Relaxations of the Maximum Cut Problem

Demetrios V. Papazaharias, Carter Mann, Luca Wrabetz
Department of Industrial and Systems Engineering
University at Buffalo, Bell Hall, Buffalo, New York, 14260
{dvpapaza, cjmann3, lucawrab}@buffalo.edu

December 2019

Abstract

Maximum cut is a classic NP-Hard and considered to be one of the most difficult problems in combinatorial optimization. In this work we present several relaxations for the maximum cut problem. In addition to several well-known mathematical programs, we utilize the Laplacian matrix of G in order to compute an eigenvalue bound and further strengthen it with a subgradient algorithm. We then implement the Ellipsoid algorithm on the semi-definite programming relaxation of max cut. Finally, we construct a 2-swap simulated annealing heuristic in order to construct feasible solutions of Max Cut.

1 Introduction

The maximum cut problem (maxcut) is a classical problem in combinatorial optimization in which we are given a graph $G = (V, E)$, with node set $V = \{1, ..., n\}$ and edge set $E = \{i, j\} \subseteq \binom{V}{2}$. We are also given, for each edge $\{i, j\} \in E$, a non-negative weight w_{ij} . The maxcut problem consists in determining a subset of $S \subseteq V$ nodes for which the sum of the weights of the edges that cross from S to its complement $\bar{S}$ (i.e., the weight of the edges in the cut $(S, \bar{S})$ is maximized. In this particular case, we are assuming that the edges are undirected; however, this problem can be easily generalized to the directed case.

Unlike the minimization variant to this problem, which can be solved in polynomial time, the maxcut problem is NP-Hard. Given its nature, the maxcut problem is a highly studied problem in Graph Theory. Although no polynomial algorithm exists to solve maxcut (Unless we discover $P = NP$) a multitude of approximation algorithms and heuristics have been developed in an attempt to solve this problem to a relative level of optimality. One simplistic method is a local search algorithm, which starts with an arbitrary set S and then iteratively either adds or removes an edge, as long as it helps the objective function. A more complex method would be the Goemans and Williamson algorithm, which combines semi-definite programming and a rounding procedure to produce an approximate solution to the max-cut problem.

In this paper we will focus our attention on IP formulations to the maxcut, along with a variety of relaxations and a heuristic which will help us find upper and lower bounds respectively. In section 2 we will discuss two separate formulations to the maxcut. Section 3 will feature a variety of relaxations to the maxcut, which will help us find upper bounds to our problem. Section 4 will discuss a heuristic which utilizes two-swap simulated annealing approach to find a sufficient lower bound to our problem. Section 5 will display the computational analysis for our study, and finally section 6 will be the conclusion to our findings.

2 Mathematical Formulations

2.1 Formulation 1

In this formulation, we define binary variables x_i for each $i \in V$ that indicate whether node i belongs to S. we also define the binary variables y_{ij} to indicate whether edge $\{i, j\}$ is part of the cut. Using these variables, we can formulate this problem as follows:

$$
\begin{aligned}
z^* = \max \quad & \sum_{\{i,j\}\in E} w_{ij} y_{ij} && (1)\\
\text{s.t.} \quad & y_{ij} \leq 2 - x_i - x_j, \quad \{i,j\} \in E && (2)\\
& y_{ij} \leq x_i + x_j, \quad \{i,j\} \in E && (3)\\
& y_{ij} \geq x_i - x_j, \quad \{i,j\} \in E && (4)\\
& y_{ij} \geq x_j - x_i, \quad \{i,j\} \in E && (5)\\
& x_i \in \{0,1\}, \qquad i \in V && (6)\\
& y_{ij} \in \{0,1\}, \quad \{i,j\} \in E && (7)
\end{aligned}
$$

where constraint (2) imposes that if both ends of an edge are in S, this edge is not in the cut, (3) imposes that if both ends are in $\bar{S}$, this edge is not in the cut, (4) and (5) impose that if the ends of this edge are in different sets, this edge is in the cut, and (6) and (7) define the domain of the variables.

2.2 Formulation 2

In this formulation, we define variables z_i for each $i \in V$ to take the value of -1 if $i \in S$ and 1 if $i \notin S$. Using these variables, we can formulate this problem as follows:

$$
\begin{aligned}
z^* = \frac{1}{4} \max \sum_{i\in V} \sum_{j \in V} w_{ij}(1 - z_i z_j) && (8)\\
st. \quad z_i \in \{-1, 1\}, \qquad i \in V && (9)
\end{aligned}
$$

3 Relaxations for Max Cut

In this section we propose several relaxations, other than the linear programming relaxation for Max Cut. We restrict the contents of this sections to algorithms and necessary proofs for computing each bound. We postpone discussion of results, selected parameters and implementation for the computational study in Section 5.

3.1 Eigenvalue Relaxation

We define the Laplacian of graph G as the matrix $L(G) = D(G) - A(G)$ such that $D(G)_{ii} = \sum_{j\in V:\{i,j\}\in E} w_{ij}$, $D(G)_{ij} = 0$ for $i, j \in V, i \neq j$, and $A(G)_{ii} = 0$ for $i \in V$ and $A(G)_{ij} = w_{ij}$ for $i, j \in V$ such that $i \neq j$. The Laplacian is well studied in spectral graph theory. In this work we will utilize $L(G)$ in order to compute the upper bound for Max Cut.

Proposition 1. *The optimal value of z^* in the max cut problem defined on graph $G = (V, E)$ satisfies*

$$z^* \leq z^{ev} := \frac{n}{4} \lambda_{max}(L(G))$$

Proof. Consider the maximization problem

$$\max\{\mathbf{z}^\top L(G)\mathbf{z} \,|\, \mathbf{z} \in \{-1,1\}^n\}$$

Since $L(G) = D(G) - A(G)$, then $L(G)_{ii} = \sum_{j\in V} w_{ij}$ for $i \in V$ and $L(G)_{ij} = -w_{ij}$ for $i, j \in V$ such that $i \neq j$. We expand the objective function and obtain

$$\max_{\mathbf{z}\in\{-1,1\}} \sum_{i\in V}\sum_{j\in V} w_{ij} z_i z_i - \sum_{i\in V}\sum_{j\in V} w_{ij} z_i z_j$$

Furthermore,

$$\max_{\mathbf{z}\in\{-1,1\}} \sum_{i\in V}\sum_{j\in V} w_{ij}(z_i z_i - z_i z_j)$$

Since $z_i \in \{-1, 1\}$ then $z_i z_i = 1$, we can simplify the objective function to

$$\max_{\mathbf{z}\in\{-1,1\}} \sum_{i\in V}\sum_{j\in V} w_{ij}(1 - z_i z_j)$$

From the previous section we saw an almost identical formulation for max cut. The only difference being a multiplier of $\frac{1}{4}$. Let z^* represent the optimal solution for max cut

$$z^* = \frac{1}{4}\max\{\mathbf{z}^\top L(G)\mathbf{z} \,|\, \mathbf{z} \in \{-1,1\}^n\} \tag{10}$$

Taking the continuous relaxation of (10), namely $z_i \in [-1, 1]$ for $i \in V$, is equivalent to maximizing over the norm infinity, $\|\mathbf{z}\|_\infty \leq 1$. We can relax this further by maximizing over a ball of radius $\sqrt{n}$. In other words, our region is now defined where $\|\mathbf{z}\| \leq \sqrt{n}$.

$$z^* \leq \frac{1}{4}\max\{\mathbf{z}^\top L(G)\mathbf{z} \,|\, \mathbf{z} \leq \sqrt{n}\} \tag{11}$$

We can define our problem over the unit ball with a simple transformation. Let $\|\mathbf{z}\| = \sqrt{n}\mathbf{x}$ and we now have

$$z^* \leq \frac{n}{4}\max\{\mathbf{x}^\top L(G)\mathbf{x} \,|\, \|\mathbf{x}\| \leq 1\} \tag{12}$$

Since for any symmetric matrix $\mathbf{A}$, $\max\{\mathbf{x}^\top \mathbf{A}\mathbf{x} \,|\, \|\mathbf{x}\| \leq 1\} = \lambda_{\max(\mathbf{A})}$ and $L(G)$ is symmetric, then we have

$$z^* \leq z^{ev} := \frac{n}{4}\lambda_{max}(L(G)) \tag{13}$$

□

We can further strengthen this bound by introducing a vector $\mathbf{u} \in \mathbb{R}^n$, which changes the diagonal elements of $L(G)$. Our goal is to reduce the size of the maximum eigenvalue and thus reducing the eigenvalue bound. We propose an upper bound involving $L(G) + diag(\mathbf{u})$.

Proposition 2. *The optimal value of z^* in the max cut problem defined on graph $G = (V, E)$ satisfies*

$$z^* \leq z^{ev} := -\frac{1}{4}\sum_{i=1}^{n} u_i + \frac{n}{4}\lambda_{max}(L(G) + diag(\mathbf{u}))$$

for all $\mathbf{u} \in \mathbb{R}^n$

Proof. Consider the maximization problem

$$\max\{\mathbf{z}^\top(L(G) + diag(\mathbf{u}))\mathbf{z} | \mathbf{z} \in \{-1, 1\}\} \tag{14}$$

From the previous proof, we perform a similar technique to (14) to obtain the following optimization problem

$$\max_{\mathbf{z}\in\{-1,1\}} \sum_{i\in V} \left(\sum_{j\in V} w_{ij} + u_i \right) z_i z_i - \sum_{i\in V} \sum_{j\in V} w_{ij} z_i z_j$$

Rearranging terms and using the fact that $z_i z_i = 1$,

$$\max_{\mathbf{z}\in\{-1,1\}} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(1 - z_i z_j) + \sum_{i=1}^{n} u_i \tag{15}$$

Which is closely related to the objective function in our prevoius formulation. We obtain the following relationship between (15) and z^*:

$$z^* = \frac{1}{4} \max\{\mathbf{z}^\top(L(G) + diag(\mathbf{u}))\mathbf{z} \,|\, \mathbf{z} \in \{-1, 1\}\} - \frac{1}{4} \sum_{i=1}^{n} u_i$$

We repeat the process from the previous proof of relaxing the variables so that they are continuous and then further relaxing this problem onto a ball of radius $\sqrt{n}$.

$$z^* \leq \frac{1}{4} \max\{\mathbf{z}^\top(L(G) + diag(\mathbf{u}))\mathbf{z} \,|\, \|\mathbf{z}\| \leq \sqrt{n}\} - \frac{1}{4} \sum_{i=1}^{n} u_i$$

Let $\mathbf{z} = \sqrt{n}\mathbf{x}$.

$$z^* \leq \frac{n}{4} \max\{\mathbf{x}^\top(L(G) + diag(\mathbf{u}))\mathbf{x} \,|\, \|\mathbf{x}\| \leq 1\} - \frac{1}{4} \sum_{i=1}^{n} u_i$$

Finally, we obtain the inequality

$$z^* \leq z^{ev}(\mathbf{u}) = -\frac{1}{4} \sum_{i=1}^{n} u_i + \frac{n}{4} \lambda_{max}(L(G) + diag(\mathbf{u})) \tag{16}$$

□

3.2 Lagrangian Dual

In the previous section we have proved several upper bounds on max cut based on the Laplace $L(G)$, including one which changes of the diagonal of $L(G)$ with some vector $\mathbf{u} \in \mathbb{R}^n$. In this section we will compute the Lagrangian dual bound of maximum cut, which corresponds to the values of $\mathbf{u}$ such that $z^{ev}(\mathbf{u})$ is minimized.

$$z^{LD} = \min \left\{ -\frac{1}{4} \sum_{i=1}^{n} u_i + \frac{n}{4} \lambda_{max}(L(G) + diag(\mathbf{u})) \,\middle|\, \mathbf{u} \in \mathbb{R}^n \right\}$$

Consider the gradient of our objective function, $\nabla z^{ev}(\mathbf{u})$. In order to compute $\nabla\lambda_{max}(L(G)+diag(\mathbf{u}))$ we first use the fact that $L(G)+diag(\mathbf{u})$ is a symmetric square matrix and therefore $\lambda_{max}(L(G)+diag(\mathbf{u})) = \max\{\mathbf{x}^\top(L(G)+diag(\mathbf{u}))\mathbf{x} \mid \|\mathbf{x}\| \leq 1\}$. This optimization problem can further be expanaded to

$$\max_{\|\mathbf{x}\|\leq 1} \sum_{i\in V}\left(\sum_{j\in V} w_{ij} + u_i\right) x_i x_i - \sum_{i\in V}\sum_{j\in V} w_{ij}x_i x_j$$

We can see that this objective value is maximized for $x_k = 1$ where $k = \arg\max_i\{\sum_{j\in V} w_{ij} + u_i \mid \sum_{j\in V} w_{ij} + u_i > 0\}$, and $x_i = 0$ for $i \in V \setminus \{k\}$. If there exists no positive value of $\sum_{j\in V} w_{ij} + u_i$ then the objective function is maximized for $\mathbf{x} = \mathbf{0}$. Therefore, $\nabla\lambda_{max}(L(G)+diag(\mathbf{u})) = e_k$ if $\lambda_{max}(L(G)+diag(\mathbf{u})) \geq 0$ and $\mathbf{0}$ otherwise and $\nabla z^{ev}(\mathbf{u})$ can be computed as

$$\nabla z^{ev}(\mathbf{u})_k = \begin{cases} \frac{n-1}{4} & \text{if } k = \arg\max_i \left\{ \sum_{j\in V} w_{ij} + u_i \,\middle|\, \sum_{j\in V} w_{ij} + u_i > 0\right\} \\ -\frac{1}{4} & \text{otherwise} \end{cases}$$

3.2.1 Subgradient Algorithm

We implement the following subgradient algorithm in order to compute z^{LD}. We postpone details on our implementation decisions such as stopping criteria, initial solutions and choice of sequence h_k for a later section.

Algorithm 1: Subgradient Algorithm for MaxCut

```
Result: Computes the Lagrangian dual bound $z^{LD}$
Select an initial solution $\mathbf{u}_0$ and appropriate sequence $\{h_k\}_{k=0}^{\infty}$;
for $k \geq 0$ do
    Compute $z^{ev}(\mathbf{u})$ and $\nabla z^{ev}(\mathbf{u})$;
    $\mathbf{u}_{k+1} \leftarrow \mathbf{u}_k - h_k \frac{\nabla z^{ev}(\mathbf{u})}{\|\nabla z^{ev}(\mathbf{u})\|}$;
end
```

3.3 Semi Definite Relaxation

The semi-definite relaxation of formulation 2 is expressed as follows:

$$z^* = \frac{1}{4}\max \quad \sum_{i\in V}\sum_{j\in V} w_{ij}(1-Y_{ij}) \tag{17}$$

$$\text{s.t} \quad Y_{ii} = 1, \quad i \in V \tag{18}$$

$$\mathbf{Y} \succeq 0 \tag{19}$$

The formulation is relaxed by introducing the matrix $\mathbf{Y}$. The elements Y_{ij} of this matrix correspond to the multiplication of the decision variables z_i and z_j from the original formulation. Additionally, the variables are relaxed such that $Y_{ij} \in \mathbb{R}$. Constraint (18) enforces the fact that $z_i = z_i$ from the original formulation, as the multiplication of 1 and 1 or -1 and -1 results in 1. This also makes sense conceptually, as the same node can't be in two different sides of the cut. Thus, we will maintain the diagonal of the matrix $\mathbf{Y}$ fixed during the solution of this formulation. Constraint (19), forces the values of Y_{ij} variables to stay between -1 and 1, and ultimately converge close to either 1 or -1 in feasible solutions.

To solve the formulation, we implement the ellipsoid algorithm. The algorithm is generally efficient and effective for convex problems, such as the relaxed formulation. However, we know that its nature causes it to be slow in practice in many cases, as it can iterate between infeasible and feasible solutions for a large number of steps. We will discuss this further in our computational analysis.

The ellipsoid algorithm relies on affine transformations between a unit "ball", $B(y_k, 1)$, and ellipsoids $E(y_k, \mathbf{H_k})$. Notice, that the value $\mathbf{y}$ is the current proposed solution at iteration k. Ultimately, the algorithm works on each iteration and its current $\mathbf{y_k}$ in the space of $B(0, 1)$, and transforms new solutions in the ellipsoid space to continually shrink and shift the ellipsoids and converge to better, feasible solutions. At each iteration, if the current solution is infeasible, a separating cut is added to move the next $\mathbf{y_k}$ towards feasibility: if it is not PSD, the eigenvector associated with the most violating eigenvalue is used to calculate the direction to move in for y_{k+1}. If the solution is feasible, we move in the direction of the objective function's gradient.

Suppose we define the feasible region for our semi-definite relaxation to be the set $K = \{y_{ij} : Y_{ii} = 1, Y_{ij} = Y_{ji}, Y \succeq 0, i \in V, j \in V\}$. In reality, when the algorithm is implement, we only need to maintain a vector of variables $y_i, i \in V$, since the constraints $Y_{ii} = 1, Y_{ij} = Y_{ji}$ can be trivially disgarded without loss of generality, and the matrix $\mathbf{Y}$ can be constructed and converted back to the list $\mathbf{y}$ at every iteration. The algorithm is initialized by finding a value $R \in \mathbb{R}$ such that $K \subseteq B(0, R)$, and thus, our initial solution, $\mathbf{y_0}$, is the zero vector, and $\mathbf{Y}$ is the identity. Thus, we wish for our initial ellipsoid to be $B(0, R)$, so we initialize $\mathbf{H_0}$ to be $R^2\mathbf{Y}$. Thus, we iterate over the algorithm described earlier as follows:

Algorithm 2: Ellipsoid Algorithm for MaxCut.

```
Result: Solution to Semi-Definite Relaxation
$R \leftarrow 2n$;
$\mathbf{Y_0} \leftarrow$ identity;
$\mathbf{y_0} \leftarrow \mathbf{0}$ ;
$\mathbf{H_0} \leftarrow R^2\mathbf{Y}$;
$\mathbf{g} \leftarrow \nabla z$;
$\epsilon \leftarrow 0.001$;
$change \leftarrow 100$;
$k \leftarrow 0$;
while $change > \epsilon$ do
    if $\mathbf{Y_k} \succeq 0$ then
        $\mathbf{y_{k+1}} \leftarrow$ ellipsoidshift($\mathbf{Y_k}$, $\mathbf{g}$, $\mathbf{H_k}$) ;
        $\mathbf{H_{k+1}} \leftarrow$ newH($\mathbf{Y_k}$, $\mathbf{g}$, $\mathbf{H_k}$) ;
    else
        $\overline{g} \leftarrow$ separatingcut(min_eigenvector) ;
        $\mathbf{y_{k+1}} \leftarrow$ ellipsoidshift($\mathbf{Y_k}$, $\overline{\mathbf{g}}$, $\mathbf{H_k}$) ;
        $\mathbf{H_{k+1}} \leftarrow$ newH($\mathbf{Y_k}$, $\overline{\mathbf{g}}$, $\mathbf{H_k}$) ;
    end
    $k \leftarrow k + 1$ ;
    $change \leftarrow ||\mathbf{y_{k+1}} - \mathbf{y_k}||$ ;
end
return $\mathbf{y}$ ;
```

The side funtions ellipsoid shift, and new H, are implementations of the closed-form matrix operations from the project guidelines. The vector **g** remains constant, and is the gradient of the objective function. This is the direction we want to move in when our solution is already feasible. $\overline{\mathbf{g}}$ is calculated by taking the gradient of the most violating eigenvector in the function separatingcut.

4 Heuristic Solution

In this section we introduce a 2-swap simulated annealing heuristic to compute good feasible solutions for Max Cut. The outline of this sections is as follows

4.1 Greedy Initial Solution

In order to obtain an initial feasible solution for our heuristic we first sort the edges by weight in descending order. Then. for each edge $\{i, j\} \in E$ we check which set i and j are assigned to. If either of the endpoints are not assigned to S or $\overline{S}$, without loss of generality say j, the place j into the opposite set of i and add $\{i, j\}$ to the cut. If both nodes are unassigned then $S \leftarrow S \cup \{i\}$ and $\overline{S} \leftarrow \overline{S} \cup \{j\}$ and we add $\{i, j\}$ to the cut. Otherwise, we cannot place $\{i, j\}$ in the cut and maintain a feasible solution. This construction can be completed in $O(m \log m)$, where the dominant procedure is the sorting of E. In addition, this procedure will place every vertex v into S or $\overline{S}$, unless v has degree 0. In that case we can remove v from V and it does not effect the solution.

4.2 2-Swap Procedure

At each iteration of our heuristic we will randomly select one vertex $i \in V$ and another vertex j such that $j \in \overline{S}$ if $i \in S$ and vice-versa. We will navigate through the solution space of Max Cut by swapping the sets which i and j belong to. In order to efficiently compute the change in the objective function from this swap we will only check the neighbors of i and j. Suppose $i \in S$ and $j \in \overline{S}$, for each $k \in N(i)$ such that $k \in S$, we add w_{ik} to the current objective value as i will belong to the complement of S after the swap. Conversely, for each $\ell \in \overline{S}$ we subtract $w_{i\ell}$ from the current objective. However, we must account for the case when $j \in N(i)$. Since j was originally in $\overline{S}$, w_{ij} was subtracted from the current solution. However, j will be placed into S after the swap, we must correct this by adding w_{ij} to the objective value. We repeat this procedure for j and obtain the potential benefit of swapping i and j. This benefit can be computed in $O(m)$ time.

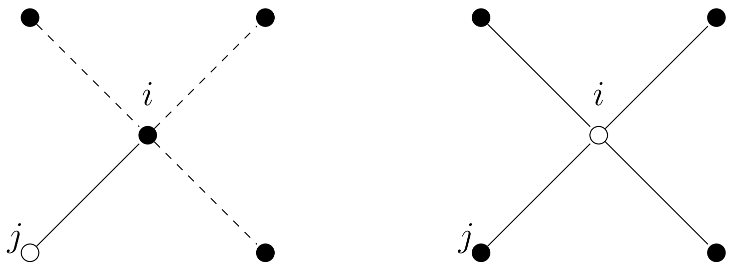

Figure 1: 2-swap procedure for Max Cut

4.3 Simulated Annealing

Simulated Annealing is a neighborhood based search method which is improving but occasionally allows switching to worse solutions in order to escape local minima. The motivation from this method came from the physical process of annealing metal, in which the metal is heated and cooled slowly in order to create a stronger material. The parameters for this algorithm include:

1. An initial temperature T_0, which is typically computed in relation to the worst possible neighboring solution

2. A temperature length T_L which represents how many neighboring solution we will check at each temperature

3. A cooling function which decreases the temperature for the next iteration

4. A stopping criteria, which in our case will be a threshold on temperature

At each iteration a random neighboring solution is selected, a potential 2-swap in this case, and the difference in objective between the current and new solution is computed. If the new solution is an improvement, it is always accepted. If the new solution is worse, then we accept the solution based on a probability function of the magnitude in the objective value change and the current temperature.

Algorithm 3: Simulated Annealing for Max Cut

```
Result: Feasible solution for Max Cut
Construct initial greedy solution $\mathbf{z}$ and select appropriate $\alpha, \Delta_{accept}, T_{stop}, T_L$;
$z_{best} \leftarrow z$;
$T_{now} \leftarrow -\frac{\Delta_{max}}{\log -\Delta_{accept}}$;
while $T_{now} > T_{stop}$ do
    for $t \in T_L$ do
        Select $i, j$ from opposite components and compute $\Delta(i, j)$;
        if $\Delta(i, j) > 0$ then
            Swap$(i, j)$;
            $z = z + \Delta(i, j)$
        else
            $q \leftarrow$ random$(0, 1)$;
            if $\exp\{\Delta(i, j)/T_{now}\} > q$ then
                Swap$(i, j)$;
                $z = z + \Delta(i, j)$
            end
        end
        if $z > z_{best}$ then
            $z_{best} \leftarrow z$
        end
    end
    $T_{now} \leftarrow \alpha T_{now}$;
end
```

5 Computational Study

In this section we present all computational results for each method discussed on 92 randomly generated weighted graphs. These instances consisted of thirty graphs with $n \in \{10, 15, 20\}$ and expected edge density $d \in \{0.2, 0.5, 0.8\}$, as well as a graph with $n = 50, d = 0.8$ and $n = 100, d = 0.2$. All algorithms were implemented in Python. Gurobi Optimizer 9.0 was used to solve all 0-1 programming formulations. All numerical experiments were conducted on a Linux Manjaro OS with Intel 2 Core i3-2100@1.6 GHz and 8-GB RAM.

5.1 Implementation Decisions

In this section we provide all of our implementation decisions and selected parameters for each method.

5.1.1 Integer Program

In this section we introduce the results of the integer program. Each instance was given a time limit of 600 seconds. All instances of such that $n \leq 20$ reached its optimal solution within this time where as the 50 and 100 node instance did not.

5.1.2 Lagrangian Dual Relaxation

In our computational study the selected sequence $h_k = M\rho^k$ with $M = 2n$ and $\rho = 0.98$ produced the overall best bound for most instances.

5.1.3 Ellipsoid Algorithm

For our implementation of the Ellipsoid algorithm we made several decisions regarding the initial solution and stopping criteria. Our initial solution $\mathbf{y}_k = \mathbf{0}$, in other words we begin with $\mathbf{Y}$ as the identity matrix. In order to select an initial R we consider the maximum distance between feasible solutions of Y. Consider the feasible solutions $Y_{ij} = 1$ for $i, j \in V$ and $Y_{ij} = -1$ for $i, j \in V$ such that $i \neq j$. The distance between these solutions can be computed as the element wise Euclidean distance. Since $\mathbf{Y} \in \mathbb{R}^{n \times n}$, the distance can be no larger than $2n$. As a result, we set $R = 2n$ for our algorithm.

Since the Ellipsoid algorithm can move to worse solutions throughout the algorithm we will select the $\|\mathbf{y}_{k+1} - \mathbf{y}_k\| < \epsilon$ as our stopping criteria. We set $\epsilon = 0.001$ as it provided us the best trade off between computation time and a feasible bound.

5.1.4 Simulated Annealing

In our implementation of Simulated Annealing we found that slower cooling rates did not help us find much better solutions except for larger instances of Max Cut. For our comparison with the IP solution we selected $\alpha = 0.90, T_L = 500$, and $\Delta_{accept} = 0.10 \cdot \Delta_{max}$.

In addition, we found that for many instances that we ran that the best solution was found fairly early on in the algorithm. We believe that our neighborhood selection was perhaps too restrictive for this problem. In future work we will attempt several other neighborhood representations.

5.2 Results

5.2.1 Upper Bound

Table 1 shows the average gap and solution time for each relaxation method discussed in this paper. We can see that there is an obvious trade off between relaxation quality and computation time. We see that the Ellipsoid algorithm produces substantially tighter bounds for maximum cut, however it is computationally expensive. In fact, the algorithm takes much longer to solve than the integer program. For larger instances, the Lagrangian Dual bound performed the best when considering quality and solution time.

Instance		LP Bound		LD Bound			Ellipsoid	
n	d	gap	time(s)	Eig gap	LD gap	time(s)	gap	time(s)
10	0.2	0.045	0.003	0.398	0.146	0.139	0.005	3.218
	0.5	0.203	0.003	0.283	0.093	0.151	0.018	3.498
	0.8	0.278	0.001	0.141	0.059	0.141	0.017	3.625
15	0.2	0.082	0.001	0.360	0.096	0.192	0.007	34.306
	0.5	0.253	0.001	0.250	0.079	0.202	0.024	34.740
	0.8	0.323	0.001	0.154	0.058	0.196	0.017	33.519
20	0.2	0.115	0.001	0.361	0.084	0.259	0.017	177.274
	0.5	0.283	0.002	0.248	0.072	0.259	0.029	144.599
	0.8	0.355	0.002	0.152	0.055	0.243	0.022	145.973
50	0.8	0.416	0.014	0.141	0.062	1.487	-	-
100	0.2	0.319	0.012	0.236	0.185	13.104	-	-

Table 1: Upper bounds for the maximum cut problem

5.2.2 Lower Bound

Table 2 shows that comparison for the heuristic solutions and the solutions obtained from our 0-1 integer programming formulation. We can see the heuristic is not very useful for smaller instances of Max Cut. However for larger instances it can quickly produce feasible solutions within comparable gap as the 0-1 formulation after 600 seconds of branch and bound.

n	d	Heuristic Gap	Heuristic Time	IP Gap	IP Time
10	0.2	9.04%	0.18	0.00%	0.01
	0.5	9.96%	0.27	0.00%	0.01
	0.8	7.63%	0.32	0.00%	0.02
15	0.2	5.63%	0.26	0.00%	0.01
	0.5	7.72%	0.41	0.00%	0.02
	0.8	5.63%	0.47	0.00%	0.09
20	0.2	9.01%	0.30	0.00%	0.01
	0.5	2.82%	0.50	0.00%	0.07
	0.8	4.77%	0.52	0.00%	0.91
50	0.8	11.12%	2.08	9.28%	600.01
100	0.2	5.16%	1.13	1.48%	600.01

Table 2: Heuristic and IP solutions for the maximum cut problem

6 Conclusion

In this work we provided several relaxations outside of the linear relaxation for the Maximum Cut problem as well as a simulated annealing heuristic for constructing good feasible solutions. In future work we would like to test more instances on larger graphs for which the IP formulation struggles more. In addition, we plan to revisit our implementation decisions to remove any inefficiencies and further tune the parameters. We are also interested in implementing this in an object-orientated language such as C++ to improve solution times.

All work for this project can be found on GitHub at `https://github.com/Dpapazaharias1/MaxCut`