CS 787: Advanced Algorithms

Semidefinite Programming Based Approximations

Instructor: Dieter van Melkebeek

In the precedings lectures we have used linear programming to approximate NP-hard optimization problems. The basic steps were: model the problem as a linear program (LP) with some integrality constraints, solve the linear program relaxation, use the solution to the LP relaxation to get a solution to the original problem, and then prove that the result is close to the optimal solution.

In this lecture, we will use semidefinite programming to approximate NP-hard optimization problems. The general method is similar to how we have used LP. We will see that semidefinite programming is in fact a generalization of linear programming. Even so, semidefinite programs are efficiently solvable under certain conditions which will allow us to use a black-box semidefinite program solver in much the same way that we have used a black-box LP solver.

We then study applications of semidefinite programming to approximation algorithms. Namely, we give a 0.878-approximation for both MaxCut and Max-2SAT problems. 0.878-approximation for Max-2SAT is an improvement over the 0.75 approximation we achieved by randomized LP-rounding. Also, we show how semidefinite programming can be used to color 3-colorable graphs using $\tilde{O}\left(n^{1/4}\right)$ colors, an improvement over the $O(\sqrt{n})$ colors derived in the homework.

1 Applications of semidefinite programming (SDP)

We mention the typical applications of both linear programming and semidefinite programming:

LP: The objective function to be maximized or minimized is a linear combination of integral variables. The LP relaxation is to relax the integrality constraint (e.g., $x \in \{0,1\}$) by replacing it with a linear constraint (e.g., $0 \leq x \leq 1$).

SDP: The objective function is a linear combination of products of pairs of variables. In fact, because there are products of variables, we will usually view them as belonging to $\{-1,1\}$. In the MaxCut problem (which appears later), we can view the variables as indicating which side of the cut each vertex is on: 1 indicating the vertex is on one side, and -1 indicating it is on the other side. This means that two vertices are on the same side of the cut, if the product of their corresponding variables is 1.

2 Semidefinite matrices

Definition 1 (Positive semidefinite)**.** *A matrix $A \in \mathbb{R}^{n\times n}$ is positive semidefinite (PSD), if*

$$(\forall x \in \mathbb{R}^n) \qquad x^T A x \geq 0 \tag{1}$$

If we view A as a bilinear map ($A(x,y) = x^T A y$), then A is PSD, when $A(x,x) \geq 0$ for all $x \in \mathbb{R}^n$. Similarly, a negative semidefinite matrix is defined by replacing $\geq$ with a $\leq$, i.e., a matrix A is negative semidefinite, if for all $x \in \mathbb{R}^n$, $x^T A x \leq 0$. We will usually be dealing with positive semidefinite matrices, so henceforth we will omit "positive."

Facts:

- A matrix A is symmetric if and only if $A = A^T$.
- A matrix A is skew-symmetric if and only if $A = -A^T$.
- Symmetric part of a matrix A is $\frac{A+A^T}{2}$.
- Skew-symmetric part of a matrix A is $\frac{A-A^T}{2}$.
- A matrix A can always be written as the sum of its symmetric part and skew symmetric part:

$$A = \frac{A + A^T}{2} + \frac{A - A^T}{2}$$

- If a matrix A is skew-symmetric, then:

$$\begin{aligned} x^T Ax &= (x^T Ax)^T \\ &= x^T A^T x \\ &= -x^T Ax \\ x^T Ax &= 0 \end{aligned}$$

  The first equality is because the transpose of a 1×1 matrix is itself. This means that we can, in general, ignore the skew-symmetric part of the matrix. We will in general assume that we are dealing with symmetric matrices, and, thus, $x^T Ax = x^T \left(\frac{A+A^T}{2}\right) x$.

- If A and B are PSD, then $A + B$ is also PSD.
- If A is PSD and $c \geq 0$ is a scalar, then cA is also PSD.

Lemma 1. *For a symmetric matrix A, the following are equivalent:*

(1) A is PSD.

(2) All eigenvalues of A are ≥ 0.

(3) $(\exists B \in \mathbb{R}^{n\times n}) \quad A = B^T B$.

(3) states that A is the *Gram matrix* for the vectors $b_1, b_2, \ldots, b_n$, where $B = [b_1, b_2, \ldots, b_n]$: the components of A are the inner products of the pairs of the collection of vectors. Representing A as $B^T B$ is also known as the *Cholesky factorization* of A.

Proof.
(1) $\Rightarrow$ (2): First, notice that because A is symmetric, all eigenvalues are real. Suppose A is PSD, and there is an eigenvalue $\lambda < 0$. Recall that eigenvalue λ satisfies $Ax = \lambda x$ for some eigenvector $x \neq 0$. We have that $x^T Ax = x^T \lambda x = \lambda x^T x < 0$ for $x \neq 0$. This contradicts that A is PSD.

(2) $\Rightarrow$ (3): Because A is symmetric, it has a full orthonormal basis of eigenvectors, i.e., there exists a $\mathcal{V} \in \mathbb{R}^{n\times n}$ with $\mathcal{V}^T\mathcal{V} = I$ and $\Lambda = \text{Diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ such that $A\mathcal{V} = \mathcal{V}\Lambda$. Multiplying both sides by $\mathcal{V}^T$, we have $A = \mathcal{V}\Lambda\mathcal{V}^T$. Because all eigenvalues are non-negative, we can represent Λ as

$\Lambda = \sqrt{\Lambda}\sqrt{\Lambda}$, where $\sqrt{\Lambda} = \mathrm{Diag}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \ldots, \sqrt{\lambda_n})$. Setting $B = \sqrt{\Lambda}^T \mathcal{V}^T$, it is easy to verify that $A = B^T B$.

(3) $\Rightarrow$ (1) directly follows:

$$x^T A x = x^T B^T B x = \|Bx\|_2^2 \geq 0 \tag{2}$$

$\|v\|_2$ is the 2-norm of vector v. □

Notes:

- In time $O(n^3)$, a process similar to Gaussian elimination can be used to construct the Cholesky factorization $A = B^T B$.
- Consider the 2×2 matrix A:

  $$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

  The eigenvalues are non-negative iff both their sum and product is non-negative. Thus, A is PSD iff the following hold:

  $$\sum \lambda = \mathrm{tr}(A) \geq 0 \iff a + c \geq 0$$
  $$\prod \lambda = \det(A) \geq 0 \iff ac - b^2 \geq 0$$

  $\mathrm{tr}(A)$ and $\det(A)$ are the trace and determinant of the square matrix A respectively.

3 Semidefinite programs

A semidefinite program (SDP) is similar to a linear program (LP), except that in general we may deal with linear combinations of matrices, and we have an additional type of constraint at our disposal, i.e., the constraint that some matrix is PSD. We will give a few different standard forms for semidefinite programs. For the first two forms, we will also give the corresponding LP form, and demonstrate that the SDP is a generalization of the LP.

(0) This form is convenient as our first example, because the similarity between the LP and SDP is very apparent.

$$\text{LP:} \quad \min \sum_{j=1}^{m} c_j x_j \quad \text{s.t.} \quad \sum_{j=1}^{m} a_{k,j} x_j \geq b_k \quad (\forall\, k = [n])$$

$$\text{SDP:} \quad \min \sum_{j=1}^{m} c_j x_j \quad \text{s.t.} \quad \sum_{j=1}^{m} A_{k,j} x_j \succeq B_k \quad (\forall\, k = [n])$$

Notes:

- $A_{k,j}, B_k \in \mathbb{R}^{n_k \times n_k}$ are (symmetric) matrices.
- The x_j and c_j are scalars in both cases.

- The semidefinite form is the same as the linear programming form, except that the linear inequality constraints have been replaced by semidefinite matrix constraints.
- $A \succeq B$ means that $A - B$ is PSD.
- Note that the SDP is a generalization of the LP. Setting $A_{k,j}$ and B_k as 1×1 matrices, we can notice that a 1×1 matrix is PSD, if it is non-negative.

(1)

$$\text{LP:} \quad \min \sum_{j=1}^{m} c_j x_j \quad \text{s.t.} \quad \sum_{j=1}^{m} a_{k,j} x_j = b_k \quad (\forall\, k = [n]) \quad x \geq 0$$

$$\text{SDP:} \quad \min \sum_{i,j} c_{i,j} x_{i,j} \quad \text{s.t.} \quad \sum_{i,j} a_{k,i,j} x_{i,j} = b_k \quad (\forall\, k = [n]) \quad x \succeq 0$$

Notes:

- For the SDP, $x \in \mathbb{R}^{m \times m}$ is a (symmetric) matrix.
- Recall the method for converting between this LP formulation and the previous one. Inequality constraints can be replaced by equality constraints and the addition of slack variables. Unconstrained variables can be represented by the difference of two constrained variables. Going the other way is easy. An equality constraint can be replaced by two inequality constraints, and non-negativity can be just added in.
- Similarly, both SDP forms are interchangeable as well. Inequality constraints can be written as equalities with the aid of slack variables and slack matrices (difference of right-hand and left-hand sides). Each of these slack matrices needs to be positive semidefinite, and we can guarantee this by including them as a diagonal block matrix in the single semidefiniteness constraint. The other direction is again, easy.

(2) The final form, we mention, is known as a Vector Program.

$$\text{SDP:} \quad \min \sum_{i,j} c_{i,j} \langle y_i, y_j \rangle \quad \text{s.t.} \quad \sum_{i,j} a_{k,i,j} \langle y_i, y_j \rangle = b_k \quad (\forall\, k = [n])$$

Notes:

- Each $y_i \in \mathbb{R}^m$, and $\langle y_i, y_j \rangle$ denotes the inner product of y_i and y_j.
- Notice that form (2) SDP is similar to form (1) SDP. If the x in form (1) is viewed as the Gram matrix of the y_i's in form (2), it can be seen that these forms are equivalent.
- The objective function and constraints are linear combinations of inner products of pairs. In the 1-dimensional case, these would be products of pairs of variables. Recall that this is the form we mentioned at the beginning of the lecture, and the Vector Program form is often useful in approximation algorithms.

4 Solving Semidefinite Programs

Linear programs have the nice property that we can find a point with optimal value in polynomial time. This is something we cannot hope for in the general case of semidefinite programs. First of all, even if the input to the semidefinite program consists of all rational numbers, the value of an optimal solution may not be rational. Thus, we cannot hope to get an exact solution in general, and we are forced to resort to an approximate solution.

Example:

$$\begin{aligned} &\max\ x_1 \\ &\text{s.t.} \begin{bmatrix} 1 & x_1 \\ x_1 & 2 \end{bmatrix} \succeq 0 \end{aligned}$$

By applying the result for a 2×2 PSD matrix from section 2, we get:

$$\begin{aligned} 3 &\geq 0 \\ 2 &\geq x_1^2 \\ \text{OPT} &= \sqrt{2} \end{aligned}$$

⊠

Furthermore, the bit representation of a smallest feasible solution can potentially be exponential in the input size. Clearly, we will not be able to use semidefinite programming to get a feasible solutions in polynomial time for such SDPs.

Example: Consider the optimization problem:

$$\min\{x_n : x_0 = 2, \text{and } x_i \geq x_{i-1}^2 \text{ for } i = 1, 2, \ldots, n\}$$

Clearly, $x_n = x_0^{2^n} = 2^{2^n}$ is the solution, which requires exponential number of bits to represent. However, this can be written as the following semidefinite program:

$$\begin{aligned} &\min x_n \\ &\text{s.t.} \\ &\qquad \begin{bmatrix} x_i & x_{i-1} \\ x_{i-1} & 1 \end{bmatrix} \succeq 0 \quad i = 1, 2, \ldots, n \\ &\qquad x_0 = 2 \end{aligned}$$

⊠

Modulo these two limitations, semidefinite programs can be solved efficiently using either the ellipsoid method or an interior point method. We will use semidefinite programming as a black box by using the following theorem, which we state without proof.

Theorem 1. *Given a semidefinite program, $\epsilon > 0$, and $R > 0$, if the feasible region lies within a ball of radius R centered at the origin, then we can find a feasible solution such that the objective function is at most $\text{OPT} + \epsilon$ in time $\text{POLY}(n, \log \frac{1}{\epsilon}, \log R)$, where n is the size of the input.*

5 Maximum Cut

Our first application of this semidefinite programming will be to the MAXCUT problem. Recall that the corresponding minimization problem to this NP-hard problem is in P.

Definition 2 (MAXCUT)**.** *Given a graph $G = (V, E)$ with non-negative edge weights $w : E \to [0, \infty)$, find a cut $(S, \bar{S})$ such that the following quantity is maximized:*

$$\sum_{e \in E \cap S \times \bar{S}} w(e)$$

Note that there is a simple randomized algorithm that gives an expected approximation factor of at least $\frac{1}{2}$. For every vertex v, use a single random bit to determine whether or not $v \in S$. For any edge, the probability that it is cut is equal to the probability that both vertices are on opposite sides, which is exactly $\frac{1}{2}$. Thus, the expected contribution of every edge e is $\frac{w(e)}{2}$, and the expected weight of the cut is $\sum_{e \in E} \frac{w(e)}{2}$. However, we know the optimal solution is upper bounded by the weight of all edges, i.e., $\text{OPT} \leq \sum_{e \in E} w(e)$. Thus, the expected weight of the cut is at least $\frac{OPT}{2}$, i.e., $\text{E}[\text{weight of cut}] \geq \frac{\text{OPT}}{2}$.

Now, we will use semidefinite programming to get a better approximation, namely a factor 0.878 approximation. Under some reasonable complexity theoretic assumptions somewhat stronger than $\mathsf{P} \neq \mathsf{NP}$, this is the best we can do. This was actually the first result to use semidefinite programming to get a good approximation for an NP-hard optimization problem, so this construction is of historical significance as well.

First, we need to formulate MAXCUT as an integer semidefinite program. One way to do this is as follows:

$$\begin{aligned} \max \sum_{e=(u,v) \in E} w(e) \cdot \frac{1 - x_u x_v}{2} \\ \text{s.t.} \quad x_u \in \{-1, 1\} \quad (\forall u \in V) \end{aligned}$$

For each vertex $u \in V$, we have a corresponding variable indicating whether or not $u \in S$ (i.e., 1 or -1). For any edge $e = (u, v)$, if u and v are on the same side of the cut, then the product of the corresponding variables is one and its term $(1 - x_u x_v)$ in the sum is zero. If u and v are on opposite sides of the cut, then $w(e) \cdot \frac{1 - x_u x_v}{2} = w(e) \cdot \frac{2}{2} = w(e)$ is added. So for any possible setting of S, the objective function gives exactly the value of the cut. Note that this formulation fits our scheme where the objective function and constraints depend linearly on the products of pairs of variables, where each variable is ± 1.

For the semidefinite program relaxation, we replace each of integer variables with vectors in $\mathbb{R}^n$. The product in the objective function is replaced by the inner product of the variables, and the ± 1 constraint is replaced by a requirement that the norm of each vector is 1.

$$\begin{aligned} \max \sum_{e=(u,v)\in E} w(e) \cdot \frac{1-\langle x_u, x_v\rangle}{2} \\ \text{s.t.} \quad x_u \in \mathbb{R}^n \quad (\forall u \in V) \\ \langle x_u, x_u \rangle = 1 \quad (\forall u \in V) \end{aligned}$$

Note that if each vector was 1-dimensional, then we would have the exact same formulation as before. It is similarly clear that this is a relaxation, as any earlier solution is representable in this format; just use one component in each vector to represent the corresponding scalar and set the other components to zero.

Now, we can solve the semidefinite program to get a solution x^* where the objective function is at least $\text{OPT}_{\text{SDP}} - \epsilon$. Since the feasible set lies within the ball of radius n at the origin, this can be done in time $\textsc{Poly}(n, \log \frac{1}{\epsilon})$.

Finally, we need to come up with a way of partitioning these vectors. This can be done with randomized rounding akin to what we did in the linear programming setting. More specifically, we will use a random hyperplane through the origin to separate the vertices into either side of the cut. We want both relaxed and unrelaxed objective functions to behave similarly. That is, if the inner product between two vectors is close to one, then we would like that edge not to be cut. Since the angle between vertices with a large inner product is small, the hyperplane would only cut those vertices with a low probability.

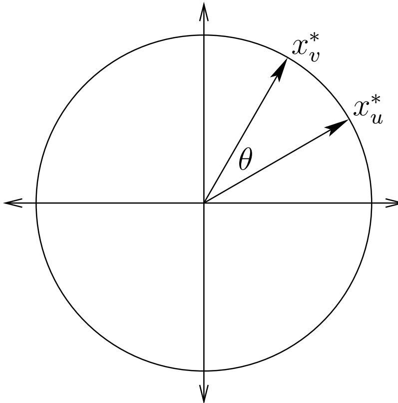

Figure 1: Plane defined by x_u^* and x_v^*

More formally, pick a vector $r \in \mathbb{R}^n$ with $\langle r, r\rangle = 1$ uniformly at random and define:

$$S = \{u \in V : \langle x_u^*, r\rangle \geq 0\}$$

Now, we calculate the probability that the vertices u and v of a fixed edge end up on different sides of the cut. Consider the plane defined by x_u^* and x_v^*, and recall that they are both on the unit circle. The orientation of the intersection of the random hyperplane with this plane will be uniformly random. Thus, if θ is the angle between them,

$$
\begin{aligned}
\Pr[u, v \text{ are separated}] &= \frac{\theta}{\pi} \\
&= \frac{\cos^{-1}\langle x_u^*, x_v^* \rangle}{\pi}
\end{aligned}
$$

Now we can derive an expression for the expected weight of the cut, but we need to linearize it so that we can relate it back to the semidefinite program output.

Lemma 2. *The largest value of α for which the inequality $\frac{\cos^{-1} z}{\pi} \geq \frac{\alpha}{2}(1 - z)$ holds over $z \in [-1, 1]$ is approximately* 0.878*.*

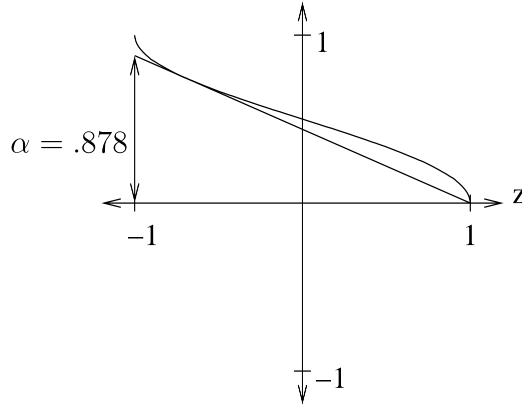

Figure 2: Functions $\frac{\cos^{-1}(z)}{\pi}$ and $\frac{\alpha}{2}(1 - z)$ on interval [-1,1]

We can prove this lemma by applying some calculus. Using this lemma, we can arrive at the approximation factor as follows:

$$
\begin{aligned}
\mathrm{E}[\text{weight of cut}] &= \sum_{e=(u,v)\in E} w(e) \cdot \Pr[u, v \text{ are separated}] \\
&= \sum_{e=(u,v)\in E} w(e) \cdot \frac{\cos^{-1}\langle x_u^*, x_v^* \rangle}{\pi} \\
&\geq \sum_{e=(u,v)\in E} w(e) \cdot \frac{\alpha}{2}(1 - \langle x_u^*, x_v^* \rangle) \\
&= \alpha \cdot \sum_{e=(u,v)\in E} w(e) \cdot \frac{1 - \langle x_u^*, x_v^* \rangle}{2} \\
&= \alpha \cdot (\mathrm{OPT}_{\mathrm{SDP}} - \epsilon)
\end{aligned}
$$

Thus, we have a 0.878-factor randomized approximation algorithm to solve MaxCut problem.

6 Maximum 2-SAT

Recall that MAX-2SAT is the problem of finding an assignment maximizing the weighted sum of clauses satisfied in a given 2-CNF formula. If we view the 2-CNF clauses in analogy to edges of a graph, we can see that this is very similar in nature to the MAXCUT problem—we must partition the variables into two sets, maximizing the weight of the clauses that have at least one literal falling on the "true" side. Therefore, it seems likely that using a similar approach to approximating MAXCUT will be useful in approximating MAX-2SAT.

The approach we used to approximate MAXCUT is as follows. First, we formulate the MAXCUT objective function as an integer program where the objective function is linear in the product of pairs of variables x_i, which take on values of ± 1 to indicate on which side of the cut the corresponding vertex is placed. We then solve the relaxation of that program, where we take the variables x_i to be vectors in $\mathbb{R}^n$ with magnitude 1, i.e., $\langle x_i, x_i \rangle = 1$. Such a relaxation can be seen to fit within the vector program formulation of a semidefinite program. To construct a solution to the MAXCUT problem from this semidefinite formulation without worsening the objective value too much, we select a random hyperplane to partition the solution vectors, and partition the corresponding vertices accordingly.

To derive an approximation algorithm for MAX-2SAT, we will use a similar approach. Although the problems are similar, they are not exactly the same, since the objective in MAX-2SAT counts not only clauses whose literals are in different partitions, but also clauses with both literals (or the only literal, in the case of unit clauses) on the "true" side.

6.1 A First Attempt

Our first attempt is as follows. Given a 2-CNF formula, we convert the variables into ± 1-valued variables with the convention that $+1$ represents "true" and -1 "false," and adjust the objective function to account for this. Namely, for a clause $(x_i \vee x_j)$ we want to find a function that will evaluate to 1 when at least one of x_i and x_j takes the value of 1, and 0 otherwise (i.e., if both x_i and x_j take on value -1). The following function accomplishes the task.

$$\text{VALUE}(x_i \vee x_j) = 1 - \frac{(1 - x_i)(1 - x_j)}{4}$$

Additionally, we need such a function for unit clauses:

$$\text{VALUE}(x_i) = \frac{1 + x_i}{2}$$

Now, we can write the MAX-2SAT objective function (W) as the following, where w_j is the weight of the jth clause C_j.

$$W = \sum_j w_j \, \text{VALUE}(C_j)$$

Such a formulation successfully adjusts the problem of MAX-2SAT to the setting where the variables take ± 1 values. Unfortunately, the objective function is not linear in only products of pairs of variables, so we cannot relax such an integer program into a SDP in the vector program form as desired. For example, expanding $\text{VALUE}(x_i \vee x_j)$ shows that it is linear in $x_i x_j$ as well as in x_i and x_j.

6.2 A Second Attempt

Our second attempt fixes this problem by introducing a variable $x_0 \in \{\pm 1\}$, which we will use to essentially define which value of ± 1 corresponds to "true" for the remaining variables x_i for $1 \le i \le n$. Specifically, we consider x_i to be "true" if and only if $x_i = x_0$ for $1 \le i \le n$. We need to modify the VALUE functions to accommodate this change. To do so, we replace the variables x_i by $x_0 x_i$, so that when $x_i = x_0$ (indicating x_i is "true"), $x_0 x_i = 1$, and when $x_i \ne x_0$ (indicating x_i is "false"), $x_0 x_i = -1$, so such a substitution results in the correct behavior:

$$\begin{aligned}
\text{VALUE}(x_i \vee x_j) &= 1 - \frac{(1 - x_0 x_i)(1 - x_0 x_j)}{4} \\
&= \frac{1}{4} \cdot (3 + x_0 x_i + x_0 x_j - x_i x_j) \\
&= \frac{1}{4} \cdot ((1 + x_0 x_i) + (1 + x_0 x_j) + (1 - x_i x_j)) \\
\text{VALUE}(x_i) &= \frac{1 + x_0 x_i}{2}.
\end{aligned}$$

Now, when we consider the objective function $\sum_j w_j \, \text{VALUE}(C_j)$, we can see that it will expand to be linear in products of pairs of variables. The only terms, which may be problematic, are the $x_0^2 x_i x_j$ terms of $\text{VALUE}(x_i \vee x_j)$. However, we can eliminate the multiplicative factor of x_0^2, since $x_0^2 = 1$. In fact, by taking advantage of the form of the VALUE functions, we can write the objective function as the following for $a_{ij}, b_{ij} \ge 0$:

$$W = \sum_{0 \le i \le j \le n} (a_{ij}(1 + x_i x_j) + b_{ij}(1 - x_i x_j)) \tag{3}$$

6.3 Finding an Assignment

We have now found an integer programming formulation of MAX-2SAT, which we can relax to a SDP in the form of a vector program. We consider the variables x_i to be unit vectors in $\mathbb{R}^m$, and the objective function to be linear in the inner products of these vectors, which can be solved as a semidefinite program. The semidefinite relaxation is as follows:

$$\begin{aligned}
\max \sum_{0 \le i \le j \le n} & (a_{ij}(1 + \langle x_i, x_j \rangle) + b_{ij}(1 - \langle x_i, x_j \rangle)) \\
\text{s.t.} \quad & x_i \in \mathbb{R}^m \quad (i = 0, 1, \ldots, n) \\
& \langle x_i, x_i \rangle = 1 \quad (i = 0, 1, \ldots, n)
\end{aligned}$$

Let x^* be the solution obtained by solving such a program. Note that solution x^* will have an objective value ϵ away from the actual optimum, since we only know how to approximate the solution to an SDP. We will now develop a true/false assignment to the problem's original variables, which does not worsen the objective function too much. Again, we will use a randomly chosen hyperplane to achieve this goal. Namely, a variable will receive the value "true" if and only if its corresponding solution vector falls on the same side of the plane as the vector x_0^*, otherwise "false." More formally,

Pick $r \in \mathbb{R}^m$ such that $\langle r, r \rangle = 1$.

for $i = 1$ **to** n
  if $\langle x_i^*, r\rangle\langle x_0^*, r\rangle \geq 0$
    // x_i^* is in the same side as x_0^*
    Set x_i to TRUE
  else
    Set x_i to FALSE

6.4 Analysis of Approximation Ratio

We now look at the expected weight of satisfied clauses from such an assignment, using the objective function in the form given by (3), which can be written as:

$$E[W] = 2 \cdot \sum_{0 \leq i \leq j \leq n} a_{ij} \Pr[x_i = x_j] + b_{ij} \Pr[x_i \neq x_j] \tag{4}$$

$\Pr[x_i \neq x_j]$ in this problem is same as $\Pr[(i, j) \text{ is cut}]$ in MaxCut problem, i.e., both x_i^* and x_j^* are on different sides of r. The same analysis will give a lower bound on $\Pr[x_i \neq x_j]$, where $\alpha \approx 0.878$.

$$\Pr[x_i \neq x_j] \geq \frac{\alpha}{2}(1 - \langle x_i^*, x_j^*\rangle)$$

We now use a similar approach to lower bound $\Pr[x_i = x_j]$:

$$\begin{aligned}
\Pr[x_i = x_j] &= 1 - \frac{\theta_{ij}}{\pi} && \text{(where } \theta_{ij} \text{ is the angle separating } x_i, x_j\text{)} \\
&= \frac{\pi - \theta_{ij}}{\pi} \\
&\geq \frac{\alpha}{2}(1 - \cos(\pi - \theta_{ij})) && \text{(using analysis for MaxCut applied to angle } \pi - \theta_{ij}\text{)} \\
&= \frac{\alpha}{2}(1 + \langle x_i^*, x_j^*\rangle) && \text{(since } \cos \pi - \theta_{ij} = -\cos\theta_{ij}\text{)}
\end{aligned}$$

Therefore, plugging these lower bounds into eq. (4) gives:

$$\begin{aligned}
E[W] &\geq \alpha \cdot \sum_{0 \leq i \leq j \leq n} \left(a_{ij}(1 + \langle x_i^*, x_j^*\rangle) + b_{ij}(1 - \langle x_i^*, x_j^*\rangle)\right) \\
&= \alpha \cdot (\mathrm{OPT}_{\mathrm{SDP}} - \epsilon)
\end{aligned}$$

Therefore, we have arrived at a factor $\alpha \approx 0.878$ approximation algorithm for Max-2SAT. The main novelty of this application is that we needed to introduce an auxiliary variable to obtain an objective function, which was linear in the product of variables.

The best known approximation to Max-2SAT also uses semidefinite programming to achieve a 0.931-approximation, while the best known lower bound is 0.986 (assuming $\mathsf{P} \neq \mathsf{NP}$), so there still exists a gap.

7 Coloring 3-Colorable Graphs

We now give an application of SDP to the problem of coloring a three-colorable graph. Similarly to the MaxCut formulation, this will be expressed as a semidefinite programming problem where

we'll associate a unit vector with every vertex. We formulate the problem with the goal that, in the optimal solution, vertices connected by an edge will be assigned vectors "far apart" (i.e., vectors with a "large" inner product). Then, we will try to color the graph using the separation technique: cluster the vectors into groups, and color all vectors in the same group with the same color. This will not produce a perfect solution; we will strive for a semicoloring, i.e., one that violates the coloring rules on at most $\frac{n}{4}$ edges. After the algorithm executes, we will reapply it on the remaining $\frac{n}{2}$ vertices that still need a proper coloring (these are the vertices connected by the $\frac{n}{4}$ edges and currently assigned the same color). We'll repeat until everything is properly colored (this will take at most $\log n$ additional runs). In each run of the algorithm, we will use a new set of colors. After $\log n$ phases, the entire graph is colored properly with at most $\log n$ times the maximum number of colors in a phase.

7.1 Vector Program for Coloring

The goal for our formulation is to maximize the distance between vectors, whose corresponding vertices are connected by edges. This will aid us in separating the maximum number of connected vertices, and, hence, properly coloring the maximum number of vertices. We now express the above scheme as the following vector program:

$$\begin{aligned} \min\ & y \\ \text{s.t.}\ & x_u \in \mathbb{R}^n \quad (\forall\, u \in V) \\ & \langle x_u, x_v \rangle \le y \quad (\forall\, (u,v) \in E) \\ & \langle x_u, x_u \rangle = 1 \quad (\forall\, u \in V) \end{aligned}$$

Note that the above is not in SDP standard form due to the scalar y. The reformulation is left as an exercise to the reader. The following claim will be used in the analysis of the SDP.

Claim 1. *For a three-colorable graph,* $\mathrm{OPT}_{SDP} \le -\frac{1}{2}$.

Proof. Since the graph is three-colorable, we first pick three vertices—one per color. The maximum pairwise separation between these vertices is achieved, when the angle θ between any two is $\frac{2\pi}{3}$. The bound follows from $\cos\frac{2\pi}{3} = -\frac{1}{2}$. □

7.2 Clustering vectors

Let x^* be the solution to the SDP above. Our next task is to cluster the vectors.

Idea 1 Use a random hyperplane approach, as described in the previous lecture. Then, with good probability, two vectors that are "far apart" will be separated. However, this guarantee does not hold for *all* pairs of vectors in any of the two clusters. To strengthen this, we'll choose t hyperplanes, thus, introducing at most 2^t cells, each of which will be assigned a distinct color.

Idea 2 Associate colors with the t random vectors as above rather than with cells induced by these hyperplanes. Color the vertex whose inner product with that vector is largest with its color.

7.3 Cell-based clustering

Pick t random vectors r_i with $\langle r_i, r_i \rangle = 1$. Consider the 2^t cells induced by the hyperplanes, and color all vertices u with x_u^* in a given cell with the same color. The quality of our semicoloring is as follows. For a fixed $(u, v) \in E$.

$$\begin{aligned}
\Pr[u, v \text{ get same color}] &= \prod_{1 \le i \le t} \Pr[x_u^*, x_v^* \text{ are on the same side of the } i^{\text{th}} \text{ hyperplane}] \\
&\le \left(\frac{\pi - \theta_{uv}}{\pi}\right)^t \\
&\le \frac{1}{3^t}
\end{aligned}$$

The angle θ_{uv} in the above equations is the angle between vectors x_u^* and x_v^*. The last two inequalities follow from the fact that $\theta_{uv} \ge \frac{2\pi}{3}$, which is due to the following:

$$\begin{aligned}
\cos \theta_{uv} &= \langle x_u^*, x_v^* \rangle \\
&\le y^* \\
&\le -\frac{1}{2}
\end{aligned}$$

For semicoloring we need to guarantee that only a few edges have end points of the same color. Let B denote the number of edges badly colored, i.e., the edges whose end-points are colored the same. For a graph G with maximum degree δ, the expected number of edges with endpoints colored the same is given by:

$$\begin{aligned}
\mathrm{E}[B] &\le \frac{1}{3^t} \cdot |E| \\
&\le \frac{1}{3^t} \cdot \frac{\delta}{2} \cdot n
\end{aligned}$$

We want this quantity to be at most $\frac{n}{8}$ so that using Markov's inequality, we can achieve a semicoloring of at most $\frac{n}{4}$ badly edges colored with probability of at least $\frac{1}{2}$.

$$\begin{aligned}
\Pr\left[B \ge \frac{n}{4}\right] &\le \frac{\mathrm{E}[B]}{\frac{n}{4}} \\
&\le \frac{\frac{n}{8}}{\frac{n}{4}} = \frac{1}{2} \\
\Pr\left[B \le \frac{n}{4}\right] &\ge \frac{1}{2}
\end{aligned}$$

We can achieve the expected number of badly colored edges B of $\frac{n}{8}$, by setting $t = \log_3(4\delta)$. Thus, the number of colors used is given by:

$$
\begin{aligned}
2^t &= (4\delta)^{\log_3 2} \\
&\leq O(n^{\log_3 2}) \qquad (\text{since } \ \delta \leq n)
\end{aligned}
$$

This approach does not work well for graphs with high maximum vertex degree. To improve this result for graphs with high-degree vertices, we deal with such vertices in the way we solved in the homework problem involving 3-coloring. The improvement is achieved by adding an extra step, which first colors neighbors of vertices with high degree using two colors.

7.3.1 Hybrid Approach

As stated above, let's color vertices of degree greater than d and their neighbors using the HW solution, where we color the neighbor with 2 colors, since they are 2-colorable. Then, we will run the clustering technique on the remaining graph, where $\delta = d$, using a new set of colors. The number of colors used is given by:

$$
1 + \frac{n}{d} \cdot 2 + (4d)^{\log_3 2}
$$

Up to a constant factor, this function will be at optimum, when the two operands are equal, which gives the following constraint:

$$
\frac{2n}{d} = (4d)^{\log_3 2}
$$

This happens when $d \approx \theta(n^{0.613})$. Thus, the number of colors used is $\tilde{\theta}(n^{0.387})$. We can drive down the exponent 0.387 to a factor of $\frac{1}{4}$ using the following clustering technique.

7.4 Vector-based clustering

Pick t random vectors r_i with $\langle r_i, r_i \rangle = 1$. Associate a color with each r_i, and color all vertices u for which $\max(x_u^*, r_j)$ is realized for $j = i$ with that color. We omit the analysis of this scheme.

$$
\Pr[u, v \text{ get the same color}] \leq \tilde{O}\left(\frac{1}{t^3}\right)
$$

Choosing $t \approx \theta(\delta^{\frac{1}{3}})$ guarantees a semicoloring with high probability. Coloring via the hybrid approach, we get the following bound.

$$
\text{Maximum number of colors used } \leq (1 + \frac{2 \cdot n}{d}) + c \cdot d^{\frac{1}{3}}
$$

Equating the two operands, we get $d^{\frac{4}{3}} = n$, when $d = \theta\left(n^{\frac{3}{4}}\right)$. This achieves an $O\left(n^{\frac{1}{4}}\right)$ algorithm.

Note that the best known approximation for coloring 3-colorable graphs achieves $\tilde{\theta}\left(n^{\frac{3}{14}}\right)$ colors. The algorithm also uses semidefinite programming with an improved hybrid approach, which considers level-2 neighbors.