Saarland Informatics Campus

Script of Lecture 13, Approximation Algorithms
Summer term 2017

Tobias Mömke, Hang Zhou
`http://www-cc.cs.uni-saarland.de/course/61/`

Written by Tobias Mömke

Semidefinite Programming

1 Background

Most of the algorithms you have seen in this class are based on linear programming. Recall that the general idea is to obtain a good lower bound that we compare with our algorithmic solution.

But what if we would allow something more general than a linear objective and linear constraints? We could, for instance, think of allowing quadratic constraints. But this would be too much to ask for. Consider the constraints $x_i^2 = x_i$. These constraints are satisfied if and only if $x_i \in \{0, 1\}$. Therefore, if we could handle quadratic constraints, we could also solve, e.g., TSP or the Steiner tree problem and thus solving quadratic programs is NP-hard.

What we are aiming for is something in between. To this end let us see why we cannot solve quadratic constraints by using the ellipsoid method. We can solve (minimize) *convex problems*, that is, problems formulated like an LP, but with convex objective function and convex constraints. Here we have a problem, since we have to normalize the equality.

- $x^2 \geq x$ is convex, since we have $x^2 - x \geq 0$ and $f(x) = x^2 - x$ is a convex function.
- $-x^2 \geq -x$ is not convex, since $-x^2 - x$ is not a convex function.

2 Semidefinite Programming Basics

Instead of a vector of variables $x = (x_1, x_2, \ldots, x_n)$, let us consider a matrix X with entries $x_{i,j}$. We now get an advantage by restricting the properties of X, but we have to make sure that we can solve the obtained mathematical program. For our purpose, we require that X is a symmetric positive semidefinite matrix.

Definition 1. A matrix $X \in \mathbb{R}^{n \times n}$ is positive semidefinite (psd) iff for all $y \in \mathbb{R}^n$, $y^T X y \geq 0$.

To express that X is psd, we write $X \succeq 0$. For symmetric X we obtain the following equivalences.

Lemma 1. *If $X \in \mathbb{R}^{n \times n}$ is a symmetric matrix, then the following statements are equivalent.*

a) X is psd;

b) all eigenvalues of X are non-negative;

c) $X = V^T V$ for a matrix $V \in \mathbb{R}^{m \times n}$ with $m \leq n$;

d) $X = \sum_{i=1}^{n} \lambda_i w_i w_i^T$ for some values $\lambda_i \geq 0$ and vectors $w_i \in \mathbb{R}^n$ such that $w_i^T w_i = 1$ and $w_i^T w_j = 0$ for $i \neq j$.

We will not prove the equivalence here, since we want to focus on the aspects related to approximation algorithms.

We are now ready to write down an example of a semidefinite program. Let X be a symmetric matrix that holds the variables $x_{i,j}$.

$$\begin{aligned}
\text{minimize or maximize} \quad & \sum_{i,j} c_{i,j} x_{i,j} && (1)\\
\text{s.t.} \quad \sum_{i,j} a_{i,j,k} x_{i,j} &= b_k && \text{for all } k\\
x_{i,j} &= x_{j,i} && \text{for all } i,j\\
X &\succeq 0 && (2)
\end{aligned}$$

Observe that the only difference to a linear program is the last constraint that X is psd. The reason why we can apply the ellipsoid method is that we can see X itself as a point in a vector space: the set of all psd matrices forms a convex cone. We note that due to some technicalities, we cannot obtain an optimal solution in polynomial time, but we can obtain a $1+\epsilon$ approximation.

To use semidefinite programming algorithmically, it is more convenient to move to an equivalent formulation called vector program. In a vector program, the variables do not hold single values, but n-vectors. For vectors $v_i, v_j \in \mathbb{R}^n$, we sometimes write $v_i \cdot v_j$ as a simplified notation of the inner product $v_i^T v_j$.

$$\begin{aligned}
\text{minimize or maximize} \quad & \sum_{i,j} c_{i,j} (v_i \cdot v_j) && (3)\\
\text{s.t.} \quad \sum_{i,j} a_{i,j,k} (v_i \cdot v_j) &= b_k && \text{for all } k\\
v_i &\in \mathbb{R}^n && i = 1, \ldots, n
\end{aligned}$$

The semidefinite program and the vector program above are equivalent because of Lemma 1: X is psd iff $X = V^T V$ for some matrix V. Given a solution X, since it is a symmetric psd matrix we can decompose it into $V^T V$. Let v_i and v_j be the ith and jth column of V. Then $x_{i,j} = v_i^T v_j = v_i \cdot v_j$ and the vectors in V give a feasible solution to the vector program.

The other way around, if V is a solution to the vector program, $V^T V$ is a symmetric psd matrix and we obtain a feasible solution by setting $x_{i,j} = v_i \cdot v_j$.

3 Max Cut

We now show how to use SDPs to find large cuts in graphs. The problem Max Cut is defined as follows. We are given an undirected graph G with non-negative edge weights w. Then each set of vertices $S \subseteq V(G)$ defines a cut $\delta(S)$. We aim to find a set S that maximizes $w(\delta(S)) = \sum_{e \in \delta(S)} w_e$.

There is a simple randomized 2-approximation algorithm for Max Cut: include each vertex v with probability $1/2$. Then one can verify that each edge is in the cut with probability $1/2$ and we obtain a cut of expected size $w(E(G))/2$.

Using SDPs, we can do better. We formulate Max Cut as follows.

$$\begin{aligned}
\text{maximize} \quad & \sum_{\{i,j\} \in E(G)} \frac{w_{ij}(1 - y_i y_j)}{2} && (4)\\
\text{s.t.} \quad & y_i \in \{-1, 1\} && i = 1, \ldots, n
\end{aligned}$$

To see that the mathematical program (4) solves Max Cut, observe that $(1 - y_i y_j)/2$ is either zero or one. It is zero if either both variables have the value -1 or both have the value 1. It is one if both variables have distinct values.

We now relax the formulation to a vector program.

$$
\begin{aligned}
\text{maximize} \quad & \sum_{\{i,j\} \in E(G)} \frac{w_{ij}(1 - v_i \cdot v_j)}{2} & & \\
\text{s.t.} \quad & v_i \cdot v_i = 1 & & i = 1, \ldots, n \\
& v_i \in \mathbb{R}^n & & i = 1, \ldots, n
\end{aligned}
\tag{5}
$$

To see that this seemingly more complicated formulation is indeed a relaxation, it is useful to observe that a solution to (4) implies a solution to (5) by setting $v_i = (0, 0, \ldots, 0, y_i)$ for each i.

Since (5) is a vector program, we can compute an almost optimal solution in polynomial time. To round it, let us first have a closer look at the vectors v_i.

The constraint $v_i \cdot v_i = 1$ implies that the length of v_i is one:

$$v_i \cdot v_i = \sqrt{v_i \cdot v_i} = \sqrt{v_{1i}^2 + v_{2i}^2 + \cdots + v_{ni}^2} = ||v_i||_2 \, .$$

We can interpret the solution geometrically: all solution vectors lie on the unit sphere $\mathbb{S}^n$.

To round the solution for (5), we will cut $\mathbb{S}^n$ into two halves. We want to choose a random hyperplane for the cut. To achieve this, we first select a random vector $r = (r_1, r_2, \ldots, r_n)$. We select r by drawing each entry from the normal distribution $\mathcal{N}(0, 1)$ (with mean value 0 and variance 1).

Geometrically, r determines a hyperplane which is the vector space of vectors orthogonal to r. Without considering the geometric intuition, we obtain a simple algorithm: For each vector v_i, compute $v_i \cdot r$. Then vertex i is in S if and only if $v_i \cdot r \geq 0$. We want to show the following statement.

Theorem 1. *The algorithm above is a* 0.878*-approximation algorithm for Max Cut.*

Before we formally proof the approximation ratio of this algorithm, let us first see why the approach should be a good idea in the first place.

The inner product between unit vectors represents the angle between the vectors: if $v_i \cdot v_j = 1$, both vectors point into the same direction. The other extreme is $v_i \cdot v_j = -1$ which means that they point into opposite directions. $v_i \cdot v_j = 0$ means that they are orthogonal.

Now the vector program tries to maximize the angle between vectors that belong to vertices that should be in opposite sides of the cut: suppose an edge e has an extremely high weight, larger than all other weights together. Then in (5), $v_i \cdot v_j$ should be close to -1 in order to maximize the objective.

To analyze the algorithm, we assume two technical facts. Observe that the sign of the inner product does not depend on the length of the vectors.

Fact 1. *The vector $r/||r||$ is uniformly distributed over $\mathbb{S}^n$.*

Fact 2. *The projections of r onto two unit vectors e_1 and e_2 are independent and are normally distributed with mean* 0 *and variance* 1 *iff e_1 and e_2 are orthogonal (i.e., $e_1 \cdot e_2 = 0$).*

Corollary 1. *Let r' be the projection of r onto a* 2*-dimensional plane P. Then $r'/||r'||$ is uniformly distributed on a circle in P.*

Intuitively this means that choosing r the way we did give a random hyperplane that separates vectors with a probability determined by the angle between the vectors.

Lemma 2. *The probability that edge (i, j) is in the cut is* $\arccos(v_i \cdot v_j)/\pi$.

Proof. Let us consider the plane P spanned by v_i and v_j. By Corollary 1, r projected on P and normalized to unit length is uniformly distributed on the unit circle of P. We only have to consider the projection r' of r onto P: $r = r' + r''$ with r'' orthogonal to v_i and v_j, i.e.,

$$v_i \cdot r = v_i \cdot (r' + r'') = v_i \cdot r' + v_i \cdot r'' = v_i \cdot r' \,.$$

Analogously, $v_j \cdot r = v_j \cdot r'$.

Let θ be the angle between v_i and v_j. Then also the angle between the lines orthogonal to v_i and v_j is θ.

We distinguish the different inner products. We have $v_i \cdot r \geq 0$ and $v_j \cdot r < 0$ if r points between the two orthogonal lines on the side of v_i, which happens with probability $\theta/(2\pi)$. (This becomes clear when drawing a picture.) Analogously, $v_i \cdot r < 0$ and $v_j \cdot r \geq 0$ happens with probability $\theta/(2\pi)$. Otherwise i and j are either both in S or none of them is. Thus $\{i, j\}$ is in the cut with probability θ/π.

The relation to the inner product is that $v_i \cdot v_j = ||v_i||\,||v_j|| \cos\theta = \cos(\theta)$, and therefore $\theta = \arccos(v_i \cdot v_j)$. Dividing by π gives the claim. □

Using calculus, Lemma 2 implies the following.

Lemma 3. *For* $x \in [-1, 1]$, $\arccos(x)/\pi \geq 0.878 \cdot (1 - x)/2$.

To finish the proof of Theorem 1, let us analyze the expected value of the cut. Let X_{ij} be a random variable with $X_{ij} = 1$ if $\{i, j\}$ is in the cut and $X_{ij} = 0$ otherwise. Let $W := \sum_{i,j\colon \{i,j\} \in E(G)} w_{ij} X_{ij}$. Then by Lemma 2,

$$E[W] = \sum_{i,j\colon \{i,j\} \in E(G)} w_{ij} \cdot \Pr[\{i, j\} \in \delta(S)] = \sum_{i,j\colon \{i,j\} \in E(G)} w_{ij} \arccos(v_i \cdot v_j)/\pi \,.$$

Then Lemma 3 implies

$$E[W] \geq 0.878 \sum_{i,j\colon \{i,j\} \in E(G)} w_{ij}(1 - v_i \cdot v_j)/2 \geq 0.878\beta \,,$$

where β is the weight of an optimal solution to the vector program (5). This implies the theorem.

References

[1] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

The sections 2 and 3 are based on the book of Williamson and Shmoys [1]