UGP: Report

Areen Mahich

April 25, 2025

Acknowledgments

I would like to express my sincere gratitude to Professor Sutanu Gayen and my TA Debjyoti Dey for their invaluable guidance and mentorship throughout the course of this Undergraduate Project.

Abstract

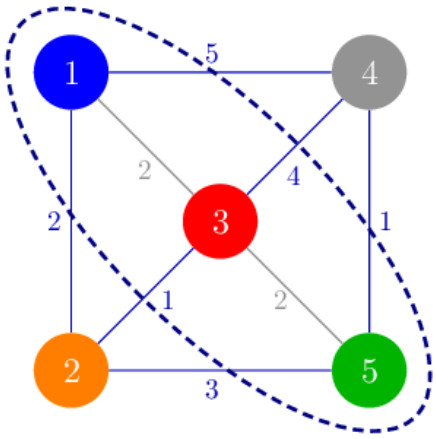
The Max-Cut problem is one of Karp's 21 classical NP-hard problems, for which no known polynomial-time exact algorithm is believed to exist. In this work, we explore the Goemans-Williamson algorithm, a celebrated approximation algorithm that guarantees a solution at least approximately 0.878 of the optimal cut weight. This algorithm leverages semidefinite programming (SDP) and randomized rounding via hyperplane cuts, making it one of the most well-known and influential approaches for tackling the Max-Cut problem.

The Max-Cut problem is a classical graph problem. It is often approached using the Goemans-Williamson algorithm.

Contents

1 Introduction to the Problem

The Max-Cut problem is a fundamental problem in graph theory and combinatorial optimization, which seeks to partition the vertex set of a given graph into two disjoint subsets such that the total weight of edges crossing between the subsets is maximized. Formally, given an undirected graph $G = (V, E)$ with a weight $w(e)$ assigned to each edge $e \in E$, the objective is to find a cut $(S, V \setminus S)$ that maximizes the sum of the weights of edges between S and $V \setminus S$.



1.1 Motivation

The Max-Cut problem is a classical and widely studied problem in graph theory and combinatorial optimization. It serves as a representative example of an NP-hard problem that is easy to state but computationally challenging to solve exactly.

Understanding Max-Cut is valuable not only from a theoretical perspective but also due to its broad range of applications in various fields. These include:

- **VLSI design** – circuit partitioning for layout optimization
- **Statistical physics** – modeling spin glass systems
- **Clustering** – partitioning data points based on similarity
- **Network design** – identifying bottlenecks or optimal disconnections
- **Image segmentation** – separating foreground from background
- **Social networks** – detecting community structures

Its rich mathematical structure and wide applicability make it a key problem in theoretical computer science and applied optimization, and studying efficient approximations like the Goemans-Williamson algorithm provides insights that extend well beyond this single problem.

1.2 Hardness Question

First, let us consider the brute force approach to solve the problem, in which we evaluate all the possible partitions and check them one by one. The number of possible partitions is $2^{|V|-1} - 1$, where $2^{|V|}$ represents the total number of partitions. We divide this by 2 to avoid symmetry and subtract 1 to exclude the empty subset.

This approach has an exponential time complexity, and since no polynomial-time algorithm has been discovered yet, the problem remains computationally hard and is classified as an NP-hard problem.

Additionally, the Max-Cut problem can be framed as an integer programming problem when it is converted from its visual form to a mathematical representation.

Let w_{ij} be the weight of the edge connecting the vertices i and j, and let c denote the value of the Max-Cut. Then, the Max-Cut can be expressed as:

$$c = \sum_{i,j \in V} \frac{w_{i,j}(1 - x_i \cdot x_j)}{4}$$

subject to the constraints:

$$x_i \in \{-1, +1\}$$

The values of x_k are determined based on the partition to which vertex k belongs: -1 represents one partition, and $+1$ represents the other. If both vertices i and j belong to the same partition, the product $x_i \cdot x_j$ is $+1$, which contributes zero to the sum. When they belong to different partitions, the product becomes -1, which results in a factor of 2. We then divide by 2 to make the contribution 1. Additionally, due to the symmetry between i and j, we divide by 2 again.

1.3 Approximation: A Gentle workaround

Now that we have seen the integer programming formulation of the Max-Cut problem, it is evident that such problems are generally very difficult to solve due to their NP-hard nature. To make the problem more tractable, we employ relaxation techniques. The Goemans-Williamson algorithm, a famous approach to solving the Max-Cut problem approximately, relaxes the binary constraints by representing x_i as vectors in a high-dimensional space and uses semi-definite programming to solve the relaxed problem.

Bonus: The c=0.5 version of the problem is very easy to look over using greedy algorithm and pigeon-hole principle.

2 Goemans-Williamson Algorithm

2.1 Original Problem Formulation

The Max-Cut problem can be approached through an Integer Programming (IP) formulation.The original Max-Cut problem can be formulated as an integer program. Let $x_i \in \{-1, 1\}$ be a binary decision variable associated with each vertex i, and w_{ij} denote the weight of the edge between vertices i and j. The objective function is designed to maximize the weight of edges that are cut, i.e., edges where $x_i \neq x_j$. This is given by the following optimization problem:

$$\max_{x_i \in \{-1,1\}} \frac{1}{4} \sum_{i,j} w_{ij} \left(1 - x_i x_j\right)$$

Here, $x_i x_j$ takes the value 1 if the two vertices are on the same side of the cut, and -1 otherwise. The goal is to maximize the sum of the weights of edges where $x_i \neq x_j$.

To represent the problem more succinctly, recognize that any rank-1 positive semi-definite (PSD) matrix X with $X_{ii} = 1$ can be written as $X = xx^\top$, where $x \in \{-1, 1\}^n$. we can express the variables as a rank-1 matrix X where:

$$X = xx^\top \quad \text{with} \quad X_{ij} = x_i x_j \quad \text{and} \quad X_{ii} = 1$$

Thus, the objective function becomes:

$$\frac{1}{4}\sum_{i,j} w_{ij}\,(1 - X_{ij})$$

This allows us to rewrite the original problem (P1) in matrix form as follows:

$$\max_X \frac{1}{4}\sum_{i,j} w_{ij}\,(1 - X_{ij}) \quad \text{subject to} \quad X = xx^\top, X_{ii} = 1$$

While this reformulation makes the problem more compact, it is still non-convex due to the rank-1 constraint on X, which makes it difficult to solve directly.

2.2 SDP Relaxation

To make the problem more tractable, we relax the rank-1 constraint on X while retaining the positive semi-definiteness (PSD) condition. The resulting problem is a Semidefinite Programming (SDP) relaxation of the original Max-Cut problem:

$$\text{(P3)} \quad \max_X \frac{1}{4}\sum_{i,j} w_{ij}\,(1 - X_{ij}) \quad \text{subject to} \quad X \succeq 0, X_{ii} = 1$$

This relaxation is now a standard SDP problem, which can be solved efficiently using well-established algorithms. The three key changes made during the relaxation are:

1. The rank-1 constraint is relaxed to allow for a general positive semi-definite (PSD) matrix X.
2. The binary constraints $x_i \in \{-1, 1\}$ are replaced by continuous variables $X_{ij} \in \mathbb{R}$.
3. The problem is now an approximate solution rather than an exact solution to the original combinatorial problem.

While the relaxation may not always provide an exact solution, it provides a powerful approximation and is the basis for many approximation algorithms for the Max-Cut problem.

2.3 Solving the SDP

After formulating the problem as an SDP, we use an appropriate solver to compute the optimal matrix X^*.We call this function as Solve_SDP. This matrix satisfies:

$$X^* \succeq 0, \quad X^*_{ii} = 1 \quad \forall i \in V$$

However, X^* is not guaranteed to be rank-1, so we cannot directly extract a solution x from it. Instead, we perform a decomposition to extract vector embeddings.

2.4 Vector Recovery via Decomposition

To obtain a vector representation from X^*, we use a decomposition method such as Cholesky or eigenvalue decomposition. If $X^* = RR^T$, then the i-th row of R can be viewed as a vector $\mathbf{r}_i$ corresponding to vertex i.

If Cholesky decomposition fails due to numerical issues (e.g., X^* not being full-rank), we can perform eigen-decomposition:

$$X^* = Q\Lambda Q^T, \quad R = Q\sqrt{\Lambda}$$

This results in a set of vectors $\{\mathbf{r}_i\}$ on the unit sphere, where $\langle \mathbf{r}_i, \mathbf{r}_j \rangle = X^*_{ij}$.

2.5 Hyperplane Rounding

To convert the vector solution into a discrete partition ie.we want to reduce down our vector solution in n dimensions to the binary or the one dimension to get our cut. In Goemans-Williamson algorithm we uses randomized hyperplane rounding:

- Choose a random vector $\mathbf{u}$ uniformly from the unit sphere (e.g., by sampling $\mathcal{N}(0, I_n)$ and normalizing).
- Assign vertex i to subset S if $\langle \mathbf{r}_i, \mathbf{u} \rangle \geq 0$, and to $V \setminus S$ otherwise.

This defines a random hyperplane through the origin, and the two sides of the hyperplane determine the two subsets of the cut. The probability that an edge (i, j) is cut depends on the angle θ between $\mathbf{r}_i$ and $\mathbf{r}_j$.

2.6 Geometric Intuition Behind SDP Relaxation

The original Max-Cut problem involves binary variables $x_i \in \{-1, 1\}$, where each x_i denotes the side of the cut to which vertex i belongs. This formulation imposes strict combinatorial constraints, making the optimization problem NP-hard.

To make the problem tractable, we relax the binary constraints by allowing each x_i to be replaced by a vector $\mathbf{u}_i \in \mathbb{R}^n$. This change introduces geometric flexibility into the problem, enabling the use of convex optimization techniques like semidefinite programming.

The key idea is that the product $x_i x_j$ in the original formulation, which equals 1 if i and j are on the same side of the cut and -1 otherwise, can be approximated by the inner product $\langle \mathbf{u}_i, \mathbf{u}_j \rangle = \cos \theta_{ij}$ in the relaxed version. Here, θ_{ij} is the angle between the vectors $\mathbf{u}_i$ and $\mathbf{u}_j$.

This relaxation preserves some notion of similarity or alignment between variables while expanding the feasible region. As a result, the semidefinite program becomes solvable in polynomial time.

However, this flexibility comes with a trade-off. Since the vectors $\mathbf{u}_i$ are allowed to lie anywhere on the unit sphere, the resulting solution may not correspond directly to any feasible binary assignment. In other words, the relaxed solution contains *less information* about the exact discrete cut — a fundamental reason why we resort to randomized rounding to extract a near-optimal integral solution.

Therefore, while the SDP relaxation does not yield the exact maximum cut, it provides a powerful approximation framework that captures essential geometric structure and yields a provably good solution upon rounding.

3 Algorithm Pseudo-code

The following algorithm provides a semidefinite programming (SDP) based approximation for the Max-Cut problem, achieving an expected approximation ratio of at least 0.878.

Algorithm 1 Goemans-Williamson Max-Cut Approximation

Require: A weighted undirected graph $G = (V, E)$ with edge weights $w_{ij} \geq 0$
Ensure: A cut $(S, V \setminus S)$ with expected weight $\geq 0.878 \cdot \text{OPT}$

Construct Weight Matrix
$n \leftarrow |V|$
Initialize $W \in \mathbb{R}^{n \times n}$ as a zero matrix
for each $(i, j) \in E$ **do**
    $W_{ij} \leftarrow w_{ij}$ and $W_{ji} \leftarrow w_{ij}$
end for
Formulate SDP Relaxation
Define symmetric matrix $X \in \mathbb{R}^{n \times n}$ with:
    $X \succeq 0$ and $X_{ii} = 1$ for all i
Objective: maximize $\frac{1}{4} \sum_{i,j} W_{ij}(1 - X_{ij})$
Solve the SDP
$X^* \leftarrow$ optimal solution to the above SDP
Extract Vectors from X^*
Try Cholesky decomposition: $X^* = V^\top V$
if Cholesky fails **then**
    Compute eigen-decomposition: $X^* = Q \Lambda Q^\top$
    Set $V \leftarrow Q\sqrt{\Lambda}$
end if
Hyperplane Rounding
Sample random vector $r \sim \mathcal{N}(0, I_n)$
Normalize: $r \leftarrow r/\|r\|$
Define cut: $S \leftarrow \{i \in V \mid \langle V_i, r \rangle \geq 0\}$
return $(S, V \setminus S)$

Time Complexity Summary

- Weight matrix construction: $\mathcal{O}(|E|)$
- SDP formulation: $\mathcal{O}(n^2)$
- SDP solving: $\mathcal{O}(n^3)$ to $\mathcal{O}(n^4)$
- Decomposition (Cholesky/eigen): $\mathcal{O}(n^3)$
- Rounding: $\mathcal{O}(n)$

Overall complexity: Dominated by SDP solving step, i.e., up to $\mathcal{O}(n^4)$

4 Finding the ratio of approximation

Now after doing the computation done in the previous section, we must now verify the ratio, that we had promised at the beginning.

Let us call our algorithm as A and it returns S on input G. Let's assume Solve_SDP returns the optimal solution X and the exact decomposition matrix is U.

we want to show :

$$\mathbb{E}[W(S)] \geq \alpha \cdot OPT(G) \quad \text{where} \quad \alpha > 0.878$$

In the discussed algorithm, the only random element involved was **r**, the randomly chosen hyperplane. In our computation, we use randomized algorithms where we define the indicator variable Y_{ij} as follows:

$$Y_{ij} = \begin{cases} 1 & \text{if } i \in S, j \in T \text{ or vice versa} \\ 0 & \text{otherwise} \end{cases}$$

The expected weight of the cut can be writted as shown:

$$\mathbb{E}[W(S)] = E[\frac{1}{2} \sum_{i \in V} \sum_{j \in V} w_{ij} \cdot \mathbb{Y}_{ij}]$$

By linearity of expectation

$$= \frac{1}{2} \sum_{i \in V} \sum_{j \in V} w_{ij} \cdot \mathbb{E}[Y_{ij}]$$

For Xij$\neq$ 1

$$= \frac{1}{2} \sum_{i \in V} \sum_{j \in V} \frac{w_{ij} \cdot \mathbb{E}[Y_{ij}]}{(1 - X_{ij})} \cdot (1 - X_{ij})$$

$$\geq \min_{i,j \in V} \left\{ \frac{2\,\mathbb{E}(Y_{ij})}{1 - X_{ij}} \right\} \cdot \frac{1}{4} \sum_{i \in V} \sum_{j \in V} w_{ij}(1 - X_{ij})$$

Since Y_{ij} is the indicator random variable,

$$= \min_{i,j \in V} \left\{ \frac{2 \Pr(Y_{ij} = 1)}{1 - X_{ij}} \right\} \cdot \frac{1}{4} \sum_{i \in V} \sum_{j \in V} w_{ij}(1 - X_{ij})$$

$$\geq \min_{i,j \in V} \left\{ \frac{2 \Pr(Y_{ij} = 1)}{1 - X_{ij}} \right\} \cdot \mathrm{OPT}(G)$$

In the last step, initially the term we had is the solution with no rank constraint on the matrix X but the case we are dealing with is the specific case where the rank of the matrix X is 1 so a solution without restriction will always yield a solution that is at least the solution with a constraint.

Now our goal is to show that:

$$0.878 \leq \min_{i,j \in V} \left\{ \frac{2 \Pr(Y_{ij} = 1)}{1 - X_{ij}} \right\} \cdot \mathrm{OPT}(G)$$

The event $\Pr(Y_{ij} = 1)$ occurs only when the i^{th} and j^{th} vertices lie on different sides of the cut. This is possible only when they give different signs for the inner product with the selected random vector **r**.This occurs in two cases when one gives a positive product and the other a negative with **r** and vice versa.

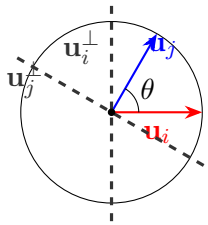
Let θ be the angle between the vectors $\mathbf{u}_i$ and $\mathbf{u}_j$. We show the following:

$$\Pr(\mathbf{u}^\top \mathbf{r}_i \geq 0 \text{ and } \mathbf{u}^\top \mathbf{r}_j < 0) = \frac{\theta}{2\pi}$$

Due to symmetry

$$Pr(Y_{ij} = 1) = 2 \cdot \Pr(\mathbf{u}^\top \mathbf{r}_i \geq 0 \text{ and } \mathbf{u}^\top \mathbf{r}_j < 0)$$

The above expression is also **rotational symmetric** (the probability remains the same if we rotate all the vectors by the same angle). The result obtained is highly intuitive as it can be clearly understood with the help of a diagram.



From the geometric diagram, we can visualize the relationship between the vectors and the cut probability. Consider two unit vectors $\mathbf{u}_i$ and $\mathbf{u}_j$. The black dotted lines represent hyperplanes perpendicular to each of these vectors, dividing the plane into four distinct regions. Among these, two regions correspond to the angle θ between $\mathbf{u}_i$ and $\mathbf{u}_j$, and the other two correspond to the supplementary angle $\pi - \theta$.

Now, a random vector $\mathbf{r}$ drawn uniformly from the unit sphere lies in the θ region with probability proportional to the angle θ. In this region, $\mathbf{r}$ gives opposite signs when projected onto $\mathbf{u}_i$ and $\mathbf{u}_j$, i.e., $\text{sign}(\langle \mathbf{r}, \mathbf{u}_i \rangle) \neq \text{sign}(\langle \mathbf{r}, \mathbf{u}_j \rangle)$. Therefore, by basic probability,

$$\Pr(Y_{ij} = 1) = \frac{2\theta}{2\pi} = \frac{\theta}{\pi}$$

where Y_{ij} is the indicator variable that is 1 when i and j lie on opposite sides of the hyperplane (i.e., the edge (i, j) is cut).

Since $X = UU^\top$, we have

$$X_{ij} = \mathbf{u}_i^\top \mathbf{u}_j = \cos\theta_{ij}$$

From the analysis, we get

$$\Pr(Y_{ij} = 1) = \frac{\theta_{ij}}{\pi}$$

Using this, we compute:

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

Let us define the function:

$$f(\theta) = \frac{2\theta}{\pi(1 - \cos\theta)}$$

We differentiate this to find the minimum:

$$f'(\theta) = \frac{2\left[\pi(1 - \cos\theta) - \theta \cdot \pi \sin\theta\right]}{[\pi(1 - \cos\theta)]^2}$$

Solving $f'(\theta) = 0$ numerically yields the minimum at approximately:

$$\theta^* \approx 2.3$$

Substituting this value back into $f(\theta)$, we get:

$$f(2.3) \approx \frac{2 \cdot 2.3}{\pi(1 - \cos 2.3)} \approx 0.878$$

Finally, we get:

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

Let $\mathrm{OPT}_{\mathrm{SDP}}$ be the solution of the relaxed problem Expected value of rounded solution:

$$\mathbb{E}[\mathrm{Cut}] \geq \alpha_{GW} \cdot \mathrm{OPT}_{\mathrm{SDP}}, \quad \alpha_{GW} \approx 0.878$$

This is the best-known approximation ratio assuming the Unique Games Conjecture.

5 Experiment

We ran some test for randomly generated graph and got highly satisfying results.

```
Run 1:
Adjacency Matrix:
[[0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1]
 [1 0 0 0 0 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1]
 [0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 0]
 [1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0]
 [0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1]
 [1 1 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0]
 [0 1 0 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0]
 [0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 0 0 1]
 [0 1 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0]
 [0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 1]
 [0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0]
 [0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0]
 [0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0]
 [1 0 0 1 1 0 0 1 0 0 0 0 1 0 0 1 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0]
 [0 0 1 1 0 1 0 1 0 1 0 1 1 1 0 0 0 0 1 0]
 [0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0]
 [1 1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0]]
GW Cut Value: 50
Optimal Cut Value: 50
SDP Relaxation Value: 52.6016
Approximation Ratio (GW/OPT): 1.0000
Approximation Ratio (GW/SDP): 0.9505
```

```
Run 8:
Adjacency Matrix:
[[0 0 0 0 0 1 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0 0]
 [0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 0 1 0 1 0]
 [0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 1 1 1]
 [0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1]
 [0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0]
 [1 0 0 0 1 0 1 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1]
 [0 0 0 0 0 1 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 0]
 [1 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1]
 [0 1 0 0 0 1 1 1 0 1 1 1 0 1 0 0 0 0 1 0 0 0]
 [1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0]
 [1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0]
 [0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0]
 [1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 0 1 0 1]
 [0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0]
 [1 0 1 0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0]
 [0 1 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 0 0]
 [1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1]
 [1 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 0 1]
 [0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0]
 [0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0]
 [0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0]]
GW Cut Value: 54
Optimal Cut Value: 58
SDP Relaxation Value: 59.1377
Approximation Ratio (GW/OPT): 0.9310
Approximation Ratio (GW/SDP): 0.9131
```

```
Run 1:
Adjacency Matrix:
[[0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1 0 0 1]
 [0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0]
 [1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1]
 [0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0]
 [0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0]
 [1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1]
 [0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0]
 [0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 0 1]
 [0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0]
 [0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 1 1 1]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 0 0 1 0 1]
 [0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 1]
 [0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0]
 [0 1 0 1 1 1 0 0 0 0 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1]
 [1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 0 1 0 0]
 [0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0]
 [0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0]
 [1 0 0 1 1 1 0 0 0 1 0 0 0 0 0 1 1 1 0 1 0 0 1 0 0]
 [1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1]
 [0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0]
 [1 0 1 0 0 1 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0]]
GW Cut Value: 64
Optimal Cut Value: 66
SDP Relaxation Value: 68.4616
Approximation Ratio (GW/OPT): 0.9697
Approximation Ratio (GW/SDP): 0.9348
------------------------------------------------------------
```

6 Summary

6.1 Step 1: SDP Relaxation

The Goemans-Williamson algorithm starts by relaxing the Max-Cut problem using a Semi-Definite Programming (SDP) formulation. The Max-Cut problem is NP-hard, so we first solve a relaxed version of the problem where the cut variables are replaced by real vectors.

The SDP formulation is as follows:

$$\max_{\{u_i \in \mathbb{R}^n : \|u_i\|=1\}} \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - u_i^\top u_j)$$

where u_i are vectors corresponding to each vertex i, and w_{ij} is the weight of the edge between vertices i and j.

This relaxation allows us to work in a continuous space rather than a discrete one.

6.2 Step 2: Geometric Interpretation

The vectors u_i represent the directions of the vertices in a high-dimensional space. The goal is to find a set of vectors that approximate the structure of the graph.

Geometrically, the angle between vectors u_i and u_j is related to the cut value between vertices i and j. The larger the angle (or the smaller the dot product $u_i^T u_j$), the larger the cut between i and j.

The SDP optimization seeks to minimize the dot product $u_i^T u_j$, and by doing so, it maximizes the cut between vertices.

6.3 Step 3: Random Hyperplane and Partitioning

Once the SDP problem is solved and we have the vectors u_i, we proceed to partition the graph. We do this by selecting a random hyperplane.

Specifically, we sample a random vector r from the standard normal distribution $r \sim N(0, I)$, where I is the identity matrix. Then, we partition the vertices based on the sign of the dot product of u_i and r:

$$S = \{i \in V \mid u_i^T r \geq 0\}, \quad T = V \setminus S$$

This gives us a cut (S, T), where S and T are the two sets formed by the partitioning.

6.4 Step 4: Approximation Ratio and Theorem

The final step is to analyze the expected cut value. The expected value of the cut is at least 0.878 times the optimal cut value. This comes from the geometry of the random hyperplane and the structure of the vectors u_i.

Key Theorem: The expected cut value between the sets S and T is:

$$\mathbb{E}[\text{cut}(S,T)] \geq 0.878 \cdot \text{OPT}$$

where OPT is the optimal cut value.

Hence, the Goemans-Williamson algorithm achieves an approximation ratio of 0.878 for the Max-Cut problem.

7 Extensions

Extensions of the Max-Cut problem, such as K-Max-Cut, involve partitioning a graph into K subsets to maximize the cut. This extension includes developments like approximation algorithms that aim to find efficient solutions despite the problem's NP-hard nature. Semi-Definite Programming (SDP) is extended to handle multiple partitions, transforming the problem into a continuous optimization. Additionally, degree-constrained K-Max-Cut introduces vertex degree constraints for practical uses like network design. These extensions find applications in clustering, network optimization, and circuit design.

8 Improvements

Their is always a scope for improvement in any good thing also.

8.1 Ways to Improve the Algorithm

There are always ways to make good algorithms even better. For the Max-Cut algorithm, we can improve it by:

- Making the SDP solver faster (e.g., using parallel computing)
- Developing better rounding methods than random hyperplanes
- Simplifying the input graph by finding symmetrical parts

These changes could help the algorithm run faster while keeping (or even improving) its 0.878 approximation guarantee. For example, when dealing with social networks, we might first identify similar groups of users before running the full algorithm - this could make the problem much easier to solve.

8.2 Breakthroughs Beyond Goemans-Williamson (GW)

Recent advances in Max-Cut algorithms have significantly improved upon the classic GW 0.878-approximation:

- **Improved Approximations**:
  - 0.921 for dense graphs (KKMO 2007) via Gaussian noise stability
  - 0.943 for low-threshold-rank graphs (Bourgain 2021) using spectral methods
  - **Planar graphs**: Exact solution in $O(n^{3/2} \log n)$ via dual graph matching (Hadlock 1975)

- **Faster SDP Methods**:
  - Matrix Multiplicative Weights: $O(n^{2.5})$ runtime (Arora-Kale 2016)
  - Sublinear rounding: $(0.878 - \epsilon)$-approx in $O(n \cdot \text{poly}(1/\epsilon))$ (Lee 2020)
- **Combinatorial Advances**:
  - 0.614-approx in $O(|E|)$ (Trevisan 2019)
  - 0.7-approx for bounded-degree graphs (Feige 2023)
- **Theoretical Limits**:
  - GW's 0.878 optimal under UGC
  - Planar graphs remain polynomially solvable

9 Conclusion

The Goemans-Williamson algorithm elegantly bridges the gap between combinatorial optimization and continuous convex optimization. By transforming the NP-hard Max-Cut problem into a semidefinite program, it enables efficient solving and rounding to a near-optimal discrete solution. This groundbreaking approach has not only influenced a wide range of approximation algorithms but also solidified its place as a fundamental pillar in the theory of approximation algorithms.

References

1. M. X. Goemans and D. P. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming," *Journal of the ACM*, vol. 42, no. 6, pp. 1115–1145, 1995.

2. S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

3. S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

4. G. Strang, "Introduction to Linear Algebra", 5th ed. *Wellesley-Cambridge Press*, 2016.

5. A. Sreejith, "Max-Cut and Semidefinite Programming," Indian Institute of Technology Goa.
`https://www.iitgoa.ac.in/~sreejithav/misc/maxcut.pdf`

6. L. Lessard, "Rounding and Relaxation Techniques," CS524 Course Notes, University of Wisconsin-Madison.
`https://laurentlessard.com/teaching/cs524/slides/18%20-%20rounding%20and%20relaxation.pdf`

7. CS229 Staff, "Convex Optimization Overview," Stanford University.
`https://cs229.stanford.edu/section/cs229-cvxopt.pdf`

8. Max-Cut Implementation Notebook,(Google Colab) made by Areen Mahich.
`https://colab.research.google.com/drive/1J2TzDf3Mv7CQI6ZYhK2ElcVXz6aQfigK`

9. J. Håstad, "Some Optimal Inapproximability Results", *Journal of the ACM*, vol. 48, no. 4, pp. 798–859, 2001.

10. S. Khot, G. Kindler, E. Mossel, and R. O'Donnell, "Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs", *SIAM Journal on Computing*, vol. 37, no. 1, pp. 319–357, April 2007.