U.C. Berkeley — CS270: Algorithms
Professor Vazirani and Professor Rao

Lecture 22
Scribe: Anupam
Last revised

Lecture 22

1 Semi Definite Programming

Semi-definite programming is a generalization of linear programming that has been found to be useful for designing better approximation algorithms for combinatorial optimization problems. We will see examples of semi definite programs, discuss the reason SDPs can be solved in polynomial time and present the classical Goemans Williamson algorithm for the Max-cut problem.

Recall that a symmetric matrix A is said to be positive semidefinite (denoted as $A \succeq 0$) if all the eigenvalues are non negative, or equivalently if $x^T A x \geq 0$ for all $x \in \mathbb{R}^n$. The decomposition of A in the spectral basis shows that every positive semidefinite matrix has a square root,

$$A = \sum_i \lambda_i v_i v_i^T \Rightarrow A^{1/2} = \sum_i \sqrt{\lambda_i} v_i v_i^T \tag{1}$$

The existence of the square root shows that A can be written as $B^T B$, in general there are several possible B such that $A = B^T B$. A matrix that can be written as $B^T B$ is positive semidefinite as $x^T B^T B x = |Bx|^2 \geq 0$. Instead of the square root, it is convenient to work with the Cholesky decomposition $A = B^T B$ where B is upper triangular, this decomposition is unique and is computed using Gaussian elimination.

Interpreting the decomposition geometrically, the entry a_{ij} of a positive semidefinite matrix can be expressed as an inner product $v_i.v_j$ for some vectors $v_i \in \mathbb{R}^n$. A semi-definite program optimizes a linear objective function over the entries of a positive semidefinite matrix, the constraints are linear functions of the matrix entries,

$$\begin{aligned} \max \quad & A.C \\ A.X_i \quad & \geq \quad b_i \\ A \quad & \succeq \quad 0 \end{aligned} \tag{2}$$

The matrix dot product $A.X$ in the representation of the SDP is the inner product between matrix entries $\sum_{ij} a_{ij} x_{ij}$, the matrix inner product is also equal to $Tr(AX)$ the trace of the matrix product.

Why does SDP generalize LP? A linear program is a special instance of the SDP in (2) where additional constraints are added to ensure that the non diagonal entries of A are equal to 0.

Solving an SDP in polynomial time: Consider the set of all positive semidefinite matrices as a subset of $\mathbb{R}^{n^2}$, if the matrices A, B are positive semidefinite then $\alpha A + \beta B$ is also positive semidefinite for $\alpha, \beta > 0$. The set of positive semidefinite matrices is therefore a convex cone.

The ellipsoid algorithm is a general purpose algorithm to optimize a linear function $w.x$ over a convex region C. The strategy is to first use the binary search trick to reduce optimization to feasibility testing, in order to find the optimum value of $w.x$ over C we check if the intersection of C with the hyperplane $w.x = \alpha$ is non empty.

The feasibility problem is solved by finding an ellipsoid enclosing $C \cap (w.x = \alpha)$, the initial ellipsoid is the projection of an ellipsoid enclosing C onto the hyperplane $w.x = \alpha$. If the ellipsoid contains a point from C the value α is feasible. In each iterative step we check if the center of the current ellipsoid belongs to C, if not then we can find a hyperplane through the center of the ellipsoid such that C is on one side of the hyperplane. The search problem therefore reduces to checking that 'half' the ellipsoid is non empty.

It can be shown that a 'half' ellipsoid can be enclosed in a circumscribing ellipsoid of volume $V(1-1/\text{poly}(n))$ where V is the volume of the original ellipsoid and n is the problem size. We repeat the algorithm with the circumscribing ellipsoid, within poly(n) iterations the feasibility problem is solved as the volume of the ellipsoid approaches 0.

The ellipsoid algorithm is not practical to implement, but shows that SDPs can be solved in polynomial time. Faster SDP solvers combine the ellipsoid method with dimension reduction, also a version of the experts algorithm can be used to approximately solve SDPs.

1.1 The Maximum Cut problem

The max cut problem is to partition a graph $G(V, E)$ into two pieces $(S, \overline{S})$ such that the weight of the edges cut is maximized. If OPT is the weight of the maximum cut, there is a simple randomized algorithm that produces cuts with weight at least $OPT/2$. The algorithm constructs set S by independently adding vertices $v \in G$ to S with probability $1/2$. The indicator random variable $I_e = 1$ if the edge e is cut and 0 otherwise, by linearity of expectation we have,

$$E[w(S, \overline{S})] = \sum_e E[w(e)I_e] = \sum_e \frac{w_e}{2} \geq \frac{OPT}{2} \tag{3}$$

The expected weight of the cut produced by the randomized algorithm is half the total weight, if the variance is large cuts with high weight would be produced frequently. The variance can be shown to be small for random graphs, also it is easy to derandomize the algorithm.

The simple algorithm was the best known for a long time, in order to achieve an improvement let us write the max cut problem as an integer program,

$$\max \sum_{(i,j) \in E} w_{ij} \frac{1 - x_i.x_j}{2} \qquad \forall i, x_i \in \{1, -1\} \tag{4}$$

As the integer program is NP hard to solve exactly we look for relaxations of the program that are easier to solve. Instead of one dimensional unit vectors, optimizing over vectors in n dimensional space we have the program,

$$\max \sum_{(i,j) \in E} w_{ij} \frac{1 - v_i.v_j}{2}$$
$$\forall i, v_i \in \mathbb{R}^n, \qquad |v_i| = 1 \tag{5}$$

The program is a semi definite program as the objective function and constraints are linear in the inner products $v_i.v_j$. The optimal value of the program is denoted by $VP(OPT)$, the solution vectors $v_1, v_2, \cdots, v_n \in \mathbb{R}^n$ can be found by computing the Cholesky decomposition of the matrix A output by the SDP solver.

As an example, let us consider the 5 cycle, the size of the maximum cut is 4 while the optimal solution to the vector program is two dimensional and corresponds to the embedding of the five star. The optimum value for the relaxed program is $\frac{5(1-\cos(4\pi/5)}{2} = 4.52$.

The one dimensional solution corresponding to the maximum cuts is a solution to the relaxed problem, the value $VP(OPT)$ is therefore greater than OPT. We will show that starting with a solution $v_1, v_2, \cdots, v_n$ to the vector program a cut with value $0.878VP(OPT)$ can be found,

$$0.878VP(OPT) \leq OPT \leq VP(OPT) \tag{6}$$

Select a random hyperplane $w.x = 0$ through the origin and define $S := \{i \mid w.v_i \geq 0\}$ to be the set of points that lie on one side of the hyperplane.

Claim 1
The expected weight of the cut $(S, \overline{S})$ *is at least* $0.878VP(OPT)$.

Proof: The analysis of the rounding procedure is local, we consider the contributions of the edge (i, j) to the SDP solution and to the randomized cut. If the angle between vectors v_i, v_j is θ then restricting ourselves to the 2-dimensional plane spanned by v_i and v_j it follows that the probability that the edge $(i, j) \in (S, \overline{S})$ is $\frac{\theta}{\pi}$.

The contribution of edge (i, j) to the objective value of the vector program is $\frac{1-\cos\theta}{2}$. The ratio of the expected weight of the cut $(S, \overline{S})$ to the objective value of the semi-definite program is given by,

$$\frac{E[w(S, \overline{S})]}{VP(OPT)} = \sum_e \frac{2w_e\theta}{w_e(1 - \cos(\theta))} \geq \min_{\theta \in [0,\pi]} \frac{2\theta}{1 - \cos\theta} = 0.878 \tag{7}$$

The minimum value of $\frac{2\theta}{1-\cos\theta}$ over angles $\theta \in [0, \pi]$ is attained for $\theta \approx 137.6°$ and is equal to 0.878. □

Semidefinite programming therefore provides a 0.878 approximation for max cut, it might seem that the number is not significant but if the unique games conjecture in complexity theory is true then it is not possible to obtain a better approximation algorithm. The above edge by edge of the SDP rounding procedure is optimal, a tight example can be constructed by taking several n dimensional unit vectors and adding the edge (i, j) if the angle between v_i and v_j is $137.6° \pm \epsilon$.