms

Lecture 4: Convex Geometry and Unrelated Parallel Machines

Lecturer: Nathan Klein

1 A Polyhedral View of Relax-and-Round

Remember the relax and round framework we studied last time.

Relax and Round

As input we get a discrete optimization problem O. Now, we:

1. Model O as an Integer Linear Program (ILP): linear constraints with the requirement that the solution x is integer.
2. **Relax** the condition to $x \in \mathbb{R}^n$, giving us a Linear Program (LP). Solve the resulting LP using the ellipsoid method to obtain a solution x.
3. **Round** the point x to a solution to O.

It is convenient to think about the ILP and the LP as producing solutions to optimization problems over convex polyhedra.

Definition 1.1 (Convex Polyhedron/Polytope)**.** *A convex polyhedron is the intersection of finitely many half-spaces of the form $a^T x \geq b$ for $a \in \mathbb{R}^n, b \in \mathbb{R}$. A convex polytope is a bounded convex polyhedron. See Fig. 1.*

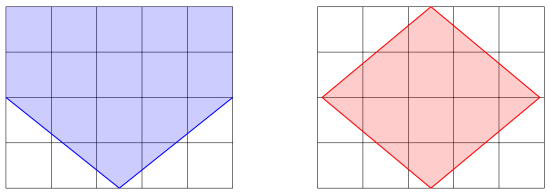
Figure 1: On the left is a convex polyhedron defined by two half-planes, and on the right is a convex polytope defined by four half-planes.

Thus, the set of linear constraints for any ILP form a polyhedron (or a polytope), as do the set of all constraints for an LP. We can therefore think of an ILP/LP pair as follows. First, define a polyhedron P with a separation oracle. Then, the ILP (left) and LP (right) are:

$$\begin{array}{rl} \min & c(x) \\ \text{s.t.} & x \in P \\ & x \in \mathbb{Z}^n \end{array} \qquad\qquad \begin{array}{rl} \min & c(x) \\ \text{s.t.} & x \in P \\ & x \in \mathbb{R}^n \end{array}$$

This highlights the importance of the integrality gap: it gives us the largest possible difference between integer points and real points in a polyhedron. So really, integrality gap is defined with respect to a polyhedron.

Here is an example of the relax-and-round framework geometrically. Suppose the marked points in $\mathbb{Z}^2$ are the feasible solutions to our optimization problem O. Then the blue polytope P is a valid ILP for the problem since $P \cap \mathbb{Z}^2$ is the set of feasible solutions for O.

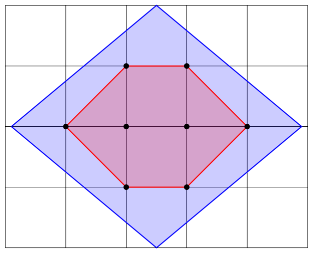

Figure 2: In purple is a polytope P with a separation oracle. In red is the convex hull of the integer coordinates in that polytope. The ILP returns an optimal vertex of the red set. The relaxed LP returns an optimal vertex of the purple set. The integrality gap measures how different these two vertices can be in terms of cost.

If we could find a separation oracle for the red convex set for an NP-Hard problem, we would prove P=NP. So, such a thing likely does not exist, and we have to settle for approximations of the red polytope. Let's redefine the relax-and-round framework with our new language:

Relax and Round: Polyhedral Version

As input we get an instance of a discrete optimization problem O. Now, we:

1. Find a polyhedron P with a polynomial time separation oracle such that $P \cap \mathbb{Z}^n$ is the set of feasible solutions to O.
2. **Relax** our problem: use the ellipsoid method to find an optimal vertex $x \in \mathbb{R}^n$ in P.
3. **Round** the point x to a solution to O.

There are many choices for P, and as we saw in the last lecture, often the most natural choice can be strengthened with additional inequalities. We will explore this further in later lectures.

Notice that in Fig. 2 and in the above definition I say we find an optimal *vertex*. This is because given a polyhedron, if we optimize in any direction, an optimal solution (so long as it is finite) will always be attained at some vertex of that set.

Definition 1.2 (Vertex)**.** *Given a polyhedron P, a point $v \in P$ is a vertex of P if and only if there is no direction $0 \neq d \in \mathbb{R}^n$ such that $v + d \in P$ and $v - d \in P$.*

A nice property of polytopes is the following:

Theorem 1.3 (Carathéodory's Theorem)**.** *Let x lie in the convex hull of a set $S \subseteq \mathbb{R}^n$. Then x can be written as the convex combination of at most $n+1$ points of S.*

It also gives the following fact immediately:

Fact 1.4. *The optimum value of an LP can always be attained at a vertex.*

Proof. Let $a \in P$ be an optimal solution. By Carathéodory, we can write a as a convex combination of vertices. So some vertex has cost at most that of a. □

2 Structure of Vertices

By the above, the optimal solution is always attained at a vertex. The algorithms for LPs we will consider in this course always obtain a vertex solution, which are often called **extreme points** in the convex of an LP. It turns out that extreme points often have robust structure which can lead to powerful algorithms. The reason for this structure is some basic linear algebra.

Lemma 2.1. *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, meaning there is no direction $d \in \mathbb{R}^n$, $d \neq \mathbf{0}$, such that $y + d \in P$ and $y - d \in P$.

(2) There is a subsystem $\tilde{A}, \tilde{b}$ of n linearly independent constraints defining P such that y is the unique solution to the equation system $\tilde{A}x = \tilde{b}$.

Proof. Consider the submatrix $\tilde{A}$ consisting of the tight constraints at y so that $\tilde{A}y = \tilde{b}$ where $\tilde{b}$ is the set of righthand sides.

Suppose (1) does not hold. Then there exists $d \neq \mathbf{0}$ such that $y + d \in P$ and $y - d \in P$. Therefore:

$$\tilde{b} \leq \tilde{A}(y + d) = \tilde{A}y + \tilde{A}d = \tilde{b} + \tilde{A}d$$

which implies $\tilde{A}d \geq \mathbf{0}$. We can similarly derive $\tilde{A}(-d) \geq \mathbf{0}$, meaning $\tilde{A}d = \mathbf{0}$. So, $d \neq \mathbf{0}$ is in the kernel of $\tilde{A}$ implying that its rank is less than n. So (2) cannot hold, meaning (1) implies (2).

Similarly suppose (2) does not hold. Then, $\tilde{A}$ has rank less than n, as otherwise we could pick n linearly independent constraints and satisfy (2). So it has a non-trivial kernel: 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 (1). $\square$

Summarizing the above two sections:

Finding a Vertex in Polynomial Time

For any linear program over a convex polytope with 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, $\tilde{A}x = \tilde{b}$.

You can replace polytope with polyhedron as long as the LP has a finite optimum, but in this course we will always deal with polytopes since we will have the constraints $0 \leq x_i \leq 1$ for all variables i.

3 Minimizing the Makespan on Unrelated Parallel Machines

We will use our new understanding of the structure of vertices 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 3.1. *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 2.1, 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. $\square$

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 $0 < x_{ij} < 1$. 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 3.2. *Any connected component of G is a tree plus at most one edge.*

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

All 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 $p_{ij} \le T^*$.

Iteratively find a machine that is a leaf, and assign it the unique job it's adjacent to. Then delete the machine and the assigned job. 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 3.3. *The analysis of this algorithm cannot be improved. (The "feasibility gap" of this collection of LPs is 2.)*

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

Finally, create one "big" job b with $p_{ib} = n$ on all machines i. 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. □

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).