A Geometric Approach to Betweenness [*]

Benny Chor[**] and Madhu Sudan[***]

Abstract. An input to the *betweenness* problem contains m constraints over n real variables. Each constraint consists of three variables, where one of the variables is specified to lie inside the interval defined by the other two. The order of the other two variables (which one is the largest and which one is the smallest) is not specified. This problem comes up in questions related to physical mapping in computational molecular biology. In 1979, Opatrny has shown that the problem of deciding whether the n variables can be totally ordered while satisfying the m betweenness constraints is NP–complete. Furthermore, the problem is MAX SNP complete. Therefore, there is some $\epsilon > 0$ such that finding a total order which satisfies at least $m(1-\epsilon)$ of the constraints (even if they are all satisfiable) is NP–hard. It is easy to find an ordering of the variables which satisfies $1/3$ of the m constraints (e.g. by choosing the ordering at random).
In this work we present a polynomial time algorithm which either determines that there is no feasible solution, or finds a total order which satisfies at least $1/2$ of the m constraints. Our algorithm translates the problem into a set of quadratic inequalities, and solves a semidefinite relaxation of them in $\mathcal{R}^n$. The n solution points are then projected on a random line through the origin. Using simple geometric properties of the SDP solution, we prove the claimed performance guarantee.

1 Introduction

An input to the *betweenness* problem consists of a finite set of n real variables $S = \{x_1, \ldots, x_n\}$, and a finite set of m constraints. Each constraint consists of a triplet $(x_i, x_j, x_k) \in S \times S \times S$. A total order $x_{i_1} < x_{i_2} < \ldots < x_{i_n}$ satisfies the constraint (x_i, x_j, x_k) if either $x_i < x_j < x_k$ or $x_k < x_j < x_i$. That is, each constraint forces the second variable x_j to be between the two other variables x_i and x_k, but does not specify the relative order of x_i and x_k. The betweenness problem is to decide if all constraint can be simultaneously satisfied by a total order of the variables.

In 1979, Opatrny [13] has shown that the betweenness problem is NP–complete. This problem naturally arises in the analysis of certain mapping problems in molecular

[*] A preliminary version of this paper appeared in the proceedings of the third annual European symposium on algorithms, ESA '95, lecture notes in Computer Science 979, Paul Spirakis (Ed.), Springer, September 1995, pp. 227–237.

[**] Dept. of Computer Science, Technion, Haifa 32000, Israel (benny@cs.technion.ac.il). Research partially supported by Technion V.P.R. funds.

[***] IBM Thomas J. watson Research Center, P.O.Box 218, Yorktown Heights, NY 10598 (madhu@watson.ibm.com)

biology. For example, when trying to order markers on a chromosome, given the results of a radiation hybrid experiment [6, 3]. A computational task of practical significance, in this context, is to find a total ordering of the markers (the x_i in our terminology) which maximizes the number of satisfied constraints.

Opatrny gave two reductions in his proof of NP–completeness. One of these reductions is from 3SAT. Following his construction, we show, in Section 2, an approximation preserving reduction from MAX 3SAT. This implies that there exists an $\epsilon > 0$, such that finding a total order which satisfies at least $m(1-\epsilon)$ of the constraints (even if they are all satisfiable) is NP–hard. It is easy to find a total order which satisfies one third of the m constraints. Simply arranges the points in a random order along the line. The probability that a specific constraint is satisfied by such a randomly chosen order is $1/3$. Thus the expected number of constraints satisfied by a random order is at least a third of the m constraints. On the other hand, it is easy to construct examples where at most $m/3$ constraints are satisfiable. Thus to achieve better approximation factors, one needs to be able to recognize instances of the betweenness problems which are not satisfiable.

We present a polynomial time algorithm which either determines that there is no feasible solution, or finds a total order which satisfies at least $1/2$ of the m constraints. Our algorithm translates the problem into a set of quadratic inequalities, and solves a semidefinite relaxation of them in $\mathcal{R}^n$. Let $v_1, \ldots, v_n \in \mathcal{R}^n$ be a feasible solution to the SDP, where each v_i corresponds to the real variable x_i. The n solution points are then projected on a random line through the origin. We show that if "x_j between x_i and x_k" is one of the betweenness constraint, then the angle between the lines $v_i v_j$ and $v_k v_j$ (in $\mathcal{R}^n$) is obtuse. Using this property, we prove that the random projection satisfies each constraint with probability at least $1/2$. This gives a randomized algorithm with the claimed performance guarantee. Next, we show how to derandomize the algorithm. In addition, we demonstrate that our analysis of the semidefinite program is tight. There is an infinite family of inputs to the betweenness problem, such that the resulting SDP is feasible, but any total order of the variables satisfies at most $1/2 + o(1)$ of the m constraints.

Our use of semidefinite programming is inspired by the recent success in using this methodology to find improved approximation algorithms for several optimization problems. The applicability of SDP in combinatorial optimization was demonstrated by Grötschel, Lovász and Schrijver [7] to show that the Theta function of Lovász [12] was polynomial time computable. This application was then turned into exact coloring and independent set finding algorithms for perfect graphs. The use of SDP in approximation algorithms was innovated by the work of Goemans and Williamson [5] who break longstanding barriers in the approximability of MAX CUT and MAX 2SAT by their SDP based algorithm. Further evidence of the applicability of the SDP approach is provided by the works of Karger, Motwani and Sudan [10], who use it to approximate graph coloring, Alon and Kahale [1] (independent set approximation) and by Feige and Goemans [4] (improvements to MAX 2SAT).

Thus the semidefinite programming method has now been used successfully to solve many optimization problems — exactly and approximately. However all the cases where SDP has been used to find approximation algorithms seem to be essentially partition problems (MAX CUT, Coloring, Multicut etc.). Our solution seems to be (to the best of our knowledge) the only case where SDP has been used to solve an ordering problem. This syntactic difference between ordered structures and unordered ones, and the ability of SDP to help optimize over both, offers critical additional evidence on the power of the SDP methodology.

The remaining of this paper is organized as following: Section 2 presents the approximation preserving reduction from MAX 3SAT, as well as other observations about the betweenness problem. Semidefinite programming is briefly reviewed in Section 3. The algorithm is presented in Section 4. Section 5 shows the tightness of our analysis. Finally, Section 6 contains some concluding remarks and open problems.

2 Preliminaries

We start this section with some preliminary observations about the betweenness problem. We begin by analyzing the complexity of finding approximate solutions to the problem. Opatrny [13] has shown that it is hard to decide if a given instance of the betweenness problem is satisfiable. Following Opatrny's proof, we present an approximation preserving reduction from MAX 3SAT to the betweenness problem. It is known that there exists a constant $\epsilon > 0$ such that finding a solution which satisfies of $1 - \epsilon$ fraction of all clauses in a satisfiable problem is NP-hard [2]. Therefore, we conclude that there exists a constant $\epsilon' > 0$ such that finding an ordering which satisfies $(1-\epsilon')$ fraction of the constraints in a satisfiable instance of the betweenness problem is NP-hard.

Proposition 1. *If, for some $\epsilon > 0$, there exists a polytime algorithm that satisfies $1 - \epsilon$ fraction of all constraints in a satisfiable instance of the betweenness problem, then there exists a polytime algorithm that satisfies $1 - 6\epsilon$ fraction of all clauses in every satisfiable 3-cnf formula.*

Proof. Given a 3-CNF formula ϕ, we construct an instance I of the betweenness problem as follows. For each Boolean variable x_i of ϕ, we create a point p_i in I. In addition we create two special points T and F. Given a clause say $C_j = x_1 \vee x_2 \vee \overline{x_3}$, we create three points $q_j^{(12)}$, $q_j^{(123)}$, and $q_j^{(\overline{3})}$. Corresponding to the clause C_j we introduce the following betweenness constraints: F is between p_3 and $q_j^{(\overline{3})}$; $q_j^{(12)}$ is between p_1 and p_2; $q_j^{(123)}$ is between $q_j^{(12)}$ and $q_j^{(\overline{3})}$ and $q_j^{(123)}$ is between T and F.

Thus corresponding to each clause we may have upto 6 betweenness constraints (3 + number of negated literals in the clause). Given a satisfiable formula ϕ with say the assignment $x_1, \ldots, x_k = \mathsf{false}$ and $x_{k+1}, \ldots, x_n = \mathsf{true}$ being a satisfying assignment — we first lay out the points p_i and T and F as follows:

$$p_1 \cdots p_k \; F \; p_{k+1} \cdots p_n \; T.$$

Then for a clause, say $C_j = x_1 \vee x_2 \vee \overline{x_3}$ we insert the points $q_j^{(\overline{3})}$, $q_j^{(12)}$ and $q_j^{(123)}$ in order into the linear order above placing each point as far to the right as possible without violating any betweenness constraint. Notice that if the clause is satisfiable then such an ordering would automatically place $q_j^{(123)}$ between F and T, thereby satisfying all betweenness constraints associated with the clause C_j.

To show that the reduction preserves the quality of the approximation, we show that given an ordering which leaves at most k betweenness constraints that are not satisfied, there exists an assignment which leaves at most k clauses of ϕ unsatisfied. W.l.o.g. assume that T lies to the right of F in the linear order. For every point p_j which lies to the right of F, we set the variable x_j to false. The remaining variables are assigned true. Notice that if all the betweenness constraints associated with a clause C_j are satisfied it must be the case that at least one of the literals associated with C_j is assigned true by such an assignment - thus proving our claim.

Thus given a 3-CNF formula ϕ with m clauses we have constructed a betweenness instance I with $m' \leq 6m$ constraints. Further more given an ordering satisfying $(1-\epsilon)m'$ constraints, we can reconstruct an assignment satisfying $m-\epsilon m' \geq m(1-6\epsilon)$ clauses of ϕ.

Corollary 2. *There exists an $\epsilon > 0$, such that satisfying $1 - \epsilon$ fraction of the constraints in a satisfiable instance of the betweenness problems is NP-hard.*

Next we show what can achieved by the obvious randomized algorithm for the betweenness problem.

The natural randomized algorithm for the betweenness problem arranges the points in a random order along the line. The probability that a specific constraint is satisfied by such a randomly chosen order is 1/3. Thus the expected number of constraints satisfied by a random order is at least a third of all the constraints. By the method of conditional probabilities one can find such order in polynomial time. Since this order satisfies 1/3rd of all constraints, it is within 1/3rd of the optimal ordering.

Before going on to more sophisticated techniques for solving this problem, let us examine the main weakness of the above algorithm. We first argue that no algorithm can do better than attempting to satisfy a third of all given constraints. Consider an instance of the betweenness problem on three points with three constraints insisting that each point be between the other two. Clearly we can satisfy only one of the above three constraints, proving the claim. Thus the primary weakness of the above algorithm is not in the (absolute) number of constraints it satisfies, but in the fact that it attempts to do so for every instance of the betweenness problem; even those which are obviously not satisfiable. Thus to achieve better approximation factors, one needs to be able to recognize instances of the betweenness problems which are not satisfiable. But this is an NP-hard task. In fact, Corollary 2 indicates that one cannot even distinguish instances which are satisfiable from those for which an ϵ fraction of the constraints remain unsatisfied under any assignment. In what follows we use a semidefinite relaxation of our problem to distinguish cases which are not satisfiable from cases where at least 50% of the given clauses are satisfiable. We then go on to show that using this relaxation we can achieve a better approximation than the naive randomized algorithm.

3 Semidefinite Programming

In this section we briefly introduce the paradigm of semidefinite programming. We describe why it is solvable in polynomial time. A complementary technique to that of semidefinite programming is the incomplete Cholesky decomposition. We describe how the combination allows one to find embeddings of points in infinite-dimensional space subject to certain constraints.

Definition 3. For positive integers m and n, a semidefinite program is defined over a collection of n^2 real variables $\{x_{ij}\}_{i=1,j=1}^{n,n}$. The input consists of a set of mn^2 real numbers $\{a_{ij}^{(k)}\}_{i=1,j=1,k=1}^{n,n,m}$, a vector of m real numbers $\{b^{(k)}\}_{k=1}^{m}$ and a vector of n^2 real numbers $\{c_{ij}\}_{i=1,j=1}^{n,n}$. The objective is to find $\{x_{ij}\}_{i=1,j=1}^{n,n}$ so as to

$$\begin{array}{rl} \text{maximize} & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{subject to} & \\ \forall k \in \{1,\ldots,m\} & \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^{(k)}x_{ij} \leq b^{(k)}. \\ \text{and} & \text{The matrix } X=\{x_{ij}\} \text{ is symmetric} \\ & \text{and positive semidefinite.} \end{array}$$

Recall that the following are equivalent ways of defining when a symmetric matrix X is positive semidefinite.

1. All the eigenvalues of X are non-negative.
2. For all vectors $y \in \mathcal{R}^n$, $y^T X y \geq 0$.
3. There exists a real matrix V such that $V^T \cdot V = X$.

It is well-known that the ellipsoid algorithm of Khaciyan [11] can be used to find an additive ϵ-approximate solution to any semidefinite program in time polynomial in the input size and the logarithm of $1/\epsilon$ (see for instance [8]).

In order to use the semidefinite programming approach for solving combinatorial optimization problems, one more tool is useful. This is the ability to find a matrix V as guaranteed to exist in part 3 of the definition of positive semidefiniteness above. The method which yields such a matrix is the incomplete Cholesky decomposition.

The matrix V can be used to interpret the solution obtained by the semidefinite programming problem geometrically. Interpret the columns of the $n \times n$ matrix V as n vectors $v_1, \ldots, v_n$ in $\mathcal{R}^n$. Now the variables x_{ij} of the matrix X simply correspond to the inner product of v_i and v_j. Thus a linear constraint on the x_{ij}'s is simply a linear constraint on the inner products of the vectors v_i's. And the objective function is simply a linear function on the inner products.

Thus the following provides a geometric interpretation of SDP:

> Find n vectors $v_1, \ldots, v_n$ so as to maximize the quantity $\sum_{i,j} c_{ij} < v_i, v_j >$, subject to the constraints $\sum_{i,j} a_{ij}^{(k)} < v_i, v_j > \leq b^{(k)}$.

Alternately one can interpret SDP as solving an optimization problem which attempts to find n points in infinite dimensional Euclidean space subject to linear constraints on the squares of the distance between the points. This is done by observing that the square of the distance between points v_i and v_j (denoted d_{ij}^2) is simply $< (v_i - v_j), (v_i - v_j) > = < v_i, v_i > + < v_j, v_j > -2 < v_i, v_j >$. Thus a linear inequality on the d_{ij}^2's is also a linear inequality on the inner products of the v_i's. (Actually the distance square interpretation is equivalent to SDP since we can express $< v_i, v_j >$ as $(d_{i0}^2 + d_{j0}^2 - d_{ij}^2)/2$.) Hence we can use SDP to solve any problem of the form:

> Embed n points in $\mathcal{R}^n$ such that the squares of the distance between the points, denoted d_{ij}, satisfy the constraints $\sum_{i,j} a_{ij}^{(k)} d_{ij}^2 \leq b^{(k)}$ while trying to maximize $\sum_{i,j} c_{ij} d_{ij}^2$.

In what follows we will use the last interpretation of SDP to solve the betweenness problem.

4 The Algorithm

The general idea of our algorithm is to express the betweenness constraints as a set of real quadratic inequalities. By considering an n–dimensional relaxation of the problem, we get an instance of semidefinite programming, and can find a feasible solution in $\mathcal{R}^n$ (if one exists). We study simple geometric properties of this solution set. We use them to argue that a projection of the set on a random line satisfies at least half the betweenness constraints (with high probability). Then we show how to derandomize the algorithm.

Consider a set of m betweenness constraints on n real variables $x_1, \ldots, x_n$. Suppose these constraints are satisfiable, and that $x_1 < x_2 < \ldots < x_n$ is a satisfying linear order. We can clearly embed the points in the unit interval, and assign $x_i = (i-1)/(n-1)$ $(i = 1, \ldots, n)$. Let x_i, x_j, x_k be a triplet such that x_j is required to be between x_i and x_k. For the assignment above, it is readily seen that $(x_i - x_j)^2 + (x_k - x_j)^2 < (x_i - x_k)^2$. The x's are at least $1/(n-1)$ apart and at most 1 apart. Therefore the ratio between the left hand side and the right hand side is maximized when x_i and x_k are extreme points (0 and 1), and x_j is as close as possible to one of them ($1/(n-1)$ or $(n-2)/(n-1)$). For these values, the left hand side is

$$\left(\frac{1}{n-1}\right)^2 + \left(1 - \frac{1}{n-1}\right)^2 = 1 - \frac{2}{n-1} + \frac{2}{(n-1)^2} .$$

Denote this value by α_n. Notice that $\alpha_n = 1 - 1/n + o(1/n)$ depends only on the number of variables.

To set up the SDP instance, we add the inequality

$$d_{i,j}^2 + d_{k,j}^2 \leq \alpha_n d_{i,k}^2$$

for every betweenness constraint "x_j between x_i and x_k". In addition, we add two inequalities for every pair x_i, x_j. These inequalities force the two variables to be neither too close nor too far apart,

$$\left(\frac{1}{n-1}\right)^2 \le d_{i,j}^2 \le 1 \ .$$

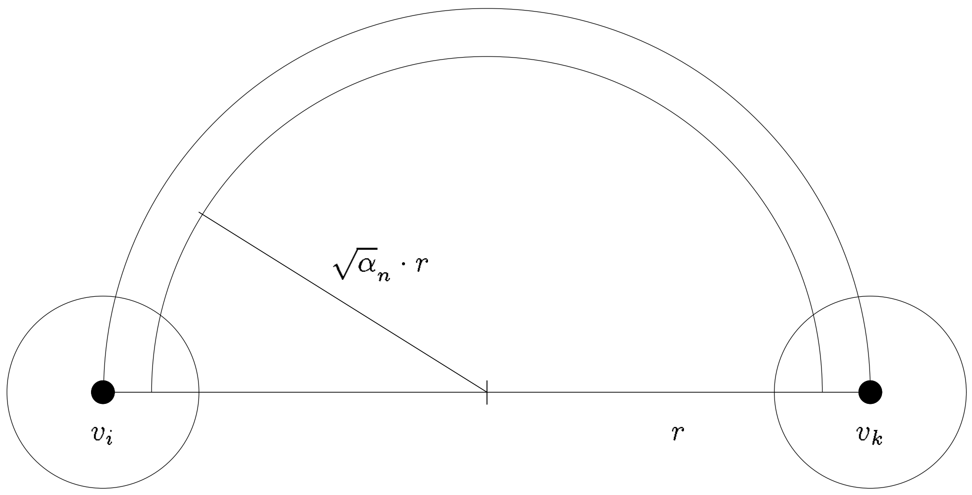

Fig. 1. Possible location for the midpoint v_j

As argued in Section 3, we can use semidefinite programming to test feasibility of these inequalities in $\mathcal{R}^n$, and to find an approximation of a feasible solution (if one exists). Let $v_1, \ldots, v_n \in \mathcal{R}^n$ be a feasible solution, and let $v_i, v_j, v_k \in \mathcal{R}^n$ be a triplet which corresponds to a betweenness constraint. Consider the two dimensional plane through the points v_i, v_j, v_k. Let $2r$ be the distance between v_i and v_k $(1/(n-1) \le 2r \le 1)$. Then v_j, the "midpoint", lies inside the ball of radius $r\sqrt{\alpha}_n$ whose center is the middle $(v_i + v_k)/2$, and outside the two small balls of radius $1/(n-1)$ around v_i and v_k (see figure 1).

This implies that the angle $\theta_{i,j,k} = \angle v_i v_j v_k$ (the angle between the lines $v_i v_j$ and $v_k v_j$) is obtuse. (Right angle correspond to Pythagorean triplets, where equality $d_{i,j}^2 + d_{k,j}^2 = d_{i,k}^2$ holds.) Furthermore, this angle is at least $(1 + O(\sqrt{\alpha}_n))\pi/2 = (1 + O(1/\sqrt{n}))\pi/2$. The algorithm proceeds by picking uniformly at random a line through the origin, and projecting the n points $v_1, \ldots, v_n$ of this random line. Let $x'_1, \ldots, x'_n$ be the n resulting points.

Claim 4. *Let $\theta_{i,j,k}$ denote the angle $\angle v_i v_j v_k$. Then the probability that x'_j lies between x'_i and x'_k equals $\theta_{i,j,k}/\pi$.*

Proof. Instead of considering an arbitrary line through the origin, we consider a parallel line that goes through the point v_j. This does not change the betweenness relation of the projections. Neither is this relation changed when considering the projection of this line on the two dimensional plane defined by v_i, v_j, v_k. Consider the section of the circle defined by the two lines which go through v_j and are perpendicular to the lines $v_i v_j$ and $v_k v_j$. It is not hard to see that only lines going through this section violate the betweenness constraint of the projections. This section occupies an angle of $\pi - \theta_{i,j,k}$ (see figure 2). The claim follows.

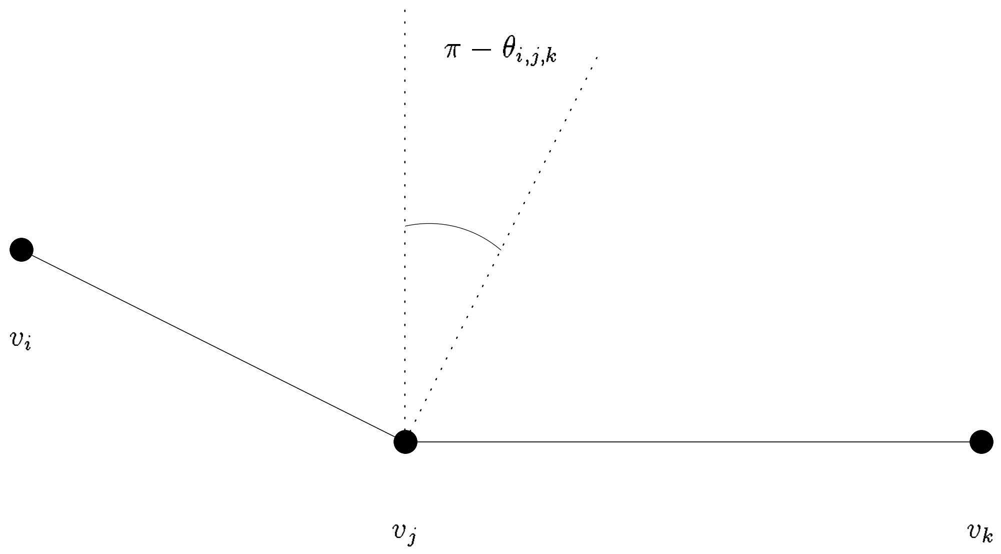

Fig. 2. Lines going through the circular section violate the constraint.

□

Corollary 5. *Suppose the SDP has a feasible solution. Then for any of the m constraints, the probability that x'_j lies between x'_i and x'_k is at least $1/2 + O(1/\sqrt{n})$.*

As a consequence, the expected number of betweenness constraints satisfied by $x'_1, \ldots, x'_n$ is at least $m\,(1/2 + O(1/\sqrt{n})) = m(1/2+o(1))$. Thus we get a randomized polynomial time algorithm which either finds that the constraints are infeasible, or generates a linear order which satisfies at least half the constraints.

We now outline a method for derandomizing our algorithm. Given an embedding of the betweenness problem, we can define a graph and an embedding of the graph in $\mathcal{R}^n$, such that the expected size of the MAX CUT found for this embedding of the graph equals the expected number of betweenness constraints that are satisfied by a random projection.

For every ordered pair of points (v_i, v_j) of the betweenness problem introduce the vertex w_{ij} with embedding $v_i - v_j$. If i, j, k is a betweenness constraint, then put an edge between w_{ij} and w_{kj}. This defines the graph and its embedding.

Now consider any hyperplane through the origin that cuts across the edge between w_{ij} and w_{kj}. Let the slope of the normal to the hyperplane be the vector r. Assume w.l.o.g. that $r.w_{ij} < 0$ and $r.w_{jk} < 0$, then $r.v_i < r.v_j$ and $r.v_j < r.v_k$. Thus j lies between i and k. Conversely if projection onto the vector r satisfies the betweenness constraint for i, j, k; then the edge between w_{ij} and w_{jk} must be cut.

Hariharan and Mahajan [9] give a method to deterministically find a vector r whose cut value equals the expected cut value. They use this algorithm to derandomize the MAX CUT and Max 2SAT algorithm of Goemans and Williamson [5]. By using their algorithm we get vector such that projection on to this vector satisfies as many constraints as satisfiable by a random vector.

Remark: Observe that the above reduction is not a generic reduction from betweenness to MAX CUT; and needs to use the fact that the graph produced for the MAX CUT problem has a specified embedding in order to map a solution of the MAX CUT problem to a solution of the betweenness problem.

5 Tightness of our analysis

In this section we show that our analysis of the semidefinite program is almost tight. We do so by exhibiting two families of instances of the betweenness problem on m constraints, for which the optimum value is at most $m(1/2 + o(1))$ but (a slight perturbation of) the SDP is nevertheless feasible.

The first example is related to the d-dimensional hypercube, and is as follows: For every integer $d > 1$, we construct the instance I_d as follows. I_d has 2^d points corresponding to the 2^d vertices of the d-dimensional hypercube. I_d has $m = \binom{d}{2}2^d$ constraints - one for every simple path of length 2 in the hypercube, with the betweenness constraint expecting the middle vertex of the path to be between the endpoints.

Consider a small perturbation of our SDP, where we have $d_{i,j}^2 + d_{j,k}^2 \leq d_{i,k}^2$ for each betweenness constraint. This SDP is clearly feasible — the natural embedding of the hypercube in d-dimensions (as a hypercube) ensures that every path of length 2 subtends an angle of 90^o at the midpoint.

Now consider a linear ordering of the points. Consider any point p and all the paths which have p as the midpoint: The number of such paths is $\binom{d}{2}$. Now let d_1 of the neighbors of p be on its left and d_2 of its neighbors be on its right (where $d_1 + d_2 = d$). The number of betweenness constraints expecting p to be in the middle, that get satisfied, is $d_1 d_2 \leq d^2/4$. Thus the fraction of betweenness constraints associated with any point, that get satisfied, is at most $(d^2/4)/(d(d-1)/2) = d/(2(d-1)) = 1/2 + 1/(2(d-1)) = 1/2 + o(1)$.

The second example, suggested to us by Michel Goemans, is related to the cuts in the complete graph K_n on n variables. For every integer $n > 1$, we construct the instance C_n as follows. C_n has $n+1$ points, a "center point" c and n "vertices" $v_1, \ldots, v_n$. C_n has $m = \binom{n}{2}$ constraints – one for every edge in the complete graph. For every $1 \le i < k \le n$, we have the betweenness constraint that c is between v_i and v_k.

We now consider the following perturbation of our SDP, where $d^2_{i,j} + d^2_{j,k} \le (1 - 1/n)d^2_{i,k}$ for each betweenness constraint. To see that this SDP is feasible, consider an embedding of a regular n vertex simplex in the $\mathcal{R}^n$ unit sphere. Let w_i and w_k be two vertices of this simplex, then it is well known that their inner product $< w_i, w_k >$ equals $-1/(n-1)$. ***** Madhu – do you have a reference for this ? ***** Plugging in $d^2_{i,k} =< w_i, w_i > + < v_k, v_k > -2 < w_i, w_k >$ we have $d^2 i, k = 2 + 2/(n-1)$, and taking $d^2_{i,0} = d^2_{k,0} = 1$, we have $d^2_{i,0} + d^2_{k,0} = (1 - 1/n)d^2_{i,k}$. Now in order to satisfy the SDP (recall that we required all pairwise distances to be at most 1) we simply scale down the simplex so that the distance between the vertices is 1, embed the center c in the origin, and each vertex v_i in the corresponding simplex vertex. This embedding satisfies all the SDP constraints.

Again, any linear ordering of the $n+1$ points induces a cut in the graph K_n (vertices to the left of c, vertices to the right of c). An edge corresponds to a satisfied betweenness constraint iff the edge is across the cut. Therefore the maximum number of satisfiable constarints equals the sized of a maximum cut in K_n, namely $(n/2)^2 = m(1/2 + o(1))$.

The advantage of this maximum cut example is that it shows tightness of the analysis with respect to quadratic inequalities of the form

$$d^2_{i,j} + d^2_{k,j} \le \beta_n d^2_{i,k} \ ,$$

where $\beta_n = 1 - 1/n - o(1/n)$. Our original SDP has the form

$$d^2_{i,j} + d^2_{k,j} \le \alpha_n d^2_{i,k}$$

where $\alpha_n = 1 - 2/n - o(1/n)$. It is an open problem an example where only $1/2 + o(1)$ of the constraints are satisfiable, yet the original α_n SDP is feasible.

6 Concluding Remarks

Our formulation of the problem as SDP only tested for feasibility of the constraints. It is interesting to see if the inclusion of an appropriate objective function, and possibly of additional inequalities, can be used to improve the performance guarantee of the algorithm. Other approaches to the problem, possibly purely combinatorial ones, are also of interest.

Finally, we remark that metric information can be easily incorporated into our algorithm. As a simple example, suppose that for some of the constraints, we know not only that x_j is between x_i and x_k, but that it is exactly in the middle, namely $x_j = (x_i + x_k)/2$. In this case, we add the inequality

$$d_{i,j}^2 + d_{k,j}^2 \leq d_{i,k}^2/4$$

instead of

$$d_{i,j}^2 + d_{k,j}^2 \leq \alpha_n d_{i,k}^2 \ .$$

Any feasible solution will have v_j exactly in the middle of v_i and v_k, and the same holds with respect to the final projections.

Acknowledgments

We are grateful to Michel Goemans for providing us with the max cut example, and for helpful discussions on semi-definite programming. Many thanks to Amir Ben-Dor for numerous helpful discussions on the betweenness problem. We would also like to thank Ron Shamir for acquainting us with reference [13] and for useful discussions, Oded Goldreich for his comments on an earlier draft of this manuscript, and Amos Beimel for his expert advice on xfig.

References

1. N. Alon and N. Kahale. "Approximating the independence number via the θ-function", Manuscript, August 1994.
2. S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. "Proof Verification and Hardness of Approximation Problems", *Proceedings 33rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 14–23, 1992.
3. D. Cox, M. Burmeister, E. Price, S. Kim, R. Myers, "Radiation Hybrid Mapping: A Somatic Cell Genetic Method for Constructing High Resolution Maps of Mammalian Chromosomes", *Science*, Vol. 250, 1990, pp. 245–250.
4. U. Feige and M. Goemans. "Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT", *Proceedings of the Third Israel Symposium on Theory and Computing Systems,*, 1995.
5. M. Goemans and D. Williamson. ".878-Approximation Algorithms for MAX CUT and MAX 2SAT", *Proceedings of the Twenty-Sixth ACM Symposium on Theory of Computing*, pp. 422-431, 1994.
6. S. Goss and H. Harris, "New Methods for Mapping Genes in Human Chromosomes", *Nature*, Vol. 255, 1975, pp. 680–684.
7. M. Grötschel, L. Lovász and A. Schrijver. "The ellipsoid method and its consequences in combinatorial optimization", *Combinatorica*, 1:169–197, 1981.
8. M. Grötschel, L. Lovász and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin, 1987.
9. R. Hariharan and S. Mahajan. Derandomization of Semidefinite Programming based algorithms. Manuscript, 1995.
10. D. Karger, R. Motwani and M. Sudan. "Approximate graph coloring via semidefinite programming", Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 1994.

11. L. Khaciyan. “A polynomial algorithm in linear programming”, (English translation appears in) *Soviet Mathematics Doklady*, vol. 20, pp. 191–194, 1979.
12. L. Lovász. “On the Shannon capacity of a graph”, *IEEE Transactions on Information Theory*, IT-25:1–7, 1979.
13. J. Opatrny, “Total Ordering Problem”, *SIAM J. Comput.*, vol. 8 No. 1, February 1979, pp. 111–114.