Approximation Algorithms Workshop June 13-17, 2011, Princeton

Lecture Combinatorial, Primal-Dual Approach to SDP. June 16, 2011

Satyen Kale *Scribe: Satyen Kale*

1 Overview

In this lecture we study a technique to obtain fast approximation algorithms for various combinatorial optimization problems via SDP, significantly improving over interior-point methods. This technique is called Matrix Multiplicative Weights (Matrix MW for short).

2 Preliminaries

2.1 Notation

Unless specified otherwise, all vectors in this lecture live in $\mathbb{R}^n$, and all matrices are symmetric and live in $\mathbb{R}^{n\times n}$. For two vectors $\mathbf{v}, \mathbf{w}$, let $\mathbf{v}\cdot\mathbf{w} = \sum_i v_i w_i$ denote their inner product, and $\mathbf{v} \succeq \mathbf{0}$ indicate that all $v_i \geq 0$. For two matrices $\mathbf{A}$ and $\mathbf{B}$, denote by $\mathbf{A}\bullet\mathbf{B}$ their inner product thinking of them as vectors in $\mathbb{R}^{n^2}$, i.e. $\mathbf{A}\bullet\mathbf{B} = \sum_{ij} A_{ij}B_{ij} = \mathrm{Tr}(\mathbf{A}^\top\mathbf{B})$. Here $\mathrm{Tr}(\cdot)$ denotes the trace of a matrix. A matrix $\mathbf{A}$ is positive semidefinite, denoted by $\mathbf{A} \succeq \mathbf{0}$, if all its eigenvalues are non-negative. Equivalently, there are n vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ such that $A_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$. We denote by $\mathbf{A} \succeq \mathbf{B}$ the fact that $\mathbf{A} - \mathbf{B}$ is positive semidefinite.

2.2 Formulation of SDP

We begin by describing the specific form of SDP we will consider. This is a fairly general form, and almost all SDP relaxations one encounters can be cast in this form. The goal is to find a matrix $\mathbf{X}$ which satisfies:

$$\begin{aligned}
\mathbf{A}_1 \bullet \mathbf{X} &\leq 0\\
\mathbf{A}_2 \bullet \mathbf{X} &\leq 0\\
&\vdots\\
\mathbf{A}_m \bullet \mathbf{X} &\leq 0\\
\mathbf{X} &\succeq \mathbf{0}\\
\mathrm{Tr}(\mathbf{X}) &= 1
\end{aligned}$$

Positive semidefinite matrices $\mathbf{X}$ of trace 1 are called *density matrices* in quantum computing.

2.3 Ubiquity of SDP

SDP is a widely used technique in approximation algorithms today. Starting with the seminal work of Goemans and Williamson [GW95] on the MaxCut problem, it has been used in various other approximation algorithms, such as those for Coloring [ACC06], Graph Partitioning [ARV04], Constraint Satisfaction [ACMM05], etc. For all these problems, the best known approximation factor has been obtained via SDP. Other fields, such as Control Theory, have also made wide use of SDP.

2.4 Algorithms for SDP

SDP is a convex optimization problem, and hence can be solved using standard convex optimization techniques. The earliest polynomial time (approximation) algorithm for SDP is due to Grötschel, Lovász, and Schrijver [GLS88] using the ellipsoid algorithm, observing that a separation oracle for the set of positive semidefinite matrices can be performed via an eigenvector computation. While the ellipsoid method gives a polynomial time algorithm, its running time is quite high, making it infeasible in practice. Much better algorithms for SDP were obtained using Interior Point methods [NN92, Ali95]. These algorithms are still the method of choice for solving SDPs in practice.

While the Interior Point algorithms are quite practical, theoretically however their running time is still quite high even for SDP relaxations of basic combinatorial optimization problems like MaxCut. This issue is exacerbated for problems like Sparsest Cut which make use of $O(n^3)$ triangle inequality constraints.

For this reason, it is of interest to develop specialized methods to solve particular SDPs faster than Interior Point methods, at least when some approximation is allowed. This motivated Klein and Lu [KL96] to develop faster algorithms for the MaxCut and Coloring SDPs using the technique of Lagrangian relaxation. In later work, Arora, Hazan and Kale [AHK05] generalized this technique for arbitrary SDPs, and obtained faster running times for certain specific SDPs.

All the above mentioned techniques were first developed in the context of linear programs (LP). Another technique which is very successful in the design of fast algorithms based on LP is the combinatorial, primal-dual method [GW97]. Such algorithms hadn't been developed for SDP relaxations however for a number of reasons. One is that such algorithms typically make local changes in the variables, and maintaining positive semidefiniteness becomes difficult. Another reason is that rounding algorithms based on SDP relaxations are inherently geometric making it difficult to use them in combinatorial, primal-dual algorithms. Finally, even if we were to surmount these difficulties, a significant issue that still remains is that matrix operations are inherently less efficient to implement than vector operations, and so running time improvements are difficult.

In this lecture, we describe a technique which tackles these issues and leads to much faster approximation algorithms for various problems such as MaxCut, Sparsest Cut, Balanced Separator, Min UnCut, and Min 2CNF Deletion [AK07]. In fact, for MaxCut, this technique gives the first near-linear time implementation of the Goemans-Williamson approximation algorithm.

3 Multiplicative Weights for LP

Before giving the Matrix MW algorithm, we first discuss the basic Multiplicative Weights (MW) algorithm for LP. This is essentially the Winnow algorithm of Littlestone [Lit87]. Consider the following LP. The goal is to find a vector $\mathbf{x}$ which satisfies:

$$\begin{aligned} \mathbf{a}_1 \cdot \mathbf{x} &\leq 0 \\ \mathbf{a}_2 \cdot \mathbf{x} &\leq 0 \\ &\vdots \\ \mathbf{a}_m \cdot \mathbf{x} &\leq 0 \\ \mathbf{x} &\succeq \mathbf{0} \\ \sum_i x_i &= 1 \end{aligned}$$

Note the similar form of this LP to the form of the SDP we are concerned with. This similarity goes deeper: if all the matrices in the SDP are diagonal, then it reduces exactly to the LP given above. Note that $\mathbf{x}$ forms a distribution, analogous to the density matrix in the SDP.

Suppose we want to solve this LP up to an additive error of δ, i.e. we want to find a distribution $\mathbf{x}$ such that for all $j = 1, \ldots, m$, we have $\mathbf{a}_j \cdot \mathbf{x} \leq \delta$. Call such an $\mathbf{x}$ "δ-feasible".

Assume that we have access to an algorithm, ViolationChecker, which given a distribution $\mathbf{x}$, efficiently determines if there is any constraint j such that $\mathbf{a}_j \cdot \mathbf{x} > \delta$. If there isn't, then we have found our δ-feasible solution to the LP.

We now give the MW algorithm. It is parameterized by $\varepsilon \in (0, 1)$ and an integer $T > 0$.

MW

1. Initialize a weight vector $\mathbf{w}^{(1)}$ to the all 1's vector.
2. For $t = 1, 2, \ldots, T$:
   (a) Let $\mathbf{x}^{(t)} = \mathbf{w}^{(t)}/\Phi^{(t)}$, where $\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)}$ is the normalization constant to make $\mathbf{x}^{(t+1)}$ a distribution.
   (b) Send $\mathbf{x}^{(t)}$ to ViolationChecker.
   (c) If ViolationChecker reports that $\mathbf{x}^{(t)}$ is δ-feasible, stop and output $\mathbf{x}^{(t)}$.
   (d) Else, let $\mathbf{a}_{j^{(t)}} \cdot \mathbf{x}^{(t)} > \delta$ be a constraint violated by $\mathbf{x}^{(t)}$ found by ViolationChecker. Update:
   $$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} \times \exp(-\varepsilon \mathbf{a}_{j^{(t)}}).$$
   The exponential and product above is taken coordinate-wise.
3. Output that the LP is infeasible.

3.1 Analysis

For the analysis, we need to define a parameter, ρ, called the *width* of the LP:

Definition 1. *Define* $\rho = \max_{ij} |a_{ij}|$.

Theorem 2. *If the MW algorithm is run with* $\varepsilon = \frac{\delta}{2\rho^2}$ *and* $T = \left\lceil \frac{2\rho^2 \log n}{\delta^2} \right\rceil$, *then either it outputs a* δ*-feasible* $\mathbf{x}$*, or outputs correctly that the LP is infeasible.*

Proof. Suppose for all iterations t, the point $\mathbf{x}^{(t)}$ is not δ-feasible. Then we want to prove that the LP is indeed infeasible.

Suppose for the sake of contradiction that the LP is feasible, i.e. there is a distribution $\mathbf{x}^\star$ such that for all j, we have $\mathbf{a}_j \cdot \mathbf{x}^\star \geq 0$. We now use the normalization factor Φ_t as a potential function. We have

$$\begin{aligned}
\Phi^{(t+1)} &= \sum_i w_i^{(t+1)} \\
&= \sum_i w_i^{(t)} \exp(-\varepsilon a_{ij^{(t)}}) \\
&\leq \sum_i w_i^{(t)} (1 - \varepsilon a_{ij^{(t)}} + \varepsilon^2 a_{ij^{(t)}}) && \because \exp(x) \leq 1 + x + x^2 \text{ for } |x| \leq 1 \\
&\leq \Phi^{(t)} (1 - \varepsilon \mathbf{a}_{j^{(t)}} \cdot \mathbf{x}^{(t)} + \varepsilon^2 \rho^2) && \because |a_{ij^{(t)}}| \leq \rho \\
&\leq \Phi^{(t)} \exp(-\varepsilon \mathbf{a}_{j^{(t)}} \cdot \mathbf{x}^{(t)} + \varepsilon^2 \rho^2) && \because \exp(x) \geq 1 + x \\
&< \Phi^{(t)} \exp(-\varepsilon\delta/2). && \because \mathbf{a}_{j^{(t)}} \cdot \mathbf{x}^{(t)} > \delta \text{ and } \varepsilon^2\rho^2 = \varepsilon\delta/2
\end{aligned}$$

Thus, by induction, after T rounds, we have

$$\Phi^{(T+1)} < \Phi^{(1)} \exp(-\varepsilon\delta T/2) = n \exp(-\varepsilon\delta T/2) \leq 1, \tag{1}$$

for $T = \left\lceil \frac{\log n}{\varepsilon\delta} \right\rceil = \left\lceil \frac{2\rho^2 \log n}{\delta^2} \right\rceil$. Since for any i, we have

$$\Phi^{(T+1)} \geq w_i^{(T+1)} = \exp\left(-\varepsilon \sum_{t=1}^{T} a_{ij^{(t)}} \right),$$

so from (1) we get that $\sum_{t=1}^{T} a_{ij^{(t)}} > 0$ for all i. But we assumed that $\mathbf{a}_j \cdot \mathbf{x}^\star \leq 0$ for all j, implying that

$$0 \geq \sum_{t=1}^{T} \mathbf{a}_{j^{(t)}} \cdot \mathbf{x}^\star = \sum_{i=1}^{n} x_i^\star \cdot \sum_{t=1}^{T} a_{ij^{(t)}} > 0,$$

a contradiction. Hence, our assumption is wrong, and the LP is indeed infeasible. □

3.2 Making it primal-dual.

It can be easily seen that the ViolationChecker doesn't necessarily need to identify a single violated constraint. Rather, the algorithm works just as well if ViolationChecker outputs a *distribution* $\mathbf{y}$ over the constraints such that the expected constraint is violated, i.e. $\sum_j y_j \mathbf{a}_j \cdot \mathbf{x}^{(t)} > \delta$. Now, the y_j variables can be seen to be dual variables for the LP. Thus, the choice of the primal variables, $\mathbf{x}^{(t)}$, induces a choice for the dual variables – namely, dual variables which certify that $\mathbf{x}^{(t)}$ is not δ-feasible – and these dual variables in turn are used to update the primal variables. In

this sense, the algorithm can be thought of as primal-dual. In several applications, it is possible to obtain running time speedups by exploiting the combinatorial structure of the problem and setting the dual variables cleverly (say, in a way that guarantees that the expected constraint is sparse, leading to fast updates).

4 Matrix Multiplicative Weights for SDP

We now turn to the SDP. Again, we are looking for a δ-feasible solution, i.e. a density matrix $\mathbf{X}$ such that for all j, $\mathbf{A}_j \bullet \mathbf{X} \leq \delta$. As before, we assume we have access to an algorithm, ViolationChecker, that given a density matrix $\mathbf{X}$, efficiently determines if there is any constraint j such that $\mathbf{A}_j \bullet \mathbf{X} > \delta$.

We now give the Matrix MW algorithm. It is parameterized by $\varepsilon \in (0, 1)$ and an integer $T > 0$.

Matrix MW

1. Initialize a weight matrix $\mathbf{W}^{(1)}$ to $\mathbf{I}$.
2. For $t = 1, 2, \ldots, T$:
   (a) Let $\mathbf{X}^{(t)} = \mathbf{W}^{(t)}/\Phi^{(t)}$, where Φ_t is the normalization constant to make $\mathbf{X}^{(t+1)}$ a density matrix.
   (b) Send $\mathbf{X}^{(t)}$ to ViolationChecker.
   (c) If ViolationChecker reports that $\mathbf{X}^{(t)}$ is δ-feasible, stop and output $\mathbf{X}^{(t)}$.
   (d) Else, let $\mathbf{A}_j \bullet \mathbf{X}^{(t)} > \delta$ be a constraint violated by $\mathbf{X}^{(t)}$ found by ViolationChecker. Update:
   $$\mathbf{W}^{(t+1)} = \exp(-\varepsilon \textstyle\sum_{\tau=1}^{t} \mathbf{A}_j).$$
3. Output that the SDP is infeasible.

Note that if all the matrices involved in the SDP were diagonal, then this algorithm exactly reduces to the MW algorithm for LP given earlier. This algorithm can also be made primal-dual by having the ViolationChecker find a distribution $\mathbf{y}$ over the constraints such the expected constraint is violated by at least δ, i.e. $\sum_j \mathbf{A}_j \bullet \mathbf{X}^{(t)} > \delta$.

4.1 Analysis

For the analysis, we need to define a parameter, ρ, called the *width* of the SDP:

Definition 3. *Define* $\rho = \max_j \|\mathbf{A}_j\|$.

Theorem 4. *If the MW algorithm is run with* $\varepsilon = \frac{\delta}{2\rho^2}$ *and* $T = \left\lceil \frac{2\rho^2 \log n}{\delta^2} \right\rceil$, *then either it outputs a* δ*-feasible* $\mathbf{X}$, *or outputs correctly that the SDP is infeasible.*

Proof. The proof is on the same lines as that of Theorem 2, using $\mathrm{Tr}(\Phi^{(t)})$ as a potential function. The analysis needs to use the following two inequalities:

1. Golden-Thompson inequality [Gol65, Tho65]: for any two symmetric matrices $\mathbf{A}$ and $\mathbf{B}$ of like order, we have
$$\mathrm{Tr}(\exp(\mathbf{A}+\mathbf{B})) \;\leq\; \mathrm{Tr}(\exp(\mathbf{A}))\mathrm{Tr}(\exp(\mathbf{B})).$$
2. For any matrix $\mathbf{X}$ with $\|\mathbf{X}\| \leq 1$, we have
$$\exp(\mathbf{X}) \;\preceq\; \mathbf{I}+\mathbf{X}+\mathbf{X}^2.$$

□

5 Applying Matrix MW in Approximation Algorithms

Approximation Algorithms via SDP relaxations. The usual recipe for designing approximation algorithms based on SDP is the following.

1. Start with an NP-hard combinatorial optimization problem (say MaxCut). Formulate it as an equivalent quadratic program.
2. Replace all products between scalar variables in the quadratic program by inner products between vector variables (i.e., replace $x_i x_j$ by $\mathbf{v}_i \cdot \mathbf{v}_j$). The result is an SDP.
3. Solve the SDP to (near-)optimality using a standard SDP solver such as an Interior Point algorithm. This gives a positive semidefinite matrix $\mathbf{X}$ as the optimal solution.
4. Decompose $\mathbf{X}$ via the Cholesky factorization into vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ such that $X_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$.
5. Run some geometric rounding algorithm (such as hyperplane rounding in the case of MaxCut) to obtain an approximately-optimum integral solution to the original problem.

Using Matrix MW in Approximation Algorithms. To use the Matrix MW algorithm to speed up such an approximation algorithm, we replace steps 3, 4, and 5 of the above with the Matrix MW algorithm. Here is the general recipe:

1. Start with an NP-hard combinatorial optimization problem. Formulate it as an equivalent quadratic program.
2. Replace all products between scalar variables in the quadratic program by inner products between vector variables (i.e., replace $x_i x_j$ by $\mathbf{v}_i \cdot \mathbf{v}_j$). The result is an SDP.
3. Run the Matrix MW algorithm on the SDP. In each round:
   (a) Obtain the candidate solution $\mathbf{X}^{(t)}$. Note that it is positive semidefinite. Decompose $\mathbf{X}^{(t)}$ via the Cholesky factorization into vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ such that $X^{(t)}_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$.
   (b) Run the geometric rounding algorithm on the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$.
   (c) The rounding algorithm may or not run on the vectors, because $\mathbf{X}^{(t)}$ may not satisfy all the constraints of the SDP. If it succeeds, usually this produces the approximately-optimal integral solution to the original problem.

(d) Else, identify which constraints of the SDP were violated by $\mathbf{X}^{(t)}$ that caused the rounding algorithm to fail. Use these constraints to generate feedback in terms of a distribution $\mathbf{y}$ on the constraints for the Matrix MW algorithm.

The performance benefit to this approach comes from two sources: (a) the number of iterations in the Matrix MW algorithm depends only logarithmically in the dimension n, and (b) the use of the rounding algorithm directly on infeasible solutions gives a way to identify violated constraints quickly.

Most of the work in applying this method is in understand how the rounding algorithm works and tweaking it so that it can be used as the ViolationChecker.

6 Illustration: Approximating Balanced Separator

In this section, we give a high-level overview of applying this method to approximate the Balanced Separator problem. In this problem, we are given a graph $G = (V, E)$ (for simplicity, assume that the edges are unweighted) with $n = |V|$. For a given fraction $c < 1/2$, a c-balanced cut in the graph $S \subseteq V, \bar{S} = V \setminus S$, is one that has at least cn nodes in both S and $\bar{S}$. The objective of the Balanced Separator problem is to find the c-balanced cut of minimum capacity.

We start by writing an SDP for this problem. We follow the general recipe given above, and give the quadratic programming formulation for the problem. For every node i, we have a variable $x_i \in \{-1, 1\}$. The interpretation of the solution is that if $x_i = 1$, then $i \in S$, otherwise, $i \in \bar{S}$. For a pair of nodes $i \neq j$, consider the quantity $\frac{1}{4}(x_i - x_j)^2$. This is equal to 0 if i and j lie on the same side of the cut $S, \bar{S}$, and 1 otherwise. The quadratic program can then be written as follows:

$$\begin{aligned} \min \sum_{(i,j)\in E} \frac{1}{4}(x_i - x_j)^2 & \\ x_i^2 &= 1 \\ \sum_{i,j} \frac{1}{4}(x_i - x_j)^2 &\geq c(1-c)n^2 \\ \forall i, j, k: \ (x_i - x_j)^2 + (x_j - x_k)^2 &\geq (x_i - x_k)^2 \end{aligned}$$

The objective minimizes the number of edges cut, the first constraint ensures that $x_i \in \{-1, 1\}$, the second constraint above ensures that the solution is c-balanced, and the third constraints are triangle inequality constraints that are satisfied for any $-1, 1$ solution.

To write this as an SDP, we replace all products between scalars by inner products between vectors. We get the following SDP:

$$\begin{aligned} \min \sum_{(i,j)\in E} \frac{1}{4}\|\mathbf{v}_i - \mathbf{v}_j\|^2 & \\ \|\mathbf{v}_i\|^2 &= 1 \\ \sum_{i,j} \frac{1}{4}\|\mathbf{v}_i - \mathbf{v}_j\|^2 &\geq c(1-c)n^2 \\ \forall i, j, k: \ \|\mathbf{v}_i - \mathbf{v}_j\|^2 + \|\mathbf{v}_j - \mathbf{v}_k\|^2 &\geq \|\mathbf{v}_i - \mathbf{v}_k\|^2 \end{aligned}$$

Now, to apply the Matrix MW algorithm, we convert the problem into a feasibility problem by replacing the objective by the constraint $\frac{1}{4}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \alpha$, where α is a binary search parameter. We get the following feasibility problem:

$$\begin{aligned}
\sum_{(i,j)\in E} \frac{1}{4}\|\mathbf{v}_i - \mathbf{v}_j\|^2 &\leq \alpha \\
\|\mathbf{v}_i\|^2 &= 1 \\
\sum_{i,j} \frac{1}{4}\|\mathbf{v}_i - \mathbf{v}_j\|^2 &\geq c(1-c)n^2 \\
\forall i,j,k: \ \|\mathbf{v}_i - \mathbf{v}_j\|^2 + \|\mathbf{v}_j - \mathbf{v}_k\|^2 &\geq \|\mathbf{v}_i - \mathbf{v}_k\|^2
\end{aligned}$$

In each iteration of the Matrix MW algorithm, we are given a candidate solution $\mathbf{X}$ which is positive semidefinite, and assume that it is scaled so that its trace equals n. Now we need to implement the ViolationChecker for this problem. It is easy to check the second and third constraints above, so we omit that part, and assume that those constraints are satisfied. The difficult part is to check the triangle inequality constraints, since they are so numerous: they are $O(n^3)$ in number. We reduce the problem of checking the first constraint and the triangle inequality constraints to the following multicommodity flow problem:

Flow Problem: Find a multicommodity flow (i.e. an assignment of positive numbers f_p to every path p in the graph) such that

1. For any node, the total flow on all paths starting from the node is at most $d = \tilde{O}(\alpha/n)$ (degree constraints),
2. For any edge, the total flow on all paths using the edge is at most 1 (capacity constraints),
3. If f_{ij} is the total flow on all paths from i to j, then

$$\sum_{ij} \frac{1}{4} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 > \alpha.$$

The degree constraints are useful in bounding the width of the problem and we will not discuss them further. If we find such a multicommodity flow (assigning flow f_p to path p), then we conclude that either the first constraint or the triangle inequality constraints are violated. To show this, assume for the sake of contradiction that the first and triangle inequality constraints are satisfied. Then for any path p connecting nodes i, j, the triangle inequalities imply the path constraint

$$\sum_{\text{Edge } (k,\ell)\in p} \|\mathbf{v}_k - \mathbf{v}_\ell\|^2 \ \geq \ \|\mathbf{v}_i - \mathbf{v}_j\|^2.$$

Taking a weighted combination of such path inequalities with weight f_p for path p, we get

$$\sum_p \sum_{\text{Edge } (k,\ell)\in p} f_p\|\mathbf{v}_k - \mathbf{v}_\ell\|^2 \ \geq \ \sum_p f_p\|\mathbf{v}_i - \mathbf{v}_j\|^2 \ = \ \sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \ > \ 4\alpha.$$

On the other hand, using the capacity constraints we get

$$\sum_{p} \sum_{\text{Edge } (k,\ell)\in p} f_p \|\mathbf{v}_k - \mathbf{v}_\ell\|^2 \;=\; \sum_{(i,j)\in E} \sum_{p \ni (i,j)} f_p \|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\le\; \sum_{(i,j)\in E} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\le\; 4\alpha,$$

a contradiction.

Thus, finding a multicommodity flow of the specified type is essentially equivalent to showing a combination of constraints that is violated. For brevity we omit details of how this feedback is used in the Matrix MW algorithm, and instead specify how to obtain the required multicommodity flow to obtain $O(\log n)$ and $O(\sqrt{\log n})$ approximation factors.

6.1 Obtaining an $O(\log n)$ approximation

Recall that we are assuming that the second and third inequalities are satisfied. We run the following rounding algorithm on the vectors $\mathbf{v}_i$: take a random unit vector $\mathbf{u}$. Let $L = \{i : \ \mathbf{v}_i \cdot \mathbf{u} < -\sigma\}$, and let $R = \{i : \ \mathbf{v}_i \cdot \mathbf{u} > \sigma\}$, for $\sigma = \Theta(\frac{1}{\log n})$. The fact that the third constraint is satisfied implies that both L and R are of size $\Omega(n)$ with constant probability. Now connect all nodes in L to a single source with edges of capacity d, and all nodes in R to a single sink with edges of capacity d. Run a single-commodity max-flow computation from the source to the sink.

We can prove the following theorem (see [AK07] for details):

Theorem 5. *If the max-flow has value less than $O(\log(n)\cdot\alpha)$, then the min-cut found in the process is an $O(\log(n))$ approximation to the minimum c-balanced separator. Otherwise, the max-flow gives the required multicommodity flow (by ignoring the flow on the source and sink edges).*

6.2 Obtaining an $O(\sqrt{\log n})$ approximation

To obtain an $O(\sqrt{\log n})$ approximation, we consider a multicommodity flow problem in the graph, where the source-sink pairs are obtained by choosing all pairs i, j such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge s$, for some small constant s. To ensure the degree constraints hold, we connect each such pair to fictitious source and sink nodes with edges of capacity d. Then we compute the maximum multicommodity flow in the graph using the algorithm of Garg and Könemann.

We can prove the following theorem (see [AK07] for details):

Theorem 6. *If the maximum multicommodity flow has value less than $O(\alpha)$, then dual variables for the flow problem found in the process can be used to find an $O(\sqrt{\log(n)})$ approximation to the minimum c-balanced separator. Otherwise, the multicommodity flow is the required flow (by ignoring the flow on the fictitious source and sink edges).*

References

[ACC06] Sanjeev Arora, Eden Chlamtac, and Moses Charikar. New approximation guarantee for chromatic number. In *STOC*, pages 215–224, 2006.

[ACMM05] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF deletion, and directed cut problems. In *STOC*, pages 573–581, 2005.

[AHK05] S. Arora, E. Hazan, and S. Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *FOCS*, pages 339–348, 2005.

[AK07] S. Arora and S. Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, 2007.

[Ali95] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, 5(1):13–51, 1995.

[ARV04] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings, and graph partitioning. In *STOC*, pages 222–231, 2004.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, 1988.

[Gol65] S. Golden. Lower Bounds for the Helmholtz Function. *Physical Review*, 137:1127–1128, February 1965.

[GW95] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

[GW97] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. *Approximation Algorithms*, pages 144–191, 1997.

[KL96] P. Klein and H.-I. Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *STOC*, pages 338–347, 1996.

[Lit87] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987.

[NN92] Yu. Nesterov and A. S. Nemirovskii. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992.

[Tho65] C. J. Thompson. Inequality with applications in statistical mechanics. *Journal of Mathematical Physics*, 6(11):1812–1823, 1965.