Semidefinite Programming applied to Maximum Cut

Fabrizio Franco
School of Computer Science
University of Engineering and Technology UTEC
Email: fabrizio.franco@utec.edu.pe

Carlos Reátegui
School of Computer Science
University of Engineering and Technology UTEC
Email: carlos.reategui@utec.edu.pe

Christian Ledgard
School of Computer Science
University of Engineering and Technology UTEC
Email: christian.ledgard@utec.edu.pe

Maor Roizman Gheiler
School of Computer Science
University of Engineering and Technology UTEC
Email: maor.roizman@utec.edu.pe

Stephano Wurttele
School of Computer Science
University of Engineering and Technology UTEC
Email: stephano.wurttele@utec.edu.pe

***Abstract*—This article explains an approximation of the *Maximum Cut Problem* via *Semidefinite Programming (SDP)*. Furthermore, it analyzes in an independent way both concepts to explore their relation with other topics. For example, the relation between *Semidefinite Programming* and *Linear Programing*. Finally, the core of the research is the theoretical analysis of the *Max-Cut Problem* using *SDP* and the complexity of the application of this method.**

***Index Terms*—Semidefinite Programming, Maximum Cut, Optimization problems, Constraints.**

I. Introduction

Semidefinite programming (SDP) is a process in which a linear function is minimized to a combination of symmetric matrices that are positive semidefinite. These constraints are convex, making it one of many methods to solve convex optimization problems. [1] On itself, this derives to more specific types of problems such as are Linear or Quadratic Problems, depending on additional constraints that can be set in the procedure, that may close up on a form of a particular group of answers for a specific situation. For an easier connection between topics, we can first define what a semidefinite matrix is and then state SDP's similarities with Linear Programming as studied.

We say a symmetric matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite if $z^T M z \geq 0$ for all $z \in \mathbb{R}^n$ [1], or all its eigenvalues are non-negative. Making a sort of comparison with linear programming, we also have a matrix product for a particular maximization problem given by the primal, that is $max(cX)$ with some constraints on the X matrix of values AX where **all values of X are positive semidefinite matrices** with the properties mentioned before, thus the name and the reason why this solves convex problems. Thus, while Linear Programming works with inequalities using vectors, in SDP positive semidefinite matrices are used. With this intuition, we can narrow down a DSP situation to a LP for diagonal values of X, as in that case our matrix dot product would only consider those vectors that are considered for a Linear Programming problem.

Another point worth noting is the notion of duality for SDP, covered by Linear Programming with the Simplex Procedure, is a little more complex and leads to weaker results [1]. For SDP, a dual resembles the one in LP as we have $b^T y$ where $\sum y_i A_i + S = c$, such that y is a vector, the slack variables are now a matrix that **must also be positive semidefinite**, and where c is the original optimization problem we want to solve.

II. Related Works

Being SDP a known method for linear equations, many sources try to narrow down an explanation either geometrically or mathematically to show the proceedings of the algorithm. [2] is an MIT lecture which focuses on explaining the basics for this work, from its mathematical principles related to linear algebra and matrices, to their applications on Linear Programming and Semi-definite Programming. It also shows many examples of the algorithm on optimization problems, and briefly mentions ellipsoidal problems and its relation to the solution. After this introduction, the focus can start to change to the solution methods, where we stumble upon the ellipsoid method, discussed in multiple sources (i.e. [3], [4]) which carries on different analysis by certain methods on the ellipsoid formed by DSP, explaining it mostly graphically or programatically and calculating its average performance according to the algorithms.

Directly relating to the specific problem at hand, we have [5], which due to its nature as a lecture document focuses on giving the general intuition concerning the problem, and how a function is established with its own concerns, also reminding

us important properties of positive semi-definite matrices and how these lead to specific artifices to perform relaxations on the problem. A more developed work that has practical tests is [6], which gives a review on basic definitions on the Max-Cut problem and shows how the relaxations affect results for the algorithms, using epsilon values to represent the difference between Optimal and Exact solutions for the problem. This is also the only source referred which performs Property Testing Algorithms to test general complexity bounds and properties for the problem.

III. Maximum Cut

To understand the *Max-Cut Problem*, we must understand what a graph cut is.

A. Graph cut

A cut of a graph $G = (V, E)$ is a bi-partition of V, given by $S \subseteq V$. The cut is the pair $(S, V \setminus S)$. We say that an edge $(u, v) \in E$ is cut if $u \in S$ and $v \in V \setminus S$, or $u \in V \setminus S$ and $v \in S$. The size of a graph cut is the number of edges cut [5].

B. Max-Cut Problem

The Max-Cut problem is to find the cut that maximizes the weights of the edges in the graph G, going between the two partitions. This process is equivalent to finding a bipartite subgraph in G with as many edges as possible [5]. This problem can be applied to both directed and undirected graphs.

C. Max-Cut as an NP-Hard problem

It is important to mention that *Maximum Cut Problem* has a NP-Hard complexity. This means that any problem in NP can be reduced in polynomial time to *Max-Cut*, and it's at least as hard as a NP problem. To prove that a problem is NP-hard, we can reduce a problem already proven to the other. For example, using 3SAT NP-hard problem we can prove it. The main decision in the Max-Cut problem is: 'For a given graph G and given parameter k, does there exist a cut of size $\geq$ in G?' [7].

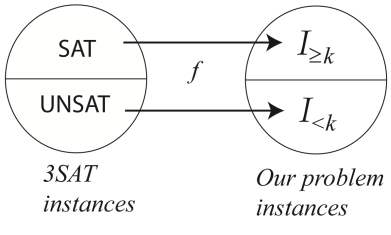

Fig. 1. Reduction from *SAT Problem* to *Max-Cut Problem* recovered from [7]

-In the figure 1 the set $I_{\geq k}$ contain all graphs with a *Max-Cut* of k edges or more. Only if the graph $G \in I_{\geq k}$ we can use the reduction of the problem as an instance of 3SAT. This relation shows that deciding *The Max-Cut Problem* is at least as hard as deciding satisfiability in 3SAT. From this, it can be said that it is also an *NP-hard Problem*.

IV. Positive Semidefinite Matrices Constraints

1) $x^T Ax \geq$ for all x $\neq 0$.
2) All eigenvalues of A satisfy $\lambda_j \geq 0$.
3) All NW (upper left) minors of A are positive.
4) All pivots in elimination ≥ 0.
5) $A = R^T R$ (Cholesky Decomposition).

We take these matrices conditions from the 'Theorem 1' of [8].

V. Applications

As the research proposes both the topic of semidefinite programming and its application to the maximum cut, the applications of these topics will be evaluated separately to enrich the research.

1) Optimization problems involving linear matrix inequality (LMI) constraints [9]: The goal is maximize the determinant of a matrix subject to LMI constraints.

$$\begin{aligned} &\text{maximize det } G(x) \\ &\text{subject to } G(x) = G_0 + x_1 G_1 + \cdots + x_m G_m \succ 0 \\ &\qquad\qquad F(x) = F_0 + x_1 F_1 + \cdots + x_m F_m \succeq 0. \end{aligned} \tag{1}$$

We can find this kind of optimization problems in computational geometry, information theory and statistics [9].

2) Ellipsoidal approximation [9]: In this type of approximations we can find minimization and maximization problems. Both related to the volume of the ellipsoid and a set C.

1. Minimization problem: *minimum* volume ellipsoid around a given set C.

$$\begin{aligned} &\text{minimize log det } A^{-1} \\ &\text{subject to } A = A^T \succ 0 \\ &\qquad\qquad \|A_v + b\| \leq 1, \forall v \in C \end{aligned} \tag{2}$$

2. Maximization problem: *maximum* volume ellipsoid contained in a given convex set C.

$$\begin{aligned} &\text{maximize log det } B \\ &\text{subject to } B = B^T \succ 0 \\ &\qquad\qquad B_y + d \in C, \forall y, \|y\| \leq 1 \end{aligned} \tag{3}$$

3) Wire and transistor sizing [9]:

$$C\frac{dv}{dt} = -G(v(t) - u(t)) \tag{4}$$

$v(t)$ is the vector of voltages, $u(t)$ is the vector of voltage sources, C is the capacitance matrix and G is the conductance matrix. This kind of mathematical models are used to the approximation of transistors and interconnect wires in large-scale integration circuits.

Using *Semidefinite Programming* we can obtain the minimum amount of transistors and cables that we can use to satisfy the model present with the differential equation.

4) Assessing the Metabolic Potential [10]: As previously stated, the Max-Cut is equivalent to finding the maximum bipartite subgraph within a graph.
The representation of metabolic networks as bipartite graphs it's used for the study of metabolic potential in the context of a metabolic system and in terms of the metabolites that the system can produce in a specific period of time.

VI. Linear Programming vs. Semidefinite Programming

"Semidefinite programming is an extension of linear programming where the component-wise inequalities between vectors are replaced by matrix inequalities" [11].

Linear Programming	*Semidefinite Programming*
$\textbf{max}\ c \cdot x$	$\textbf{max}\ c \bullet x$
$\textbf{s.t.}\ a_j \cdot x \leq b_j \quad (5)$	$\textbf{s.t.}\ A_j \bullet X \leq b_j \quad (6)$
$x \geq 0$	$x \succeq 0$
Can be solved **exactly** in polynomial time.	Can be solved **almost** exactly in polynomial time.

As we can see in 5 and 6, we move from a *Linear Programming* problem to a *Semidefinite Programming* optimization problem when we used matrices of nxn in place of linear vectors. It's important to consider that most of interior-point methods, that are a class of algorithms that solve linear and nonlinear convex optimizations problems, have been generalize to *Semidefinite Programming*. As we saw in *Linear Programming*, the worst case has a polynomial complexity [11].

VII. Approximation of the Max-Cut problem using SDP

To approximate the *Max-Cut Problem* using *SDP*, we will construct a *Quadratic Integer Program* (QIP) for this problem, apply a vector relaxation to it and finally formulate it using SDP [5] [12].

But first, let's use some notation to define the problem. Let $G = (V, E)$ be an undirected unweighted graph, $V = \{1, \ldots, n\}$ and $(i, j) \in E$ be an egde of G. We cut G by $S \subseteq V$, getting the pair $(S, V \setminus S)$.

A. Quadratic Integer Program

Let y_i be a variable which value is +1 if the vertex $i \in S$, and -1 otherwise.

So, y_i can be defined as follows:

$$y_i = \begin{cases} +1, & \text{if } i \in \text{S} \\ -1, & \text{if } i \notin \text{S} \end{cases} \tag{7}$$

Note that if (i, j) is cut, then $y_i y_j = -1$. Recall that the *Max-Cut Problem* consists in maximizing the number of edges cut, so we want an expression that yields 1 when $y_i y_j = -1$, and 0 otherwise. The expression $\frac{1}{2} - \frac{1}{2} y_i y_j$ satisfies this condition.

The only constraint of the program is that $y_i \in \{-1, +1\}\ \forall i \in V$.

So we can formulate the QIP as follows, factoring $\frac{1}{2}$ out [12]:

$$\begin{aligned} &\text{maximize } \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - y_i y_j) \\ &\text{subject to } y_i \in \{-1, +1\}\ \forall i \in V. \end{aligned} \tag{8}$$

Nonetheless, integer programs like QIPs are nearly untractable [5], so we must consider a relaxation on the formulation 8.

B. Quadratic Integer Program Relaxation

According to Goemans and Williamson (1995), the previous QIP formulation can be interpreted as restricting y_i to be a 1-dimensional vector of unit norm, and a relaxation can be defined by allowing y_i to be a multidimensional unit vector $v_i \in \mathbb{R}^n$. We need an objective function that reduces to $\frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - y_i y_j)$. Such function can be obtained by defining the objective function of the relaxation as $\frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - v_i \cdot v_j)$, which yields a number $\in \mathbb{R}$ [12].

Also, note that $v_i \in S_n$, where S_n is the *n*-dimensional unit sphere, because $\|v_i\| = 1$ [12].

So, the relaxed QIP can be formulated as:

$$\begin{aligned} &\text{maximize } \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - v_i \cdot v_j) \\ &\text{subject to } v_i \in S_n\ \forall \in V. \end{aligned} \tag{9}$$

C. SDP approach

Let's go a few steps back, to the QIP formulation, and note that:

$$\begin{aligned} \sum_{(i,j) \in E} w_{ij}(\frac{1}{2} - \frac{1}{2} y_i y_j) &= \sum_{(i,j) \in E} w_{ij} \frac{1}{4}(y_i^2 + y_j^2) - \frac{1}{4}(2 y_i y_j) \\ &= \sum_{(i,j) \in E} w_{ij} \frac{1}{4}(y_i^2 - 2 y_i y_j + y_j^2) \\ &= \sum_{(i,j) \in E} w_{ij} \frac{1}{4}(y_i - y_j)^2 \\ &= \frac{1}{4} \sum_{(i,j) \in E} w_{ij}(y_i - y_j)^2 \end{aligned} \tag{10}$$

And after the relaxation, the objective function can be defined as $\frac{1}{4}\sum_{(i,j)\in E} w_{ij}\|v_i - v_j\|^2$. Note that we take the norm because we want the objective function to reduce to $\frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - y_i y_j)$; i.e., we don't care about directions, only magnitudes [5].

Now, we will formulate the SDP problem applying some linear algebra to $\frac{1}{4}\sum_{(i,j)\in E} w_{ij}\|v_i - v_j\|^2$.

First, let X be a matrix $\in \mathbb{R}^{n\times n}$, such that $X_{ij} = v_i^T v_j$ [5]. Note that X satisfies condition 5) from section IV; thus $X \succeq 0$, which means that X is positive semidefinite.

Recall that $\|v_i\|^2 = 1$, thus $X_{ii} = 1\ \forall i \in V$. After applying some linear algebra, we get:

$$\begin{aligned}\frac{1}{4}\sum_{(i,j)\in E} w_{ij}\|v_i - v_j\|^2 &= \frac{1}{4}\sum_{(i,j)\in E} w_{ij}(v_i^T v_i - 2v_i^T v_j + v_j^T v_j)\\ &= \frac{1}{4}\sum_{(i,j)\in E} w_{ij}(X_{ii} - 2X_{ij} + X_{jj})\\ &= \frac{1}{4}\sum_{(i,j)\in E} w_{ij}(2 - 2X_{ij})\\ &= \frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - X_{ij})\end{aligned} \tag{11}$$

So, we can finally formulate the SDP as [5]:

$$\begin{aligned}&\text{maximize } \frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - X_{ij})\\ &\text{subject to } X_{ii} = 1\\ &\qquad\qquad X \succeq 0\end{aligned} \tag{12}$$

VIII. Asymptotic Analysis

It has been proven that given a system of m linear inequalities over the cone of SDP matrices of order n, they can be tested in $m \cdot n^{O(min\{m,n^2\})}$ arithmetics operations. [13]

At this point, we may state the problem to be solved as:

Given the set of matrices with order n labeled as A_i and integers values b_i, it is necessary to determine if exists a real matrix M with order n that satisfies the following condition:

$$A_i \cdot M \leq b_i, \quad i = 1...m, M \succeq 0$$

That problem has a polynomial equivalent problem where given the set of matrices B_i, exists a set of real numbers X each one multiplying a different matrix with the following condition:

$$B_0 + x_1 B_1 + \cdots + x_m B_m \succeq 0$$

Both problems are included in the NP class. Nevertheless, the SDP of MAX-CUT can be solved in a polynomial time of $O((m + n^2)\cdot n^5 lg(n \cdot R))$ because the following analysis [14]:

- P_n is the space of the set of matrices with order $n \times n$, each one can be treated as a vector in $\mathbb{R}^{\frac{n\cdot(n+1)}{2}}$
- Given a positive number R, C_R is the compact set $C \cap \{M | tr(M) \leq R\ \wedge M \in P_n \wedge M \geq 0\}$
- The discrepancy of a Matrix can be defined as "The minimum number d for which the vertices can be 2-coloured red and blue so that in each of the given sets, the difference between the numbers of red and blue vertices is at most d."
- The discrepancy of $\theta^* = min\{\theta | A_i \cdot M \leq b_i, i = 1...m, M \in C_R\}$. Note this is using the previous mentioned expression.
- Compute the optimal value of θ^* of program is a convex problem that can solved used ellipsoid method (Method for SDP solution). That method requires $O(n^4 log(2^l \cdot \frac{nR}{\epsilon}))$ iterations, l represents the maximum binary size of the original input coefficients.
- In the method mentioned above each iteration/step requires $O(n^2(m+n))$ arithmetic operations.
- The Ellipsoid method will be analyzed below because it solves the satisfiability version of SDP.

The ellipsoid method ensures a polynomial solution in all cases but the time complexity is not good. That is the trade-off given by the constraints of working on NP approximation solutions. The procedure of Ellipsoid method is as follows [15]:

1) If a solution set exists, it has a positive volume. Iterations are applied to the relaxed equations of the original matrix.
2) Bound solution set into a ellipsoid quite bigger than the solution set. This ellipsoid contains all solutions.
3) Test if the center of the ellipsoid is covered by the geometric representation of the solution set.
   - If it is, then the center is a solution for the system (satisfiability) and terminate the algorithm.
   - Else, add a separating hyperplane and cut the ellipsoid in half. The solutions contained in the half-ellipsoid will be contained in a new ellipsoid of smaller volume. (The partitions are made by a separation oracle, an algorithm that given $x \notin C$ separate them by an hyper-plane)
     - If the new ellipsoid is too small to contain the solution set, terminates the procedure and there is no solutions to that relaxed problem.
     - Else, go back to the Step 3.

The numbers of iterations and cost per iteration has been mentioned above. Thus, after the explanation of the algorithm, it is quite clear that the iterations are done by the Step 3 and the cost per iteration is given by the conditional statements inside Step 3. Note that this conditions do not have a time complexity of O(1) because they have a geometric procedure of verification to be carried out. In conclusion, the time complexity of the ellipsoid method can be stated as:

$$O((m+n^2)\cdot n^5 lg(n\cdot R)$$

m=Number of equations, R= Numerical size of coefficients

IX. Goemans-Williamson Randomized Rounding

We already know how to relax Max-Cut into a SDP, and that SDPs can be solved. Nonetheless, we still need to convert an SDP solution back into a solution for Max-Cut [16]. Goemans and Williamson proposed a way to perform this conversion using randomized rounding.

Recall that $v_i \in S_n$, where S_n is the n-dimensional unit sphere, because $\|v_i\| = 1$ [12]. We can cut v with a random hyperplane through the origin, such that everything on one side of the hyperplane is a partition, getting the bipartition given by the pair $(S, V \setminus S)$ [16].

Then, we choose a random normal vector r of the hyperplane. The probability that two vectors v_i and v_j are separated by a random hyperplane is [17]:

$$\begin{aligned}\mathbf{Pr}[(v_i\cdot r)(v_j\cdot r)<0] &= \frac{\theta}{\pi}\\ \mathbf{Pr}[(i,j)\in E \text{ is an edge cut}] &= \frac{\theta}{\pi}\end{aligned} \tag{13}$$

where θ is the angle formed by v_i and v_j.

Since v_i and v_j are unit vectors, $v_i \cdot v_j = \cos\theta$, then $\theta = \cos^{-1}(v_i \cdot v_j)$.

Then, according to [16],

$$\mathbf{E}[\text{cut value}] = \sum_{(i,j)\in E} w_{ij}\frac{\cos^{-1}(v_i\cdot v_j)}{\pi}$$

and recall that the SDP formulation is as follows:

$$SDPOpt: \sum_{(i,j)\in E} w_{ij}\frac{1-v_i\cdot v_j}{2}$$

Since $SDPOpt$ is an upper bound on $\mathbf{E}$[cut value], to conclude that $\mathbf{E}$[cut value] $\geq \alpha SDPOpt$, we must find α, which is the approximation ratio of the algorithm [12].

Recall that $\theta = \cos^{-1}(v_i \cdot v_j)$, so

$$\mathbf{E}[\text{cut value}] = \sum_{(i,j)\in E} w_{ij}\frac{\theta}{\pi}$$

and

$$SDPOpt: \sum_{(i,j)\in E} w_{ij}\frac{1-\cos\theta}{2}$$

So, to find α, we calculate:

$$\begin{aligned}\frac{\mathbf{E}[\text{cut value}]}{SDPOpt} &\geq \min_{0\leq\theta\leq\pi}\{\frac{\theta}{\pi}\div\frac{1-\cos\theta}{2}\}\\ &= \min_{0\leq\theta\leq\pi}\{\frac{2}{\pi}\frac{\theta}{1-\cos\theta}\}\\ &= 0.87856\end{aligned} \tag{14}$$

Figure 2 shows a plot of the function, with the point A being the minimum point (note that $y = 0.878567$).

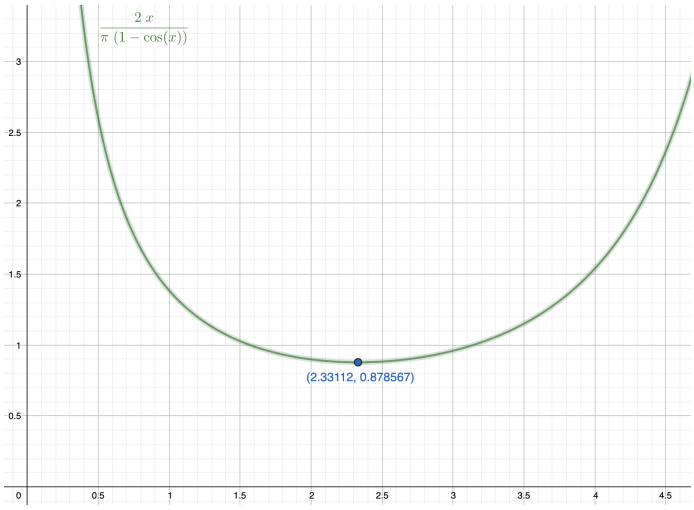

Fig. 2. Geogebra plotting of $\frac{2}{\pi}\frac{\theta}{1-\cos\theta}$

And figure 3 shows graphically how the $SDPOpt$ is an upper bound on $\mathbf{E}$[cut value].

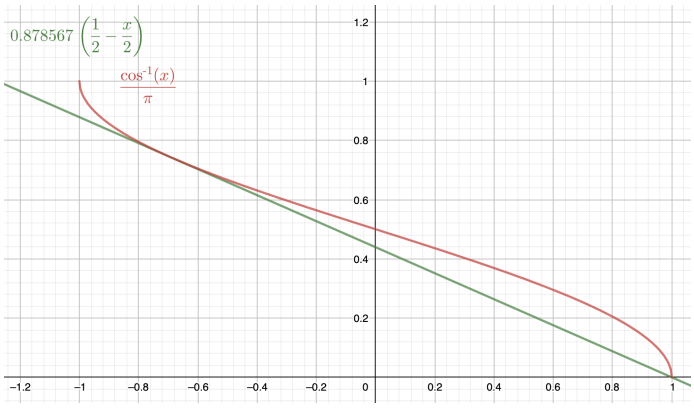

Fig. 3. 'Graphical proof' of $\mathbf{E}$[cut value] $\geq \alpha SDPOpt$

Thus, we can conclude that [16]:

$$\mathbf{E}[\text{Goemans Williamson cut}] \geq 0.87856 SDPOpt$$

X. Conclusions

As it can be reduced to 3SAT, Maximum Cut is a NP-hard problem which can not be solved in polynomial time. To reduce its complexity, certain approximation algorithms must

be performed to get almost optimal answers in a reasonable amount of time. For that matter, relaxation is performed to an extent in which we can get a SDP problem with specific constraints that make it suitable for solution, using various methods as is the Ellipsoid Method. This problem can be geometrically interpreted as a convex problem delimited by hyperplanes, formed by the constraints and the Ellipsoid algorithm itself, leading to an area that contains the answer, or showing there is no answer at all. This solution on itself is an approximation, and it is used to answer the original problem by using the Goemans-Williamson randomized rounding, which leads to a .87856-factor approximation for the real solution for Max-Cut [16] [2].

Furthermore, we showed the intuition behind the Ellipsoid algorithm and how it works, to understand how it solves SDPs and its time complexity, which is $O((m + n^2) \cdot n^5 lg(n \cdot R)$. Clearly, its polynomial degree is high, but this is the tradeoff of approximating NP-hard problems in polynomial time.

Overall, we showed how to solve a SDP of the Max-Cut problem, formulating it first as a *Quadratic Integer Program (QIP)*, applying a vector relaxation to it and some linear algebra to meet the SDPs constraints; then we mentioned how we can solve SDPs using the Ellipsoid algorithm, and the time taken by this procedure; finally, we analyzed how to convert the SDP solution back to a solution to a Max-Cut problem using randomized rounding, and also the approximation ratio of the algorithm, which in this case is 0.878567.

REFERENCES

[1] L. Vandenberghe and S. Boyd, "Semidefinite programming," *Mathematical Programming, State of the Art*, vol. 38, no. 1, p. 276–308, 1994. [Online]. Available: https://web.stanford.edu/~boyd/papers/pdf/semidef_prog.pdf

[2] R. M. Freund, "Introduction to semidefinite programming (sdp)," 2009.

[3] F. M. de Oliveira Filho, "Lecture notes on semidefinite programming - the ellipsoid method," March 2014.

[4] D. Stehlé and S. Thomassé, "Polynomiality of linear programming," pp. 159–167, 2012. [Online]. Available: http://www.ens-lyon.fr/DI/wp-content/uploads/2012/01/ellipsoid.pdf

[5] S. Sachdeva, "Max-cut and semidefinite programming," February 2015.

[6] R. O'Donnell and Y. Wu, "An optimal sdp algorithm for max-cut, and equally optimal long code tests," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, ser. STOC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 335–344. [Online]. Available: https://doi.org/10.1145/1374376.1374425

[7] H. Chan, "Approximations algorithms," 2014. [Online]. Available: https://www.cs.cmu.edu/afs/cs/academic/class/15854-f05/www/scribe/lec02.ps

[8] A. Eremenko, "Positive definite and semidefinite forms," April 2020.

[9] S. Boyd and L. Vandenberghe, "Applications of semidefinite programming," 1998. [Online]. Available: https://www.seas.ucla.edu/~vandenbe/publications/sdp-apps.pdf

[10] M. Rocha and P. G. Ferreira, "Chapter 14 - graphs and biological networks," in *Bioinformatics Algorithms*, M. Rocha and P. G. Ferreira, Eds. Academic Press, 2018, pp. 289 – 311. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128125205000146

[11] A. Bockmayr, V. Weispfenning, and M. Maher, "Chapter 12 - solving numerical constraints," in *Handbook of Automated Reasoning*, ser. Handbook of Automated Reasoning, A. Robinson and A. Voronkov, Eds. Amsterdam: North-Holland, 2001, pp. 751 – 842. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978044450813350014X

[12] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the Association for Computing Machinery*, vol. 42, no. 6, November 1995.

[13] L. Porkolab and L. Khachiyan, "On the complexity of semi-definite programs," 2016. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=8C63BE85EB3586640D5FC6BCE2F47D53?doi=10.1.1.50.425&rep=rep1&type=pdf

[14] L. Lovasz, J. Spencer, and K. Vesztergombi, "Discrepancy of set-systems and matrices," *Eur. J. Comb.*, vol. 7, no. 2, p. 151–160, Apr. 1986. [Online]. Available: https://doi.org/10.1016/S0195-6698(86)80041-5

[15] U. of San Diego. The ellipsoid algorithm. Coursera. [Online]. Available: https=//bit.ly/3niP8wF

[16] A. Gupta and R. O'Donnell, "Lecture 10: Semidefinite programs and the max-cut problem," 2011. [Online]. Available: http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture10.pdf

[17] H. Kaplan and U. Zwick, "Max cut using semidefinite programming," 2016. [Online]. Available: https://slideplayer.com/slide/14560123/