0.878-approximation for the Max-Cut problem

Lecture by Divya Padmanabhan

Scribe: Sreejith

Abstract. The Max-Cut problem is a well studied graph theoretic problem where one is interested in finding the cut of largest value in a given graph. While the problem is NP hard, it finds several applications in clustering, VLSI design and more generally in network domains. For many years, various researchers had worked with different kinds of approximation techniques but could not get beyond a guarantee of 0.5. In 1995, Goemans and Williamson proposed an approximation using convex optimization (particularly with semi-definite programming) and were able to obtain a highly improved guarantee of 0.878. This is the best known approximation guarantee for the max-cut problem today. In this talk I will briefly introduce semi-definite programming for those new to the topic and provide a proof of the Goemans-Williamson result.

How to read this writeup? The shaded region contain side remarks and/or technical details. The reader can skip them without losing the plot.

1 The maxcut problem

We are interested in weighted undirected graphs - undirected graphs where every edge (i, j) has a weight w_{ij}. Since the graph is undirected $w_{ji} = w_{ij}$. We will denote such graphs by $G = (V, E)$ where V is the set of vertices and E is the symmetric matrix where the entry $E[i, j] = w_{ij}$ is the weight of the edge (i, j). Henceforth we will call weighted undirected graphs as graphs.

We fix the graph $G = (V, E)$ and denote the sum of the weights by W.

$$W = \frac{1}{2} \sum_{i \in V} \sum_{j \in V} w_{ij} \tag{1}$$

A cut in a graph is a set $S \subseteq V$. We also denote the complement of S by $T = V \backslash S$. Note that a cut partitions the sets of vertices into two parts S and T. The set of edges, on the other hand, are partitioned into three parts. One part contains all the edges in S, and another part has all the edges in T. We are interested in the third type of edges - those split by the cut. One vertex of such an edge is in S, and another vertex is in T. There is a natural way to assign a weight to a cut S - the sum of weights of the split edges.

$$\mathcal{W}(S) ::= \sum_{(i,j) \in S \times T} w_{i,j} \tag{2}$$

We are interested in finding the cut with the maximum weight.

The maxcut problem: Given a graph, find a cut of maximum weight.

Note that there might be multiple cuts with the same maximum weight. Finding one such cut is sufficient. Let us look at an example.

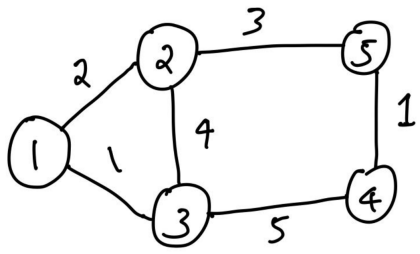


Fig. 1. A weighted graph

Example 1. Some of the cuts and associated weights of Fig. 1 are

1. For $S = \{1, 2, 3\}$, $\mathcal{W}(S) = 8$.
2. For $S = \{1, 2, 3, 4\}$, $\mathcal{W}(S) = 4$.
3. For $S = \emptyset$, $\mathcal{W}(S) = 0$.
4. For $S = \{1, 3, 5\}$, $\mathcal{W}(S) = 15$.

The maxcut is the cut $S = \{1, 3, 5\}$.

We make the following assumption regarding the input graphs.

Assumption: The weights w_{ij} are non-negative integers.

This is not a severe restriction since any graph with negative or rational weights can be reduced to a graph with non-negative integer weights.

Removing the assumption: To remove negative entries, add all the weights with the maximum absolute value of the weights. To remove fractional weights multiply all the weights with the lcm of the denominators. Neither of these reductions changes the maximum cut set. Moreover, the input size stays polynomial. The lcm of n numbers $\leq k$ is bound by k^n and takes $O(n \log k)$ space. This is polynomial in the input size.

Computational complexity of maxcut: The decision version (a Yes/No problem) of the maxcut optimization problem is

maxcut decision version: Given a graph and a number k, does there exists a cut of weight at least k?

The above problem is NP-complete. It is easy to see that the problem is in NP- guess the cut S and verify its weight is at least k. We skip the proof of hardness.

It is also easy to see that if we can solve the optimization problem in polynomial time, then the decision version is in P. What about the other direction: Can we solve the

optimization problem in polynomial time if the decision version is in P? Here is a first attempt. Run the decision version for the weight k ranging from 0 to W (sum of all weights). The largest k for which the algorithm returns Yes is the weight of the maxcut. To find the maxcut from this k, we can do the following. Pick an edge (i, j) and force this edge to be in the maxcut by giving it a weight of $W + w_{ij}$. If this edge was in the original maxcut the decision version should return Yes for the new graph and weight $W + k$; otherwise, it should return No. Once the decision on edge (i, j) is made, pick another edge and repeat the procedure (with a slight change). If the previous edge was in the maxcut, update the weight of that edge to $W + w_{ij}$ and continue; if the edge was not in maxcut we go back to our original graph. This procedure is repeated at most m times where m is the number of edges. The total running time is, therefore, a polynomial times W. Unfortunately, this is pseudo-polynomial time since W is "exponential in the input size" because the input weights are in binary notation. We need to reduce the running time to $\log W$. In the second attempt, we do a binary search from 0 to W. We run the decision version with $k = \frac{W}{2}$ and based on whether the answer is Yes or No, we search the range $[0, W/2]$ or $[W/2, W]$.
In short, the optimization and decision versions are "similar" in their running time. Given that the decision problem is NP-hard, there is little hope of a polynomial-time algorithm for the optimization version.

It is known that unless P=NP, maxcut cannot be solved in polynomial time. In many practical applications, an "approximation" to the best solution is better than no solution. Does there exist a polynomial-time algorithm that can approximate the maxcut? Before we answer that, we need to define the meaning of an approximate solution. For an input graph G, we denote by $\mathrm{OPT}(G)$ the weight associated with the maxcut. That is,

$$\mathrm{OPT}(G) ::= \max\{\mathcal{W}(S) \mid S \text{ is a subset of vertices in } G\}$$

An approximation algorithm aims to find a cut whose weight is very near $\mathrm{OPT}(G)$. Let us assume you have written such an approximation algorithm $\mathcal{A}$. Clearly,

$$\mathcal{A}(G) \leq \mathrm{OPT}(G)$$

We say that algorithm $\mathcal{A}$ is an α-approximation for an $\alpha \in (0, 1]$ if

$$\alpha\ \mathrm{OPT}(G) \leq \mathcal{A}(G), \qquad \text{(for all input instance } G)$$

The larger the α, the better the approximation algorithm. We aim to show that maxcut has a 0.878-approximation. That is, we give an algorithm $\mathcal{A}$ such that

$$0.878\ \mathrm{OPT}(G) \leq \mathcal{A}(G) \leq \mathrm{OPT}(G), \qquad \text{(for all input instance } G)$$

Can we do better than this? What about a 0.999-approximation? This is unfortunately not possible. The result follows from a deep result known as the PCP-theorem (see Arora and Barat complexity theory book [1]). The theorem shows that unless NP=P, optimization problems like the maxcut problem do not have an ϵ-approximation for all $\epsilon < 1$. Hastad in [3] gives an explicit $\epsilon = \frac{16}{17} \approx 0.941$ and shows that any algorithm bettering this ϵ-approximation is not possible. Khot et al. [4] showed that this 0.878-approximation is the best possible algorithm if the unique games conjecture is true.
Are there optimization problems that can be approximated to any ϵ as we want? Yes. The Knapsack problem can be approximated to any ϵ as one wishes. Such problems are said to admit a PTAS.

2 Approximate solution to maxcut problem

Goemans and Williamson [2] gave the 0.878-approximation algorithm. In this talk, we only look at a randomized algorithm whose expected optimal value is at least 0.878 OPT. A randomized algorithm takes as input a graph and a sequence of random bits. Let $\mathcal{A}$ be a randomized approximation algorithm for maxcut. We are interested in showing the following.

$$0.878\ \mathrm{OPT}(G) \leq \mathtt{E}(\mathcal{A}(G)) \leq \mathrm{OPT}(G), \qquad \text{(for all input instance } G)$$

where $\mathtt{E}(\mathcal{A}(G))$ is the expectation taken over all the random choices of $\mathcal{A}$. Let us first recast the maxcut problem.

$$\max\ \frac{1}{4}\sum_{i\in V}\sum_{j\in V} w_{ij}(1 - x_i x_j) \tag{P1}$$
$$\text{subject to the constraints, } x_i \in \{-1, +1\}$$

The problem is highly non-convex (infact, an integer programming problem). For what assignment to x_is do we get the maximum for the objective function? Let S be the maxcut solution. Assign $x_i = 1$ if $i \in S$. Otherwise assign $x_i = -1$. Therefore $(1 - x_i x_j) = 0$ if x_i and x_j are both in S or both not in S. Otherwise, $(1 - x_i x_j) = 2$. The weights in the cut S are added twice, whereas the weights outside the cut are not added. Thus, a maxcut solution produces the optimum solution to the above problem. How about the other direction? Does an optimal solution to the above problem give rise to a maxcut solution? We leave this as an exercise.

We rewrite P1 into an equivalent version.

$$\max\ \frac{1}{4}\sum_{i\in V}\sum_{j\in V} w_{ij}(1 - X_{ij}) \tag{P2}$$
$$\text{subject to the constraints, } X = \mathbf{x}\mathbf{x}^{\mathsf{T}}, \text{ and } X_{ii} = 1 \qquad \text{(for all } i \in V)$$

Here $\mathbf{x}^{\mathsf{T}}$ is the transpose of the vector $\mathbf{x}$ and X is a matrix. Note that the entry at $X_{ij} = x_i x_j$ for all $i, j \in V$. The condition $X_{ii} = 1$ is equivalent to $x_i^2 = 1$ which is equivalent to $x_i \in \{-1, +1\}$. We claim P2 is same as P1. The proof is left as an exercise for the reader.

We now show how to approximate P2. The main idea is as follows. The optimization problem P2 associates a variable $x_i \in \{-1, +1\}$ with a vertex i. We can view the vertices as being assigned a unit vector in $\mathbb{R}^1$ (one dimension). The vectors assigned to the vertices on the opposite sides of the cut point in opposite directions, and the vectors assigned to the vertices on the same side of the cut point in the same direction. The SDP (semidefinite program) relaxation allows the vertices to be assigned a unit vector in $\mathbb{R}^n$ (n dimension). We keep our objectives the same as before. Those on the opposite sides of the cut should be pointing in approximately opposite directions. In contrast, those on the same side of the cut should be pointing in approximately the same direction. Fig. 2 gives a pictorial view of this relaxation. We then extract the cut from a solution to this SDP relaxation.

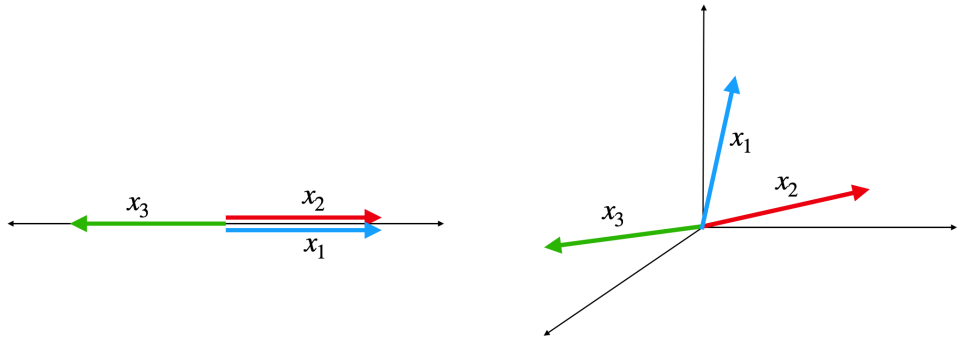


Fig. 2. SDP relaxation: The vertices are mapped to unit vectors in $\mathbb{R}^1$ in the optimization problem P2. In the approximation version, the vertices are mapped to unit vectors in $\mathbb{R}^n$.

The three main steps in this approximation algorithm are

1. Approximate P2 by a SDP optimization problem P3.
2. Decompose the solution to the optimization problem P3.
3. Extract a cut set from this decomposition.

We now provide details of each of these. This is followed by showing that this approximation algorithm runs in polynomial time. The argument that this is indeed a 0.878-approximation is deferred to the next section.

2.1 Approximate Maxcut problem by an SDP

An SDP (semi-definite program) is a special case of a convex optimization problem. The SDP that approximates maxcut is given below.

$$\max \ \frac{1}{4}\sum_{i\in V}\sum_{j\in V} w_{ij}(1-X_{ij}) \tag{P3}$$

subject to the constraints,

$$X = \begin{pmatrix} 1 & x_{11} \cdots x_{1n-1} & x_{1n} \\ x_{11} & 1 \ \cdots \ \cdots & x_{2n} \\ & \cdots\cdots \ \cdots & \\ x_{1n-1} & \cdots\cdots \ 1 & x_{n-1n} \\ x_{1n} & x_{2n} \cdots x_{n-1n} & 1 \end{pmatrix} \text{ is a psd}$$

The matrix X is a symmetric positive semi-definite matrix (psd). Such matrices satisfy the condition that $X = RR^{\mathsf{T}}$ for an $n \times k$ real matrix R and some $k > 0$. Note that X in problem P2 is also a symmetric psd (it is of the form $\mathbf{x}\mathbf{x}^{\mathsf{T}}$). How is P3 different from P2?

1. X in P2 was a rank-1 matrix whereas X in P3 has no rank restriction.
2. The information contained in individual variables x_is (or vertices) are lost in P3. The "hope" is to retrieve the individual vertex information from the matrix X.
3. The x_is in P2 can only take $+1$ or -1 values and hence the matrix X contain only $+1$ or -1 entries. In problem P3 there is no such restriction on the matrix X.

As mentioned above, we have lost some information by this relaxation on the matrix X. What have we gained? There is a polynomial-time algorithm to solve an SDP. In contrast, the problem P2 cannot be answered in polynomial time (unless P=NP).

Theorem 1. *There is an algorithm (we call it SolveSDP) that takes as input an SDP P3 and in polynomial time returns the optimal solution matrix X.*

A caveat: There is a small catch when we say SDP can be solved in polynomial time. The optimal solution X might contain irrational numbers. From an algorithmic point of view, we cannot store or do real number computations. One can work with "rational numbers" close to real numbers. We say that matrices X and Y are δ close if $|X_{ij} - Y_{ij}| \leq \delta$. In other words, all the entries are within δ distance.

Theorem 1. *There is an algorithm, which when given SDP P3 and a $\delta > 0$, returns a matrix X in time polynomial in the input SDP and $\log 1/\delta$ such that X is δ close to the optimal solution. Moreover, X is a symmetric psd.*

Note that the algorithm can get very close to the optimal value. How close is based on the input parameter δ. The smaller the δ, the more the running time of SDP.

2.2 Decomposing the SDP solution matrix X

Our aim is to now extract the cut set from the solution matrix X of P3. In Problem P2, X is of the form $\mathbf{x}\mathbf{x}^\mathsf{T}$. The binary value of x_i decides whether the vertex i is in the maxcut or not. Unfortunately, the matrix X of P3 need not be of this form.

Let us consider the solution matrix X of P3. Since X is symmetric and psd, there exists an $n \times k$ real matrix U such that $X = UU^\mathsf{T}$. There are two ways to decompose the matrix X: (1) using eigen decomposition and (2) using Cholesky decomposition.

Lemma 1 (decomposition). *There is an $O(n^3)$ algorithm which on input a symmetric psd X returns a matrix U such that $X = UU^\mathsf{T}$.*

A proof of the lemma using Cholesky decomposition is given in Section 4. The running time of an eigen decomposition depends on the ratio of the eigen values. The best case being better and the worst case being worse than the Cholesky decomposition.

A caveat: Again, there is a problem when the decomposition matrix U is irrational. This is highly likely since U is like a "square root" matrix. This necessitates an approximate decomposition of a symmetric psd.

Lemma 1 (decomposition). There is a real matrix U such that $X = UU^\mathsf{T}$. Moreover, there is an algorithm, which given X and a $\delta > 0$, returns U in time polynomial in input size of X and $\log 1/\delta$ such that

$$\widehat{X}_{ij} - \delta \leq X_{ij} \leq \widehat{X}_{ij} + \delta$$

where $\widehat{X} = UU^\mathsf{T}$. In short, $\widehat{X}$ is δ close to X.

Since $X_{ii} = 1$ for all i, the diagonal elements of $\widehat{X}$ are close to 1. Therefore

$$1 - \delta \leq \mathbf{u}_i{}^\mathsf{T}\mathbf{u}_i \leq 1 + \delta \qquad \text{(for all row vectors } u_i \text{ of } U\text{)}$$

2.3 Extracting a cut set from the decomposed matrix U

The row vectors of U (denoted as $\mathbf{u}_i$s) contain information for each vertex in the graph. Recap how x_is in Problem P2 (the original maxcut problem) determined whether i is in the maxcut or not. In that setting, vertices i and j fall in the same set (either S or T) if $x_i x_j = 1$. Let us return to the problem P3 and the vectors $\mathbf{u}_i$ and $\mathbf{u}_j$. We have that $X_{ij} = \mathbf{u}_i{}^\mathsf{T}\mathbf{u}_j$. When X_{ij} is high (close to 1), we would want i and j to be in the same set. Since the inner product is proportional to the angle between the vectors, it follows that we want i and j to be in the same set if the angle between $\mathbf{u}_i$ and $\mathbf{u}_j$ is small. There is a problem, however. Consider the following scenario: the angle between vectors $\mathbf{u}_i$ and $\mathbf{u}_j$ is small. Between $\mathbf{u}_j$ and $\mathbf{u}_k$ is small. However, the angle between $\mathbf{u}_i$ and $\mathbf{u}_k$ is not small. We are in a predicament whether to include all the three vectors i, j and k in the same set or not. The angle between the vectors

is not an "equivalence" relation. Therefore, we are interested in a "clustering" algorithm - partition the vectors $\mathbf{u}_i$s into two parts such that angles inside a part are minimized.

Randomization allows clustering the vectors into two parts. We pick a random vector $\mathbf{r}$, and all vectors close to it (less than 90 degree) are included in S, whereas all the other vectors are included in T. Algorithm 1 contains the details. It is easy to observe that the algorithm runs in randomized polynomial time.

Algorithm 1 Cluster the vectors into two parts

```
Initialize set S = ∅.
Pick a random vector r ∈ R^{|V|} such that |r|_2 = 1.
for i ∈ V do
    if r^T u_i ≥ 0 then
        add i to S
    end if
    – i is not added to S if r^T u_i < 0
end for
return S
```

In the above algorithm, the only non-trivial part is picking a random vector $\mathbf{r}$ in the unit sphere's surface. We can do this by picking r_i randomly from the normal distribution $\mathcal{N}(0,1)$. With a high probability, $\mathbf{r}$ is in the unit sphere's surface; the probability increases with the dimension. There are some implementation issues. Theoretically, we pick real numbers from the normal distribution. But practically, we leave out all the real numbers. Moreover, the rational numbers we pick can only be of the form $\frac{p}{q}$ where p and q are polynomial in the size of the input. Thus, an "algorithm" assigns all "un-representable" numbers to probability 0. Does picking from the rest of the numbers (based on normal distribution probability) also give a vector on the unit sphere's surface?

We need to discuss a special case. Note that if $u_i = u_j$ then the vertices behave similarly. The vertex i is put in the same set as vertex j. Therefore, we remove vertex i and vector u_i from the algorithm analysis. The following assumption is important.

$$\textbf{Assume: } \text{for all } i \neq j,\ u_i \neq u_j \text{ and } X_{ij} \neq 1 \tag{3}$$

This concludes the randomized algorithm.

2.4 Summary of the algorithm

The algorithm consists of mainly three steps, each running in polynomial time. (1) Approximation using SDP, (2) Decomposition of the solution to the SDP (3) Clustering the vectors from the decomposition. The following section shows that this algorithm gives in expectation a maxcut which is 0.878-approximate. See the complete algorithm in Algorithm 2.

Algorithm 2 *ApproximateMaxCut*

– *Input: A weighted graph* $G = (V, E)$
– *Output: A* 0.878*-approximation to maxcut*
SDP P = Approximate the maxcut problem into the form P3
– *Solve the SDP* P
X = SolveSDP(P)
– *Decompose* X *into* UU^{T}
U = Decomposition(X)
Let $\{u_1, \ldots, u_{|V|}\}$ be the rows of U.
Let $S = \emptyset$. – *Set* S *will be the cut.*
Pick a random vector $\mathbf{r} \in \mathbb{R}^{|V|}$ such that $|\mathbf{r}|_2 = 1$.
for $i \in V$ **do**
  if $\mathbf{r}^{\mathsf{T}}\mathbf{u}_i \geq 0$ **then**
    – *angle between* $\mathbf{r}$ *and* $\mathbf{u}_i$ *is less than* 90 *degree*
    add i to S
  end if
  – i *is not added to* S *if* $\mathbf{r}^{\mathsf{T}}\mathbf{u}_i < 0$
end for
return S

Some questions:

1. Rather than pick a random vector $\mathbf{r}$, why not pick one of the vectors $\mathbf{u}_i$ randomly, and add all $\mathbf{u}_j$s close (angular-wise) to it into S.
2. Is there any relation between the rank of returned X and the closeness of approximation?
3. Why not find the best 1-rank approximation to the SDP return matrix X and use that as an indicator for recognizing the cut.

3 Establishing the Approximation ratio

Let us call *ApproximateMaxCut* (Algorithm 2) as $\mathcal{A}$. Assume that $\mathcal{A}$ returns S on input G.

For ease of analysis, we assume Algorithms SolveSDP and Decomposition gives the optimal matrix X and the exact decomposition matrix U and not a δ approximation.

Our aim is to show that

$$0.878 \ \mathrm{OPT}(G) \leq \mathtt{E}(\mathcal{W}(S))$$

where the expectation is computed over the random choices of $\mathbf{r}$. Consider the indicator random variables Y_{ij}

$$Y_{ij} = \begin{cases} 0, & \text{if } i \text{ and } j \text{ are either both in } S \text{ or both in } T \\ 1, & \text{otherwise} \end{cases}$$

The expected weight of the cut returned by $\mathcal{A}$ is

$$\begin{aligned}
\mathtt{E}(\mathcal{W}(S)) &= \mathtt{E}(\frac{1}{2}\sum_{i\in V}\sum_{j\in V}\Big(w_{ij}\ Y_{ij}\Big)) \\
&= \frac{1}{2}\sum_{i\in V}\sum_{j\in V}\Big(w_{ij}\ \mathtt{E}(Y_{ij})\Big) && \text{(by linearity of expectation)} \\
&= \frac{1}{2}\sum_{i\in V}\sum_{j\in V}\Big(w_{ij}(1-X_{ij})\ \frac{\mathtt{E}(Y_{ij})}{1-X_{ij}}\Big) && \text{(from Eq. (3), } X_{ij}\neq 1) \\
&= \frac{1}{4}\sum_{i\in V}\sum_{j\in V}\Big(w_{ij}(1-X_{ij})\ \frac{2\ \mathtt{E}(Y_{ij})}{1-X_{ij}}\Big) \\
&\geq \min_{i,j\in V}\Big\{\ \frac{2\ \mathtt{E}(Y_{ij})}{1-X_{ij}}\Big\}\ \frac{1}{4}\sum_{i\in V}\sum_{j\in V} w_{ij}(1-X_{ij}) \\
&= \min_{i,j\in V}\Big\{\frac{2\ \mathtt{Prob}(Y_{ij}=1)}{1-X_{ij}}\Big\}\ \frac{1}{4}\sum_{i\in V}\sum_{j\in V} w_{ij}(1-X_{ij}) \\
& && \text{(Since } Y_{ij} \text{ is an indicator random variable)} \\
&\geq \min_{i,j\in V}\Big\{\frac{2\ \mathtt{Prob}(Y_{ij}=1)}{1-X_{ij}}\Big\}\ \mathrm{OPT}(G)
\end{aligned}$$

The last equality follows from the fact that X is an optimal solution to P3 and $\mathrm{OPT}(G)$ is achieved by an 1-rank matrix $\hat{X}$ which is a feasible solution for the SDP P3. To show our approximation ratio, it is sufficient to bound

$$\min_{i,j\in V}\Big\{\frac{2\ \mathtt{Prob}(Y_{ij}=1)}{1-X_{ij}}\Big\} > 0.878$$

First, we find the probability that i and j are on the opposite sides of the cut. By our algorithm, this happens if and only if the angle between $\mathbf{r}$ and $\mathbf{u}_i$ is acute and the angle between $\mathbf{r}$ and $\mathbf{u}_j$ is obtuse or vice versa.

$$\mathtt{Prob}(Y_{ij}=1) = 2\ \mathtt{Prob}(\mathbf{r}^{\mathsf{T}}\mathbf{u}_i \geq 0 \text{ and } \mathbf{r}^{\mathsf{T}}\mathbf{u}_j < 0)$$

Let θ be the angle between vectors $\mathbf{u}_i$ and $\mathbf{u}_j$. We show the following

$$\mathtt{Prob}(\mathbf{r}^{\mathsf{T}}\mathbf{u}_i \geq 0 \text{ and } \mathbf{r}^{\mathsf{T}}\mathbf{u}_j < 0) = \frac{\theta}{2\pi}$$

This is *rotational symmetric* - the probability remains the same if we rotate all the vectors by the same angle.

$$\mathtt{Prob}(\mathbf{r}^{\mathsf{T}}\mathbf{u}_i \geq 0 \text{ and } \mathbf{r}^{\mathsf{T}}\mathbf{u}_j < 0) = \mathtt{Prob}((Q\mathbf{r})^{\mathsf{T}}Q\mathbf{u}_i \geq 0 \text{ and } (Q\mathbf{r})^{\mathsf{T}}Q\mathbf{u}_j < 0)$$

where Q is any orthonormal matrix - columns are perpendicular to each other and also have unit norm. Therefore $Q^{\mathsf{T}}Q$ is the identity matrix. Rotational matrices are orthonormal

matrices. Let Q be a matrix which rotates $\mathbf{u}_i$ to the x-axis (giving $\hat{\mathbf{u}}_i$) and $\mathbf{u}_j$ to the x-y plane (giving $\hat{\mathbf{u}}_j$) . In other words except for the first co-ordinate in $\hat{\mathbf{u}}_i$ and the first two co-ordinates in $\hat{\mathbf{u}}_j$ all other co-ordinates are zero. That is, $\hat{\mathbf{u}}_i = (u_i^1, 0, 0, \ldots, 0)$ and $\hat{\mathbf{u}}_j = (u_j^1, u_j^2, 0, 0, \ldots, 0)$. Let $\hat{\mathbf{r}}$ be the projection of $\mathbf{r}$ on the x-y plane. Since apart from the first two co-ordinates all the other co-ordinates of $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{u}}_j$ are zeros, $\mathbf{r}^\mathsf{T}\hat{\mathbf{u}}_i = \hat{\mathbf{r}}^\mathsf{T}\hat{\mathbf{u}}_i$ and $\mathbf{r}^\mathsf{T}\hat{\mathbf{u}}_j = \hat{\mathbf{r}}^\mathsf{T}\hat{\mathbf{u}}_j$. So, it is sufficient to analyse the probability in the 2 dimensional x-y plane.

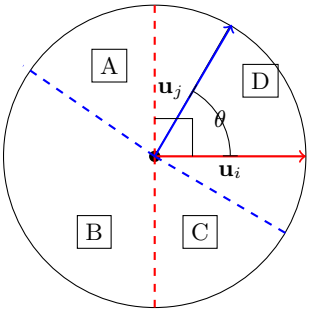
Figure 3 provides a "visual" proof of $\texttt{Prob}(\mathbf{r}^\mathsf{T}\mathbf{u}_i \geq 0 \text{ and } \mathbf{r}^\mathsf{T}\mathbf{u}_j < 0) = \frac{\theta}{2\pi}$. Hence $\texttt{Prob}(Y_{ij} = 1) = \frac{\theta}{\pi}$. Therefore

$$\frac{2\ \texttt{Prob}(Y_{ij} = 1)}{1 - X_{ij}} = \frac{2\ \theta}{\pi(1 - X_{ij})} = \frac{2\ \theta}{\pi(1 - \cos\theta)}$$

The last equality comes from the fact that $X_{ij} = \mathbf{u}_i{}^\mathsf{T}\mathbf{u}_j = \cos\theta$, and because for all $l \in V$, $\mathbf{u}_l{}^\mathsf{T}\mathbf{u}_l = 1$. The paper [2] claims that basic calculus shows

$$\min_\theta \left\{ \frac{2\ \theta}{\pi(1 - \cos\theta)} \right\} > 0.878$$

This concludes the analysis of the algorithm.



1. Region A: $\mathbf{r}^\mathsf{T}\mathbf{u}_i \geq 0$ and $\mathbf{r}^\mathsf{T}\mathbf{u}_j < 0$
2. Region B: $\mathbf{r}^\mathsf{T}\mathbf{u}_i \geq 0$ and $\mathbf{r}^\mathsf{T}\mathbf{u}_j \geq 0$
3. Region C: $\mathbf{r}^\mathsf{T}\mathbf{u}_i < 0$ and $\mathbf{r}^\mathsf{T}\mathbf{u}_j \geq 0$
4. Region D: $\mathbf{r}^\mathsf{T}\mathbf{u}_i < 0$ and $\mathbf{r}^\mathsf{T}\mathbf{u}_j < 0$

Fig. 3. A random vector $\mathbf{r}$ can lie in any of the four regions. Region A (resp. C) cover $\frac{\theta}{2\pi}$ of the area. The probability that $i \in S$ and $j \notin S$ is equal to the area of region A.

If the matrices have irrational entries, we work with matrices δ close to the optimal. Maintaining an 0.878-approximation is possible since we can take a δ as close to the optimal as we want.

4 psd decomposition

This section shows that a symmetric positive semidefinite matrix X can be decomposed as a product of a matrix and its transpose.

We say a matrix is lower triangular (resp. upper triangular) if all entries above (resp. below) the diagonal are zero and the diagonal entries are 1.

Claim. Let X be any full rank matrix. Then $X = LDU$ for a lower triangular matrix L, diagonal matrix D and upper triangular matrix U.
Moreover if X is rational L, D and U are rational and an algorithm can output L, D and U in $O(n^3)$.

Proof (Proof sketch (see Strang [5]).). Consider the full rank matrix X. We can do a row operation and make the first column second row cell to zero. Let this new matrix be Y. You can also go back from Y to X be doing the reverse row operation. Both these row operations (are linear transformations) are equivalent to multiplication by a lower triangular matrix. That is

$$X = \hat{L}Y \qquad \text{(where } \hat{L} \text{ is lower triangular)}$$

By a series of such operations (gaussian elimination) you can get a matrix $\hat{U}$ where all entries below the diagonal are zero (the diagonal entries need not be 1). These series of row operations are equivalent to multiplication by lower triangular matrices. Since product of lower triangular matrices are lower triangular we have

$$X = \hat{L}_1\hat{L}_2 \dots \hat{L}_n\hat{U} = L\hat{U} \qquad \text{(here } L \text{ is lower triangular)}$$

We can factor out the diagonal elements in $\hat{U}$ as follows

$$\begin{aligned}\hat{U} &= \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix} \begin{pmatrix} 1 & a_{12}/d_1 & a_{13}/d_1 & \\ & 1 & a_{23}/d_2 & \\ & & \ddots & \\ & & & 1 \end{pmatrix} \\ &= D\ U \qquad \text{(here } D \text{ is diagonal and } U \text{ is upper triangular)}\end{aligned}$$

Therefore $X = LDU$. By a careful analysis one can establish that this decomposition can be done in $O(n^3)$. □

Claim. Let X be a full rank symmetric matrix. Then $X = LDL^{\mathsf{T}}$ for a diagonal matrix D and lower triangular matrix L. Moreover, if X is rational L and D are rational.

Proof. Since $X = LDU$ by the above theorem and $X = X^{\mathsf{T}}$ we have $X = LDU = U^{\mathsf{T}}DL^{\mathsf{T}}$. Take L to the other side (this is possible since X is full rank). That is $DU(L^{\mathsf{T}})^{-1} = L^{-1}U^{\mathsf{T}}D$. Since one is a left triangular matrix and the other right triangular, the only possibility is it is a diagonal matrix. By looking at the diagonals it follows that $L = U^{\mathsf{T}}$. □

We now show that symmetric psd matrices X have a factorization VV^{T}. We make use of the following property of psd matrices.

Lemma 2. *X is a psd iff the pivots of X are greater than or equal to 0.*

Lemma 3 (Choleksy decomposition)). *Let $\delta > 0$. There is an algorithm that returns a matrix V on input a symmetric psd X in time polynomial in X and $\log 1/\delta$ and such that X is δ close to VV^T.*

Proof. We first show the decomposition for symmetric positive definite matrices (these are full rank matrices). Assume Y is a full rank symmetric psd. Since X is symmetric and full rank we have that $Y = LDL^\mathsf{T}$ from Section 4. From the previous claim, the pivots are non-negative. Therefore, there are real square roots for the diagonal entries of D.

$$Y = LDL^\mathsf{T} = L\sqrt{D}\sqrt{D}L^\mathsf{T} = L\sqrt{D}(L\sqrt{D})^\mathsf{T} = VV^\mathsf{T}$$

Let us now extend the above argument for symmetric positive semidefinite matrices. These matrices need not be of full rank. Consider such an X

$$X = \left(\begin{array}{c|c} Y & A^\mathsf{T} \\ \hline A & N \end{array}\right)$$

where Y is a full rank submatrix and the rows of $[A \;\; N]$ are dependent on the rows of $[Y \;\; A^\mathsf{T}]$. Therefore, there is a matrix Z such that $Z[Y \;\; A^\mathsf{T}] = [A \;\; N]$. In other words

$$ZY = A,\ ZA^\mathsf{T} = N \text{ and therefore } Z = AY^{-1} \text{ and } AY^{-1}A^\mathsf{T} = N \tag{4}$$

Since Y is a symmetric positive definite matrix there exists a V such that $VV^\mathsf{T} = Y$. We can therefore write X as

$$X = \left(\begin{array}{c|c} Y & A^\mathsf{T} \\ \hline A & N \end{array}\right) = \left(\begin{array}{c} V \\ \hline M \end{array}\right)\left(\begin{array}{c|c} V^\mathsf{T} & M^\mathsf{T} \end{array}\right) \tag{5}$$

where M is such that $VM^\mathsf{T} = A^\mathsf{T}$ or $M^\mathsf{T} = V^{-1}A^\mathsf{T}$ (this is possible since Y is full rank and therefore V is also full rank). We need to verify that $MM^\mathsf{T} = N$.

$$MM^\mathsf{T} = A(V^{-1})^\mathsf{T}V^{-1}A^\mathsf{T} = AY^{-1}A^\mathsf{T} = N$$

Where the last equality follows from Eq. (4). From Eq. (5) it follows that $X = \hat{V}\hat{V}^\mathsf{T}$ for a matrix $\hat{V}$. This concludes the proof of the fact that a psd can be decomposed into the product of a matrix and its transpose.

Finally, we have to argue the algorithmic complexity. We only used gaussian elimination and inverses of matrices. A detailed analysis will show an $O(n^3)$ running time. Moreover, the only place where an approximation is required is while finding the square root of the diagonal entries of matrix D. Newton's method can approximate this to an accuracy as claimed in the statement of the theorem. □

References

1. Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
2. Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.
3. J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.
4. Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM J. Comput.*, 37(1):319–357, April 2007.
5. Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 2016.