Lecture 13: Semidefinite programming continued

We present and analyze a rounding algorithm for the SDP relaxation of Max CUT. Then we compare SDP and LP relaxation of max independent set, and say why SDP could be much better. Finally, we see how to write SDP relaxations for different kinds of combinatorial problems.

SDP relaxation for Max Cut

We saw last time the semidefinite programming relaxation for the max cut problem, in which we have a unit vector for each vertex, and the goal is to maximize $\sum_{ij \in E} \frac{1-\langle v_i, v_j \rangle}{2}$. We wanted to show how to round the obtained vector solution to $x_i \in \pm 1$.

We mentioned that this is equivalent to partitioning the surface of the sphere into two parts – the first which maps to $+1$ and the second to -1. We then said that we consider a natural division using a plane through the origin. How do we select the plane so as to maximize the number of cut edges?

The idea is to pick a *uniformly random* hyperplane! The analysis of the size of the cut this gives was done by Goemans and Williamson, who pioneered the SDP approach for Max Cut in their seminal 1994 paper.

For details of the analysis presented in class, please see the following: (be careful with the $\sim$ in the URL)
http://www.cs.utah.edu/∼bhaskara/courses/x968/notes/williamson.pdf

Key steps:

- How well does this do? What is the probability that one edge is cut?

- Expected value of the total cut. How can we relate it to the objective value of the SDP? The approximation ratio we get is 0.878..

We also observe that in the case when the SDP value is close to 1, the approximation factor is better than 0.878. In particular, if the objective value of the SDP is $(1-\epsilon)|E|$, then the rounding produces a cut with value $(1 - O(\sqrt{\epsilon}))|E|$.

SDP vs LP for Independent Set

Let us consider the independent set problem: given a graph $G = (V, E)$, we wish to find the largest possible subset S of vertices such that no pair of vertices in S have an edge.

What is the natural LP formulation for this problem? We can start with one variable x_i that indicates if $i \in S$. Then, for every edge ij, we have

$$x_i + x_j \leq 1.$$

We can think of maximizing $\sum_{i \in V} x_i$ subject to the constraints above, along with $0 \leq x_i \leq 1$.

The issue with this natural linear program is that setting $x_i = 1/2$ for all i is always a feasible solution, and it gives a value $n/2$ for any graph. This is particularly bad when the graph is a clique on n vertices. In this case, the optimal independent set has size 1, while the LP optimum is $n/2$.

Now let us give an SDP formulation, which does much better, at least in this case.

The idea is to have a vector $\mathbf{v}_i$ for every i, which is supposed to be equal to 0 or a *fixed* unit vector $\mathbf{u}$ (i.e., one that is independent of i). We would like to impose constraints that try to enforce this.

For this, let us start with a simpler question: what is a constraint that forces a real number to be 0 or 1? One answer is $x^2 = x$, or equivalently $x \cdot x = x \cdot 1$. We can write precisely this for vectors!

$$\langle \mathbf{v}_i, \mathbf{v}_i \rangle = \langle \mathbf{v}_i, \mathbf{u} \rangle.$$

Here $\mathbf{u}$ is simply constrained to be a unit vector indpendent of i. Geometrically, this is equivalent to saying $\mathbf{v}_i$ lies on the sphere whose center is $\mathbf{u}/2$ and radius $1/2$. (This sphere passes through $\mathbf{u}$ and 0, as desired.) Once we have vectors $\mathbf{v}_i$, there is a more natural way to represent the constraint $x_i + x_j \leq 1$, which is to say $x_i x_j = 0$ for an edge. This can be written as a constraint on the inner products. Thus the overall SDP we consider is the following:

$$\begin{aligned} \max \quad & \sum_i \langle \mathbf{v}_i, \mathbf{v}_i \rangle \quad \text{subject to} \\ \langle \mathbf{v}_i, \mathbf{v}_j \rangle &= 0 \text{ for all } ij \in E, \\ \langle \mathbf{v}_i, \mathbf{v}_i \rangle &= \langle \mathbf{v}_i, \mathbf{u} \rangle \text{ for all } i, \\ \langle \mathbf{u}, \mathbf{u} \rangle &= 1. \end{aligned}$$

It is easy to see that this is a relaxation for the independent set that can be solved efficiently. For the clique, we claim that the optimum of the relaxation is in fact ≤ 1.

To prove this, consider any solution that satisfies all the conditions. In a clique, every pair $i \neq j$ have an edge, which means that $\langle \mathbf{v}_i, \mathbf{v}_j \rangle = 0$ for all $i \neq j$ (i.e., $\mathbf{v}_i$ are mutually orthogonal).

Let S denote the vector $\sum_i \mathbf{v}_i$. This means that

$$\langle S, S \rangle = \langle \sum_i \mathbf{v}_i, \sum_i \mathbf{v}_i \rangle = \sum_i \langle \mathbf{v}_i, \mathbf{v}_i \rangle = \langle \sum_i \mathbf{v}_i, \mathbf{u} \rangle.$$

(We also used the constraints of the SDP.) Thus $\|S\|^2 = \langle S, \mathbf{u} \rangle \leq \|S\| \|\mathbf{u}\|$. This implies that $\|S\| \leq 1$, since $\|\mathbf{u}\| = 1$. This implies that the objective is ≤ 1.

This is an instance in which an SDP relaxation is much better than an LP. For the independent set problem, it turns out that there are instances in which even the SDP does very badly – roughly $\Omega(n)$ larger than the true optimum.

SDP for Coloring 3-colorable graphs

Graph coloring is one of the fundamental problems in theoretical CS. Given a graph, the goal is to color the vertices with as few colors as possible, such that for every edge, the end points are colored with different colors. In other words, every *color class* is an independent set.

It turns out that approximating the minimum number of colors needed for a given class is NP hard (it's called the chromatic number). In fact, *given* that a graph is colorable with 3 colors, it is NP hard to *find* the coloring.

SDP for Coloring 3-colorable graphs continued on next page...

This is referred to as the problem of 3-coloring. Given a graph G that is promised to be 3-colorable, find a coloring using as few colors as possible. It has gained a lot of interest due to the difficulty in obtaining algorithms, as well as proving hardness results. The best algorithm colors the graph using roughly $n^{0.2}$ colors, while the best known hardness results say that it is hard to color using fewer than a constant number of colors.

The best algorithmic results rely on a semidefinite programming formulation, which we briefly outline. Our goal is not to study the problem itself, but just give an example of how to write an SDP for a problem that's quite different from those we've seen so far.

The difference is that it is an optimization problem (minimize the number of colors), but with a *promise*: the input graph is guaranteed to be 3-colorable (we just don't know the coloring). Can a semidefinite program take advantage of this fact?

The idea is to have vectors $\mathbf{v}_i$, one for each vertex, and the *intended solution* is to use exactly three vectors, one for each color class, and the vectors are all at an angle 120 degrees from one another (on a plane). For any edge ij, we know that $\mathbf{v}_i, \mathbf{v}_j$ make an angle of 120 degrees, or equivalently, $\langle \mathbf{v}_i, \mathbf{v}_j \rangle = -1/2$. The nice thing about an SDP formulation is that we can enforce it as a constraint (it just involves inner products). Thus if a graph is 3-colorable, then the following SDP is *feasible*:

$$\langle \mathbf{v}_i, \mathbf{v}_i \rangle = 1 \text{ for all } i, \text{ and } \langle \mathbf{v}_i, \mathbf{v}_j \rangle = -1/2 \text{ for all } ij \in E.$$

Thus, we can start with a collection of vectors that satisfy the constraints, and try to *round*. In this case, this means we need to assign vectors to colors, using as few distinct ones as possible, while ensuring that vertices that have an edge get different colors. This is a bit tricky to do directly, so the known algorithms proceed by trying to find an independent set that is as large as possible (an independent set of size $n/3$ exists in the graph). We can then assign one color to all those vertices, remove them from the graph, write an SDP for the other vertices, and recurse. If we can always find independent sets that are of size $\geq \Delta$, the entire graph will be colored with $O(n/\Delta)$ colors.

We will not go into the formal details.

SDP for 'Betweenness'

Finally, we consider an *ordering* problem, and show how SDP formulations are also useful here. The problem is called *betweenness*. We have a universe of n elements (called $a_1, a_2, \ldots, a_n$, say), and the goal is to arrange them on a line, subject to constraints of the following form: each constraint is defined by a triple (i, j, k), and the ordering we obtain is supposed to have a_j occurring *between* a_i and a_k. Formally, if we denote by π_i the position of a_i, we must have either $\pi_i < \pi_j < \pi_k$, or $\pi_k < \pi_j < \pi_i$.

Suppose we have m such constraints. It is known that it is NP hard to determine if a set of constraints are all satisfiable. Thus it is natural to try to maximize the number of satisfied constraints. How could we write an SDP for such a problem?

The key idea is that a betweenness constraint has a quadratic nature: we can write it as

$$(\pi_i - \pi_j)(\pi_j - \pi_k) > 0.$$

Suppose we write an SDP, whose intended solution is a set of unit vectors that are arranged on a semi-circle

(assume 2 dimensions), in the *correct* order (which we do not know, of course). These vectors $\mathbf{v}_i$ will satisfy

$$\langle(\mathbf{v}_i - \mathbf{v}_j), (\mathbf{v}_j - \mathbf{v}_k)\rangle > 0 \text{ for all constraints } (i, j, k).$$

Now the question is, suppose we have a set of vectors that satisfy the constraint above, can we come up with an ordering π_i that satisfies many of the constraints? How can we come up with an ordering, given a bunch of vectors?

One natural idea is to consider a *random direction* $\mathbf{u}$ (a unit vector), and order the vectors based on the projection onto $\mathbf{u}$. What is the expected number of constraints that are satisfied by the ordering thus obtained? As we did for max cut, we can try to analyze the probability that one of the betweenness constraints is satisfied, and then we can appeal to the linearity of expectation.

Thus consider one constraint, say (i, j, k). Since the projection of $\mathbf{v}_i$ onto $\mathbf{u}$ is simply $\langle \mathbf{v}_i, \mathbf{u}\rangle$, the constraint is satisfied iff

$$(\langle \mathbf{v}_i, \mathbf{u}\rangle - \langle \mathbf{v}_j, \mathbf{u}\rangle)(\langle \mathbf{v}_j, \mathbf{u}\rangle - \langle \mathbf{v}_k, \mathbf{u}\rangle) > 0 \iff \text{sgn}(\langle \mathbf{u}, \mathbf{v}_i - \mathbf{v}_j\rangle) = \text{sgn}(\langle \mathbf{u}, \mathbf{v}_j - \mathbf{v}_k\rangle).$$

Now, by the constraints in the SDP, we know that the inner product of $\mathbf{v}_i - \mathbf{v}_j$ and $\mathbf{v}_j - \mathbf{v}_k$ is ≥ 0, which means these vectors make an angle $\leq \pi/2$. Thus for a random direction $\mathbf{u}$, the probability of the above event is $\geq 1/2$.

This implies that the expected number of satisfied constraints is $\geq m/2$. Note that this is non-trivial, because if we consider a *uniformly random* ordering, only $m/3$ constraints are satisfied in expectation. Thus the SDP allows us to do significantly better.