CS880: Approximations Algorithms

Scribe: Chi Man Liu **Lecturer:** Shuchi Chawla

Topic: Bourgain's Embedding into ℓ_1, Semi-Definite Programming **Date:** 3/22/2007

In the previous lecture, we saw how to approximate Sparsest Cut given a low distortion embedding from arbitrary metrics into ℓ_1. In the first part of this lecture, we present such an embedding with $O(\log n)$ distortion [1], and hence a $O(\log n)$-approximation for Sparsest Cut [2]. In the second part of this lecture, we introduce a generalization of linear programming known as semi-definite programming (SDP), and give an SDP relaxation for the Max-Cut problem [3].

19.1 Bourgain's Embedding into ℓ_1

Last time we showed that Sparsest Cut could be approximated by interpreting the problem as an LP over ℓ_1-metrics. In this section, we show how to embed an arbitrary metric over a set of n points into an ℓ_1 metric with $O(\log n)$ distortion [1]. An immediate result would be a $O(\log n)$-approximation for Sparsest Cut [2].

Let V be a set of n points and d be a metric over V. We want to find an embedding of d into $\mathbb{R}^k$ for some k with distortion $O(\log n)$, i.e. we want to find an embedding $f : V \to \mathbb{R}^k$ such that there exists $\alpha, \beta > 1$ with $\alpha\beta = O(\log n)$ and for all $x, y \in V$,

$$\begin{aligned}\ell_1(f(x), f(y)) &\leq \alpha \cdot d(x, y) \text{ and}\\ \ell_1(f(x), f(y)) &\geq \frac{1}{\beta} \cdot d(x, y).\end{aligned}$$

19.1.1 The Algorithm

In our construction, we use *Fréchet embeddings*, which map general metric spaces to normed metric spaces as follows. Let (V, d) be a metric space. Suppose that we want to embed this metric space into ℓ_1 over $|V|$ points in $\mathbb{R}_k$. Then, for the i^{th} coordinate ($1 \leq i \leq k$), we pick a subset $A_i \subseteq V$, and set $f_i(x) = d(x, A_i)$ for all $x \in V$, where $d(x, A_i) = \min_{y \in A_i} d(x, y)$.

We are going to map V to n points in $\mathbb{R}^{M_1 M_2}$, where $M_1 = \lceil \log_2 n \rceil$ and M_2 is a constant multiple of $\log n$ to be determined later. The algorithm is as follows.

```
(1) For $i = 1, \ldots, M_1$,
(2)     For $j = 1, \ldots, M_2$,
(3)         Form the set $A_{ij}$ by picking every $x \in V$ with probability $2^{-i}$.
(4) For all $x \in V$, set $f(x) = (f_{11}(x), f_{12}(x), \ldots, f_{M_1 M_2}(x))$, where $f_{ij}(x) = d(x, A_{ij})$.
```

We give an intuition for the above algorithm. Consider a particular coordinate. Arrange the points in V on the real number line according to their coordinates. For any two points, we do not want their distance on the number line to be too large, otherwise the expansion of the embedding would be large. In fact, using the triangle inequality, we can conclude that the distance will never be too large: $|f_{ij}(x) - f_{ij}(y)| = |d(x, A_{ij}) - d(y, A_{ij})| \leq d(x, y)$. Likewise, we do not want them too close, since we want to keep the contraction small. One way to achieve this is to ensure that $d(x, A_{ij})$ is quite large and $d(y, A_{ij})$ is relatively small (or vice versa). That is, we want A_{ij} to include at least one point close to y, but no point close to x. If the set A_{ij} is too large, it may include points close to x, and if it is too small, it may not include any point close to y. This is why we use randomness in our algorithm: we hope that the overall distortion will be small by randomly picking sets of varying sizes (depending on the value of i).

19.2 Analysis

We need to show that our algorithm gives a $O(\log n)$-distortion embedding with reasonbly high probability. The following lemma bounds the expansion of f.

Lemma 19.2.1 *The expansion of f is at most $M_1 M_2$.*

Proof: Pick any $x, y \in V$ and look at one coordinate. By the definition of f and the triangle inequality, we have

$$|f_{ij}(x) - f_{ij}(y)| = |d(x, A_{ij}) - d(y, A_{ij})| \leq d(x, y).$$

Thus, the ℓ_1 distance between $f(x)$ and $f(y)$ is bounded by

$$\sum_{i=1}^{M_1} \sum_{j=1}^{M_2} |f_{ij}(x) - f_{ij}(y)| \leq M_1 M_2 \cdot d(x, y).$$

■

The following lemma bounds the contraction of f. The proof of this lemma is deferred.

Lemma 19.2.2 *The contraction of f is $O(\frac{1}{\log n})$, i.e. for all $x, y \in V$, $\ell_1(f(x), f(y)) = \Omega(\log n) \cdot d(x, y)$.*

Our main theorem follows from Lemma 19.2.1 and Lemma 19.2.2 directly.

Theorem 19.2.3 *There exists an efficiently computable $O(\log n)$-distortion embedding from (V, d) into $(\mathbb{R}^k, \ell_1)$, where $k = O(\log^2 n)$.*

Before we prove Lemma 19.2.2, we need a few definitions.

Definition 19.2.4 *For any $x \in V$ and $r \geq 0$, we denote by $B(x, r)$ the (closed) ball centered at x with radius r, i.e. $B(x, r) = \{y \in V \mid d(x, y) \leq r\}$. For any $x \in V$ and non-negative integer i, let $r_i(x)$ be the smallest r such that $|B(x, r)| \geq 2^i$.*

We now proceed to the proof of Lemma 19.2.2.

Proof: [Proof of Lemma 19.2.2] Fix $x, y \in V$. For any i, let $\rho_i = \max\{r_i(x), r_i(y)\}$. Note that $|B(x, \rho_i)| \geq 2^i$ and $|B(y, \rho_i)| \geq 2^i$. Let t be the smallest index such that $\rho_t \geq \frac{1}{2} d(x, y)$. Consider $\rho_0, \rho_1, \ldots, \rho_t$. If $\rho_t + \rho_{t-1} > d(x, y)$, we redefine $\rho_t = d(x, y) - \rho_{t-1}$. By doing this, we ensure that $B(x, \rho_{i-1})$ and $B(y, \rho_i)$ are disjoint (but they can touch) for $i = 1, \ldots, t$. We want to show that for any i and j, there is a good chance that A_{ij} contains a point near x but no point near y. This effectively bounds the contraction of f between x and y. This is formalized in the following claim.

Claim 19.2.5 *Let $1 \leq i \leq t$. Let $S_i = \{j \mid |f_{ij}(x) - f_{ij}(y)| \geq \rho_i - \rho_{i-1}\}$. Then there exists a constant $c > 0$ such that $\mathbf{Pr}[|S_i| \geq c \log n] \geq 1 - n^{-3}$.*

Proof: [Proof of Claim 19.2.5] Without loss of generality, suppose that $\rho_i = r_i(y)$. Let the i^{th} *good ball* G_i be $B(x, \rho_{i-1})$, and the i^{th} *bad ball* B_i be the **open** ball centered at y with radius ρ_i, i.e. $B_i = \{v \in V \mid d(y, v) < \rho_i\}$. Note that $|G_i| \geq 2^{i-1}$ and $|B_i| \leq 2^{i-1}$ (since B_i is open).

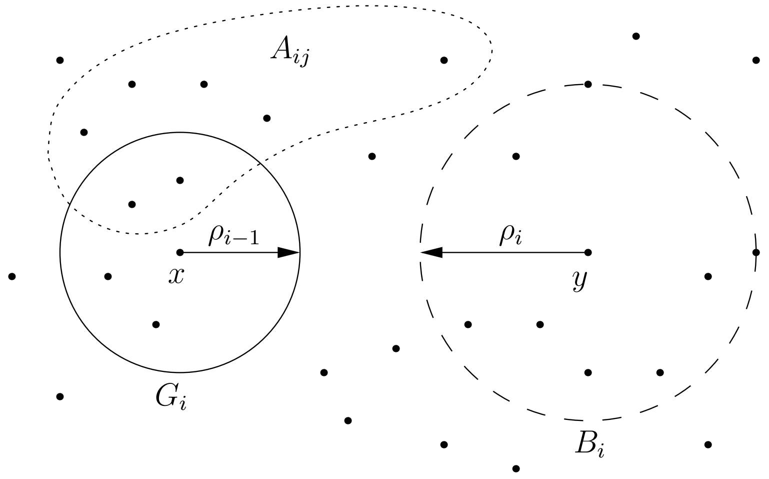

Figure 19.2.1: The i^{th} good ball G_i, the i^{th} bad ball B_i, and the set A_{ij}.

Fix j. We bound the probability that some point in A_{ij} lies in G_i but none lies in B_i as follows.

$$\begin{aligned}
& \mathbf{Pr}[A_{ij} \cap G_i \neq \emptyset \text{ and } A_{ij} \cap B_i = \emptyset] \\
= \; & \mathbf{Pr}[A_{ij} \cap G_i \neq \emptyset] \cdot \mathbf{Pr}[A_{ij} \cap B_i = \emptyset] && \text{(by independence, because } G_i \text{ and } B_i \text{ are disjoint)} \\
\geq \; & \left(1 - \left(1 - \frac{1}{2^i}\right)^{2^i/2}\right) \left(1 - \frac{1}{2^i}\right)^{2^i} \\
\geq \; & (1 - e^{-1/2}) \cdot \frac{1}{4}.
\end{aligned}$$

Thus, the above probability is at least some constant. Note that $A_{ij} \cap G_i \neq \emptyset$ implies that $f_{ij}(x) \leq \rho_{i-1}$, and $A_{ij} \cap B_i = \emptyset$ implies that $f_{ij}(y) \geq \rho_i$. Hence, it follows from the above that $\mathbf{Pr}[|f_{ij}(x) - f_{ij}(y)| \geq \rho_i - \rho_{i-1}]$ is at least some constant.

For each j, let Z_j be an indicator random variable taking the value 1 if and only if $|f_{ij}(x) - f_{ij}(y)| \geq \rho_i - \rho_{i-1}$. Then by linearity of expectation, we have $\mathbf{E}\left[\sum_{j=1}^{M_2} Z_j\right] \geq c' \cdot M_2$ for some constant $c' > 0$. By choosing M_2 (recall that M_2 is a constant times $\log n$ and we have not picked the constant yet) and a suitable constant $c'' > 0$ and applying the Chernoff bound[1], we have

$$\mathbf{Pr}\left[\sum_{j=1}^{M_2} Z_j < c \cdot M_2\right] < e^{-\frac{c'' \cdot M_2}{3}} = \frac{1}{n^3},$$

where c depends on c', c'' and M_2. Recall that $M_2 = \Theta(\log n)$ and $\sum_{j=1}^{M_2} Z_j = |S_i|$. This proves our claim. ■

Let c be the constant in Claim 19.2.5. For every i, we call it *good* if $|S_i| \geq c \cdot \log n$, and *bad* otherwise. If all i's are good, then

$$\begin{aligned}\sum_{i=1}^{t}\sum_{j=1}^{M_2} |f_{ij}(x) - f_{ij}(y)| &\geq \sum_{i=1}^{t} \Omega(\log n)(\rho_i - \rho_{i-1})\\ &\geq \Omega(\log n)\rho_t\\ &\geq \Omega(\log n) d(x,y).\end{aligned}$$

Thus, the contraction of f is $O(\frac{1}{\log n})$. It remains to show the probability that all i's are good is not too small. By union bound, we have

$$\mathbf{Pr}[\text{there exists a bad } i] \leq \frac{1}{n^3}\log n < \frac{1}{n^2}.$$

Hence, for fixed $x, y \in V$, all i's are good with probability at least $1 - n^{-2}$. We say that a pair (x, y) *fails* if not all i's are good. By union bound, we get

$$\mathbf{Pr}[\text{there exists a pair } (x,y) \text{ which fails}] \leq \frac{1}{2}.$$

We can repeat the algorithm until the contraction of f is $O(\frac{1}{\log n})$. The expected number of times we need to run the algorithm is constant. ■

Note. It can be shown that f is a $O(\log n)$-distortion embedding into ℓ_p for general p using a similar analysis. Also, there are metrics which cannot be embedded into ℓ_1 with distortion $o(\log n)$, e.g. expander graph metrics.

19.3 Semi-Definite Programming

One downside of linear programming is that it cannot capture nonlinear constraints. In this section, we introduce semi-definite programming, which is capable of capturing a specific form of nonlinear constraints that turns out to be useful in some formulations.

19.3.1 Definitions

A *semi-definite program* (SDP) can be thought of as a linear program with an additional "semi-definiteness" constraint. More specifically, an SDP is a mathematical program with the following elements:

[1]The Chernoff bound we use here is $\mathbf{Pr}[\sum Z_j < (1-\epsilon)\mathbf{E}[\sum Z_j]] < \exp(-\epsilon^2 \mathbf{E}[\sum Z_j]/3)$.

- a set of variables;
- a linear objective function to be minimized or maximized;
- a set of linear constraints;
- a special constraint saying that some matrix of variables is positive semi-definite (see below).

The special constraint is the only thing that is not seen in linear programs. This constraint has the form "A has to be positive semi-definite, where A is a square matrix whose entries are taken from the variables in the program". In the following, we define positive semi-definite matrices and give some related results in linear algebra.

Definition 19.3.1 (Positive semi-definite matrix) *An $n \times n$ matrix A is* positive semi-definite *(denoted as $A \succeq 0$) if and only if*

1. *A is real symmetric; and*
2. *for all $x \in \mathbb{R}^n$, $x^T Ax = \sum_{i=1}^n \sum_{j=1}^n A_{ij} x_i x_j \geq 0$.*

The following proposition implies that convex combinations of feasible solutions to an SDP are still feasible.

Proposition 19.3.2 *Let A, B be positive semi-definite matrices of the same size. Then*

1. *$A + B \succeq 0$;*
2. *$\lambda A \succeq 0$ for all real $\lambda \geq 0$.*

The following proposition gives different characterizations of positive semi-definite matrices.

Proposition 19.3.3 *Let A be an $n \times n$ matrix. The following statements are equivalent:*

1. *$A \succeq 0$.*
2. *All eigenvalues of A are real and non-negative.*
3. *There exists a matrix $C \in \mathbb{R}^{m \times n}$ ($m \leq n$) such that $A = C^T C$.*

Given a positive semi-definite matrix A, the matrix C in the third characterization can be computed in polynomial time using an algorithm called *Cholesky decomposition*. Note that a positive semi-definite matrix can be viewed as a *matrix of dot products*. If we think of $C = [C_1 \cdots C_n]$ as a bunch of column vectors in $\mathbb{R}^m$, the $(i, j)^{\text{th}}$ entry of A is the dot product $C_i \cdot C_j$. This leads us to viewing SDP as vector programming. A *vector program* is a mathematical program with the following elements:

- a set of variables v_i's in $\mathbb{R}^n$, for some $n > 0$;
- an objective function linear in all $(v_i \cdot v_j)$'s, to be minimized or maximized;
- a set of linear constraints over all $(v_i \cdot v_j)$'s

We will see some vector program formulations in later lectures.

19.3.2 Solving SDPs

Similar to an LP, the feasible region (polytope) of an SDP is convex, except that one of its faces (the one corresponding to the nonlinear constraint) might not be "flat". As a result, the optimal solution to an SDP could be in the interior of a face of the feasible region and could be irrational. Hence, having a polynomial-time algorithm for solving SDPs exactly is impossible. However, we can achieve a $(1+\epsilon)$-approximation in time $poly(n, \log \frac{1}{\epsilon})$, where n is the input size. Typical algorithms for solving SDPs include interior point methods and the ellipsoid method. Recall that the ellipsoid method starts by enclosing the whole feasible region in a large ellipsoid. It then checks if the center of the ellipsoid lies in the feasible region. If not, it picks a violated constraint and computes the intersection of the ellipsoid with the hyperplane corresponding to that constraint. A new, smaller ellipsoid is then used to enclose the intersection. This process is repeated until we have found a point lying in the feasible region, or the ellipsoid has become so small that we can conclude the feasible region to be empty. One nice thing about the ellipsoid method is that even if the SDP has a super-polynomial number of constraints, it still runs in polynomial time, provided that there exists an efficient algorithm for testing feasibility of a given solution, and finding a violated constraint if the solution is infeasible. Such an algorithm is known as a *separation oracle* for the SDP.

19.3.3 Example: Max-Cut

Recall the (Unweighted) Max-Cut problem.

Definition 19.3.4 (Max-Cut) *Given an undirected graph $G = (V, E)$, find a cut in G with maximum cut value, i.e. find a subset $S \subseteq V$ that maximizes*

$$c(S) = \left|\{(v, v') \mid v \in S, v' \in V \backslash S, (v, v') \in E\}\right|.$$

We are going to formulate Max-Cut as an SDP. For each vertex $v \in V$, we have a variable x_v that takes a value in $\{1, -1\}$ depending on the partition to which v is assigned. It is clear that the value of the cut is $\sum_{(u,v)\in E} \frac{|x_u - x_v|}{2}$. We are going to maximize the value. Thus, we have the following nonlinear program.

$$\begin{array}{rll} \text{maximize} & \sum_{(u,v)\in E} \frac{|x_u - x_v|}{2} & \\ \text{subject to} & x_v \in \{-1, 1\} & \forall v \in V \end{array}$$

Note that the objective function in not linear in the x_v's. We can convert the above program to the following equivalent quadratic integer program.

$$\begin{array}{rll} \text{maximize} & \sum_{(u,v)\in E} \frac{(x_u - x_v)^2}{4} & \\ \text{subject to} & x_v^2 = 1 & \forall v \in V \end{array}$$

Solving quadratic programs is NP-hard in general. By introducing new variables, the objective function can be converted to a linear one. Specifically, we have a new variable y_{uv} for each pair $(u, v) \in V \times V$. We add (nonlinear) constraints to enforce the equality $y_{uv} = x_u x_v$ for every u and v. By observing that $(x_u - x_v)^2 = x_u^2 + x_v^2 - 2x_u x_v = 2 - 2y_{uv}$, we obtain the following program.

$$\begin{array}{rll} \text{maximize} & \sum_{(u,v)\in E} \frac{1 - y_{uv}}{2} & \\ \text{subject to} & y_{uv} = x_u x_v & \forall (u, v) \in V \times V \\ & y_{vv} = 1 & \forall v \in V \end{array}$$

Let $Y = [y_{uv}]_{u,v \in V}$ be a matrix of variables. Then $Y = X^T X$ where $X = [x_v]_{v \in V}$ is a row vector containing the variables x_v's. We rewrite the above program in matrix form.

$$\begin{array}{rll} \text{maximize} & \sum_{(u,v)\in E} \frac{1-y_{uv}}{2} & \\ \text{subject to} & y_{vv} = 1 & \forall v \in V \\ & Y = X^T X & \\ & X \text{ is a row vector} & \end{array}$$

The above program can be relaxed into an SDP. By Proposition 19.3.3, $Y \succeq 0$. The resulting SDP relaxation is as follows. Note that x_v's no longer appear in the program.

$$\begin{array}{rll} \text{maximize} & \sum_{(u,v)\in E} \frac{1-y_{uv}}{2} & \\ \text{subject to} & y_{vv} = 1 & \forall v \in V \\ & Y \succeq 0 & \end{array}$$

Since the SDP is a relaxation of the original program, its optimal value must be at least the optimal cut value. To get an approximation, we still need to show how to convert the SDP solution to a cut whose value is not much smaller than the optimal value of the SDP. Let Y^* be an optimal solution to the above SDP. Using Cholesky decomposition, we can find in polynomial time a matrix X^* such that $Y^* = X^{*T} X^*$. Suppose that X^* has m rows, where $m \leq n$. Then each vertex v is represented by a vector in $\mathbb{R}^m$. Note that these vectors are unit vectors because $y_{vv} = 1$ for all v, and hence they all lie on the unit sphere in $\mathbb{R}^m$. In order to obtain a cut, we need separate these vectors into two sets. Let (u, v) be an edge in G. Let X^*_u and X^*_v be a unit vectors corresponding to u and v. If X^*_u and X^*_v are far apart, their dot product $X^*_u \cdot X^*_v = y_{uv}$ is small, and so the value $\frac{1-y_{uv}}{2}$ is large. Thus, to maximize $\sum_{(u,v)\in E} \frac{1-y_{uv}}{2}$, our objective is to separate the vectors into two sets such that "long" edges get cut. If we pick a random hyperplane through the origin, the probability that an edge gets cut is roughly proportional to its length. This randomized approach seems to be a nice and easy way to meet our objective. We will continue our discussion next time. The algorithm is due to Goemans and Williamson [3].

References

[1] J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel J. Math.*, 52(1–2), pp. 46–52, 1985.

[2] N. Linial, E. London and Y .Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15, pp. 215–245, 1995.

[3] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6), pp. 1115–1145, 1995.