# Semidefinite Programming applied to Maximum Cut

## Asymptotic analysis

Franco, Ledgard, Reátegui, Roizman & Wurttele

January 20, 2021

# Introduction

## Semi-definite programming (SDP)

Process in which **a linear function is minimized** to a combination of symmetric matrices that are positive semi-definite. [VB94]

## Semi-definite Matrix

We say a symmetric matrix $M \in \mathbb{R}^{n \times n}$ is positive semi-definite if $z^T M z \geq 0$ for all $z \in \mathbb{R}^n$ [VB94], or all its eigenvalues are non-negative.

# Maximum Cut

## Graph Cut Definition

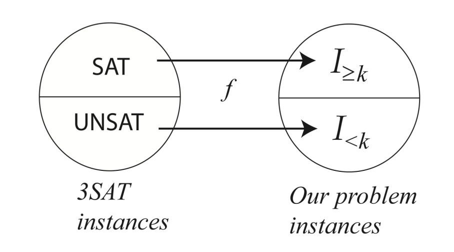
A cut of a graph $G = (V, E)$ is a bi-partition of V, given by $S \subseteq V$. The cut is the pair $(S, V \setminus S)$. We say that an edge $(u, v) \in E$ is cut if $u \in S$ and $v \in V \setminus S$, or $u \in V \setminus S$ and $v \in S$. The size of a graph cut is the number of edges cut [Sac15].

## Max-Cut problem

The maxcut problem consists in finding the maximum cut of a graph $G$, which is equivalent to finding a maximum bipartite graph in $G$ [Sac15].

## Max-Cut as an NP-Hard problem

The *Maximum Cut Problem* is NP-Hard. This means that any problem in NP can be reduced in polynomial time to *Max-Cut*, and it's at least as hard as any NP problem.



# Applications

## Optimization Problems

Problems involving **linear matrix inequality** (LMI) constraints [BV98]: *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 [BV98].

## Structural Optimization

'We cosider a truss structure with m bars connecting a set of n nodes. External forces are applied at each node, which cause a displacement in the node positions' [BV98].

## Optimization problem

$$\begin{aligned} W_{tot}(x) &= w_1 x_1 + \ldots + w_m x_m \\ W_{tot}(x) &\leq W: \text{ is a given limit on truss weight.} \end{aligned} \tag{2}$$

**Goal:** Design the stiffest truss, subject to bounds on the bar cross-sectional areas and total truss weight.

## Wire and transistor sizing

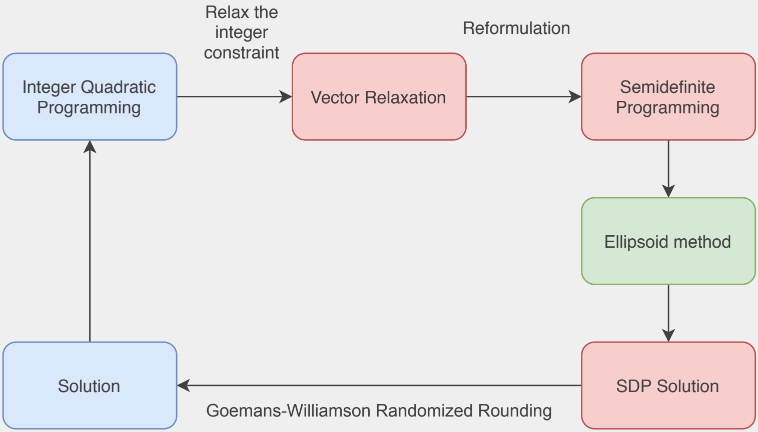
Approximation of transistors and interconnect wires in large-scale integration circuits. Using *Semidefinite Programming* we can obtain the minimum amout of transistors and cables that we can use to satisfy the model present with the differential equation.

$$C\frac{dv}{dt} = -G(v(t) - u(t)) \tag{3}$$

### Assessing the Metabolic Potential [RF18]

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.

# Max-cut Schematic



# Linear Programming vs Semi-definite Programming

*Linear Programming*

$$\begin{aligned} \textbf{max}\ & c \cdot x \\ \textbf{s.t.}\ & a_j \cdot x \le b_j \\ & x \ge 0 \end{aligned} \tag{4}$$

Can be solved **exactly** in polynomial time.

*Semidefinite Programming*

$$\begin{aligned} \textbf{max}\ & c \bullet x \\ \textbf{s.t.}\ & A_j \bullet X \le b_j \\ & x \succeq 0 \end{aligned} \tag{5}$$

Can be solved **almost** exactly in polynomial time.

# Approximation of the Max-Cut problem using SDP

# Approximation of the Max-Cut problem using SDP

## Definition

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 $(S, V \setminus S)$.

## QIP $\rightarrow$ Relaxation $\rightarrow$ 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.

# Approximation of the Max-Cut problem using SDP

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

## $y_i$ definition

$$y_i = \begin{cases} +1, & \text{if } i \in \text{S} \\ -1, & \text{if } i \notin \text{S} \end{cases} \tag{6}$$

We want an expression that yields 1 when $y_i y_j = -1$ (i.e, when the edge cuts), and 0 otherwise. An expression that meets this condition is $\frac{1}{2} - \frac{1}{2} y_i y_j = \frac{1}{2}(1 - y_i y_j)$.

# Approximation of the Max-Cut problem using SDP

From the previous expression, $\frac{1}{2}(1 - y_i y_j)$, we can formulate the problem as a Quadratic Integer Program (QIP) as follows:

## QIP

$$\begin{aligned} &\textbf{maximize } \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - y_i y_j) \\ &\textbf{subject to } y_i \in \{-1, +1\} \ \forall i \in V. \\ &(\text{i.e., } y_i^2 = 1) \end{aligned} \tag{7}$$

# Approximation of the Max-Cut problem using SDP

## Relaxation

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 $v_i \in \mathbb{R}^n$.

## Relaxed problem

$$\begin{aligned} \textbf{maximize } & \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - v_i \cdot v_j) \\ \textbf{subject to } & v_i \in S_n \ \forall \in V. \\ & (\text{i.e, } v_i \cdot v_i = 1) \end{aligned} \tag{8}$$

# Approximation of the Max-Cut problem using SDP

## Intuition to formulate the SDP (recall the previous QIP)

$$\begin{aligned}
\sum_{(i,j)\in E} w_{ij}\left(\frac{1}{2}-\frac{1}{2}y_iy_j\right) &= \sum_{(i,j)\in E} w_{ij}\frac{1}{4}(y_i^2+y_j^2)-\frac{1}{4}(2y_iy_j) \\
&= \sum_{(i,j)\in E} w_{ij}\frac{1}{4}(y_i^2-2y_iy_j+y_j^2) \\
&= \sum_{(i,j)\in E} w_{ij}\frac{1}{4}(y_i-y_j)^2 \\
&= \frac{1}{4}\sum_{(i,j)\in E} w_{ij}(y_i-y_j)^2
\end{aligned} \tag{9}$$

# Approximation of the Max-Cut problem using SDP

Until now, we have the QIP

$$\frac{1}{4}\sum_{(i,j)\in E} w_{ij}(y_i - y_j)^2$$

and the vector relaxation

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

Note that since we care only about magnitudes, and not directions

$$\frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - v_i \cdot v_j) = \frac{1}{4}\sum_{(i,j)\in E} w_{ij}\|v_i - v_j\|^2$$

# Approximation of the Max-Cut problem using SDP

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

Let $X$ be a matrix $\in \mathbb{R}^{n\times n}$, such that $X_{ij} = v_i^T v_j$, **then** $X = v^T v$ [Sac15]. Note that $X \succeq 0$.

Recall that $\|v_i\|^2 = 1$, thus $X_{ii} = 1\ \forall i \in V$.

# Approximation of the Max-Cut problem using SDP

After applying some linear algebra to the relaxation, we get:

$$\begin{aligned}
\frac{1}{4}\sum_{(i,j)\in E} w_{ij}\|v_i - v_j\|^2 &= \frac{1}{4}\sum_{(i,j)\in E} w_{ij}(v_i^T v_i - 2v_i^T v_j + v_j^T v_j) \\
&= \frac{1}{4}\sum_{(i,j)\in E} w_{ij}(X_{ii} - 2X_{ij} + X_{jj}) \\
&= \frac{1}{4}\sum_{(i,j)\in E} w_{ij}(2 - 2X_{ij}) \\
&= \frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - X_{ij})
\end{aligned} \tag{10}$$

# Approximation of the Max-Cut problem using SDP

So we can finally formulate the SDP [GW95]:

## SDP for Max-Cut

$$\begin{aligned} \text{maximize } & \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - X_{ij}) \\ \text{subject to } & X_{ii} = 1 \\ & X \succeq 0 \end{aligned} \tag{11}$$

# Asymptotic Analysis

# Asymptotic Analysis

## Complexity

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.

## Constraints

Given the set of matrices with order n labeled as $A_i$:

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

SDP of MAX-CUT can be solved in a polynomial time

$$O((m+n^2)\cdot n^5 lg(n\cdot R)) \tag{12}$$

because the following analysis:

- $P_n$ is the space of the set of matrices, each one can be treated as a vector for convenience
- 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 $\theta^*$ 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.

### Steps

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.

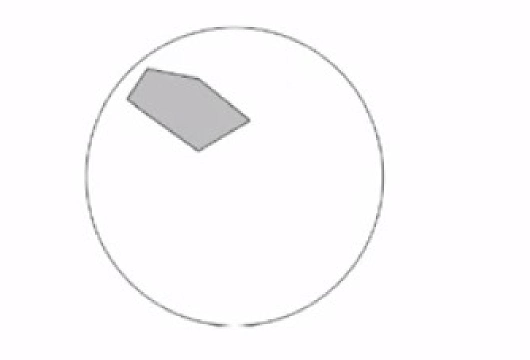


**Figure:** Graphic representation of Step 2 [15]

# Ellipsoid method

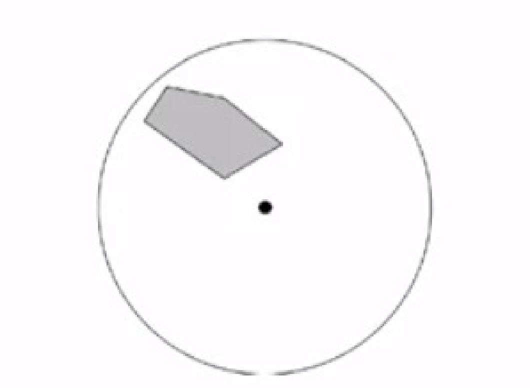


**Figure:** Graphic representation of Step 3 [15]

## Verification in Step 3

- 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.

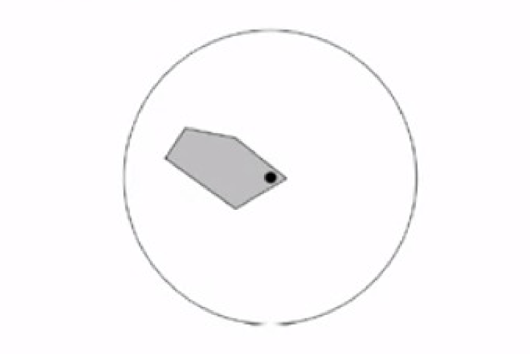


**Figure:** Graphic representation of Solution [15]

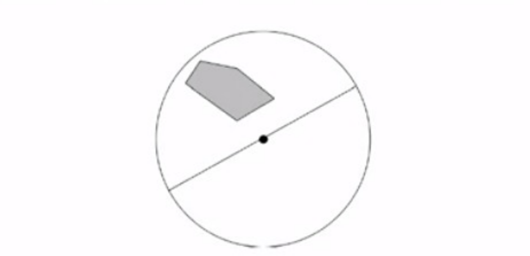


**Figure:** Graphic representation of failed iteration [15]

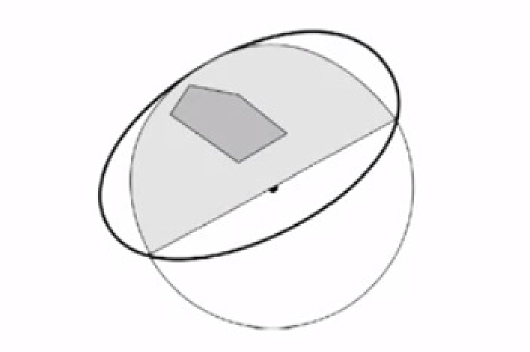


**Figure:** Graphic representation of half-ellipsoid covered by new ellipsoid [15]

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

# Goemans-Williamson Randomized Rounding

# Goemans-Williamson Randomized Rounding

We already know how to **formulate Max-Cut via SDP**, and that **SDPs can be solved**.

Nonetheless, we still need to **convert an SDP solution back into a solution for Max-Cut** [GO11].

Goemans and Williamson proposed a way to perform this conversion using **randomized rounding**.

# Goemans-Williamson Randomized Rounding

Recall that $v_i \in S_n$, where $S_n$ is the $n$-dimensional unit sphere, because $\|v_i\| = 1$ [GW95].

Cut $v$ in half with a **hyperplane** that passes through the origin. It produces a **bipartition** of $V$ [GO11].

Then, choose a random normal vector to the hyperplane $r$. The probability that two vectors $v_i$ and $v_j$ are separated by a random hyperplane is [KZ16]:

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

where $\theta$ is the angle formed by $v_i$ and $v_j$.

$v_i \cdot v_j = \cos\theta$, then $\theta = \cos^{-1}(v_i \cdot v_j)$.

Then, according to [GO11]:

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

and recall that the SDP formulation is as follows:

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

# Goemans-Williamson Randomized Rounding

*SDPOpt* is an upper bound on **E**[cut value]

To conclude that **E**[cut value] $\geq \alpha$*SDPOpt*, we must find $\alpha$, which is the approximation ratio of the algorithm [GW95].

So, to find $\alpha$, we calculate:

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

# Goemans-Williamson Randomized Rounding

$$MAXCUT \geq \mathbf{E}[\text{GW94 cut}] \geq 0.87856 \cdot SDPOpt \geq MAXCUT$$

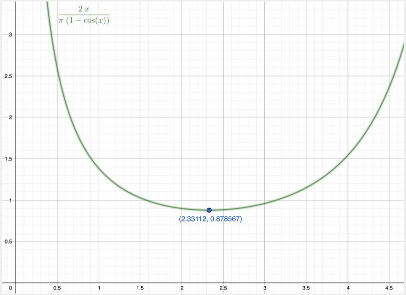



**Figure:** Geogebra plotting of $\frac{2}{\pi}\frac{\theta}{1-\cos\theta}$

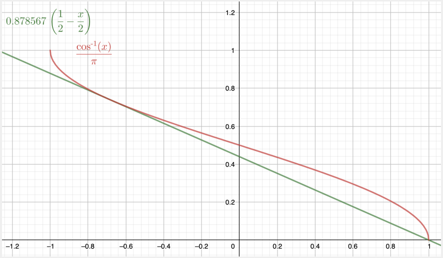



**Figure:** 'Graphical proof' of $\mathbf{E}[\text{cut value}] \geq \alpha SDPOpt$

The value of the *SDPOpt* is no more than 12.3% higher than the value of the NP-hard problem *MAXCUT*.

- 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.
- 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. [Fre09].
- 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.

# Conclusions

- 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.

# References I


- Stephen Boyd and Lieven Vandenberghe, *Applications of semidefinite programming*.
- Robert M. Freund, *Introduction to semidefinite programming (sdp)*, 2009.
- Anupam Gupta and Ryan O'Donnell, *Lecture 10: Semidefinite programs and the max-cut problem*.
- Michel X. Goemans and David P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, Journal of the Association for Computing Machinery **42** (1995), no. 6.
- Haim Kaplan and Uri Zwick, *Max cut using semidefinite programming*.
- Miguel Rocha and Pedro G. Ferreira, *Chapter 14 - graphs and biological networks*, Bioinformatics Algorithms (Miguel Rocha and Pedro G. Ferreira, eds.), Academic Press, 2018, pp. 289 – 311.

# References II

Sushant Sachdeva, *Max-cut and semidefinite programming*, February 2015.

Lieven Vandenberghe and Stephen Boyd, *Semidefinite programming*, Mathematical Programming, State of the Art **38** (1994), no. 1, 276–308.