MISG2022 Problem 4

Mathematical Modelling of the Max 2-Cut Problem and Solving the Relaxed Model

The team

Name	Contact email
Evan Rex	evangeorgerex@gmail.com
Stefany Bam	stefany49bam@gmail.com
Leago Mashishi	leagonmashishi@gmail.com
Johanna Matloa	jzmatloa@gmail.com
Gideon Ilung	Ilunggideon@gmail.com
Lwazi Zama	lwazi.zama26@gmail.com

Max-Cut

Problem

“The objective of Max-Cut is to partition the set of vertices of a (undirected) graph G = (V, E) into two subsets, such that the sum of the weights (cut value) of the edges having one endpoint in each of the subsets is maximised. This problem is known to be NP-Complete.”

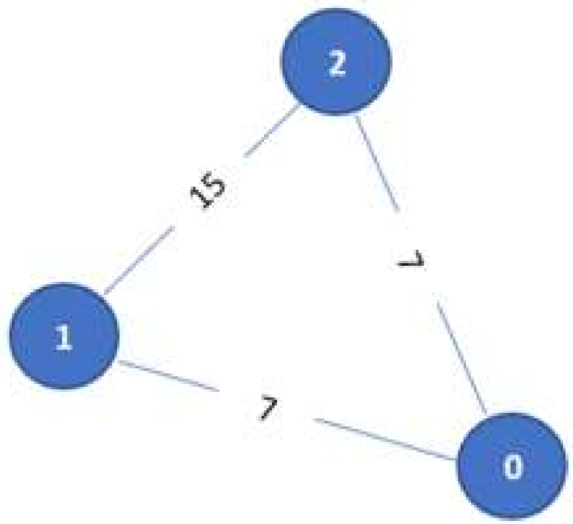
Solution

“The first step of converting the integer program into an SDP is known as relaxation. A relaxation of an optimization program is another optimization program which, ideally, is easier to solve and every solution of original program is also a solution of the new program with the same (or related) objective value. Our scheme is as follows: convert the integer program into a semi-definite program (SDP), and then solve.”

(Montaz, Ali, 2022)

Formulation of Max Cut

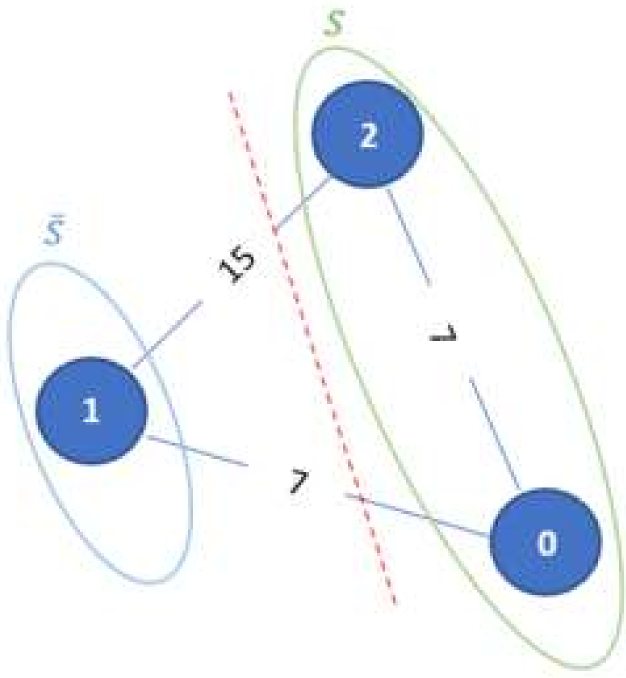
Consider the graph below:



With corresponding weight matrix:

0	7	7
7	0	15
7	15	0

The goal of the max cut problem is to partition the graph in such a way that the value of the edges that are ‘cut out ’ of the graph when partitioning is maximised, which ensures that similar points belong to the same set.



$$x_i = \begin{cases} 1 & x \in S \\ -1 & x \in \vec{S} \end{cases}$$

How do we determine if 2 points are in the same sets

Using the following equation we can determine if 2 points are in the same set or not

$$\frac{1 - x_i x_j}{2}$$

Examples

Points in same set

$$\frac{1-(-1)(-1)}{2}=0$$

$$\frac{1-(1)(1)}{2}=0$$

Points in different sets

$$\frac{1-(-1)(1)}{2}=1$$

$$\frac{1-(1)(-1)}{2}=1$$

Since the goal is to maximise the edges ‘cut out’ and we have a function that returns 1 if the points are in different sets

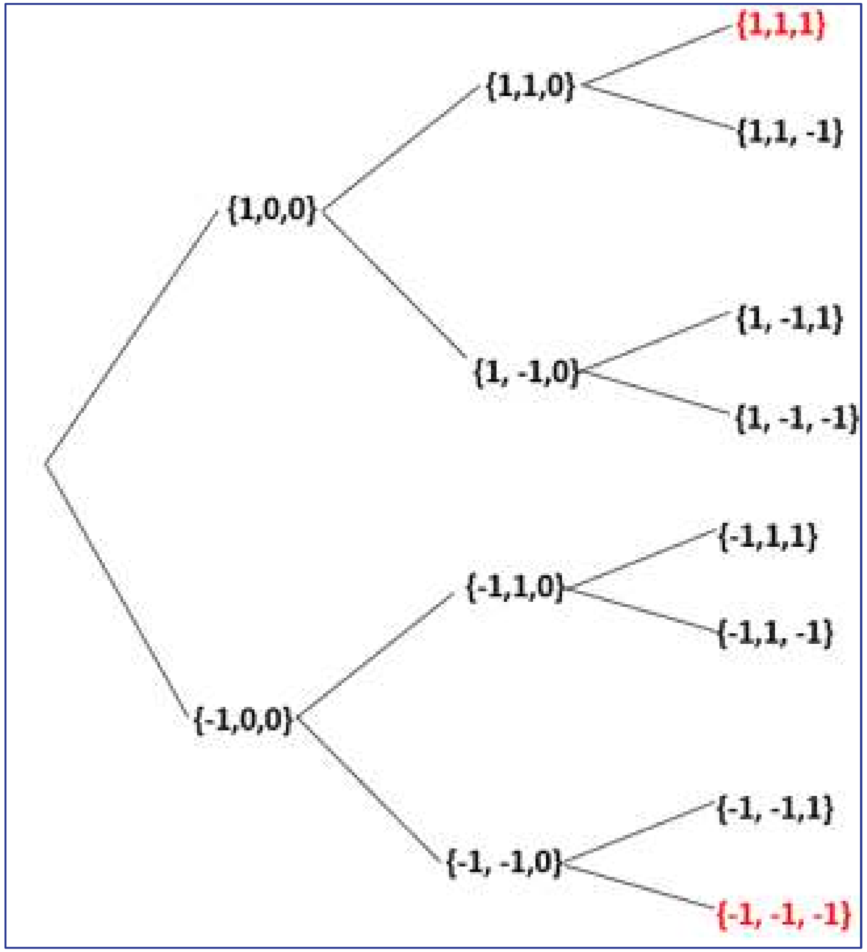
$$\sum_{i=0}^{n}\sum_{j=0}^{n} W_{ij}\frac{1-x_i x_j}{2}$$

Since graph is symmetric, i.e. $W_{ij} = W_{ji}$ The same edge will be considered twice

$$\frac{1}{2}\sum_{i=0}^{n}\sum_{j=0}^{n} W_{ij}\frac{1-x_i x_j}{2}$$

The Cost Function

$$\frac{1}{4}\sum_{i=0}^{n}\sum_{j=0}^{n} W_{ij}(1 - x_i x_j)$$



Dynamic programming Approach

Dynamic Programming is largely an optimization over plain recursion. In this case we find all possible combinations to divide the nodes into subset S and S bar.

Given the cost function, we would like to transform it into an SDP problem of the form:

max $C \cdot X$

s.t $A_i \cdot X = b_i \ \forall_i$

$$X \succeq 0$$

$$x_i x_j = 0 + 0 + .. + x_i x_j$$

$$= 0 \times 0 + 0 \times 0 + ... + x_i xj$$

$$= \begin{bmatrix} 0 & 0 & \dots & x_i \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dots \\ x_j \end{bmatrix}$$

$$= y_i^T y_j$$

Where $y_i = \begin{bmatrix} 0 & 0 & \dots & x_i \end{bmatrix}$

$$L_{ij} = \begin{cases} \sum_k^n W_{ik} & i == j \\ -W_{ij} & i \neq j \end{cases}$$

$$\begin{aligned} \frac{1}{4}\sum_{i=0}^{n}\sum_{j=0}^{n} W_{ij}(1 - x_i x_j) &= \frac{1}{4}\sum_{i=0}^{n}\sum_{j=0}^{n} W_{ij}(1 - X_{ij}) \\ &= \frac{1}{4}\sum_{i}\sum_{j} W_{ij} - \frac{1}{4}\sum_{i}\sum_{j} W_{ij}X_{ij} \\ &= \frac{1}{4}\sum_{i} Lii + \frac{1}{4}\sum_{i}\sum_{j}(-W_{ij})X_{ij} \end{aligned}$$

Since $x_i = -1 or x_i = 1$

$x_i^2 = 1$ **always**

$$\Rightarrow X_{ii} = 1$$

$$= \frac{1}{4}\sum_i L_{ii}(1) + \frac{1}{4}\sum_{i\neq j}\sum_{j\neq i} L_{ij}X_{ij}$$

$$= \frac{1}{4}\sum_i L_{ii}X_{ii} + \frac{1}{4}\sum_{i\neq j} L_{ij}X_{ij}$$

$$= \frac{1}{4}\langle L, X\rangle$$

$$= \frac{1}{4}(L \cdot X)$$

Hence the resulting SDP is as follows:

max $\frac{1}{4}(L \cdot X)$

s.t $X_{ii} = 1 \forall i$

$X \succeq 0$

Challenges deep-dive

Challenge 1	Challenge 2	Challenge 3
Understanding the maths behind the SDP Formulation	**Finding software to solve the SDP formulation**	**Using the software**

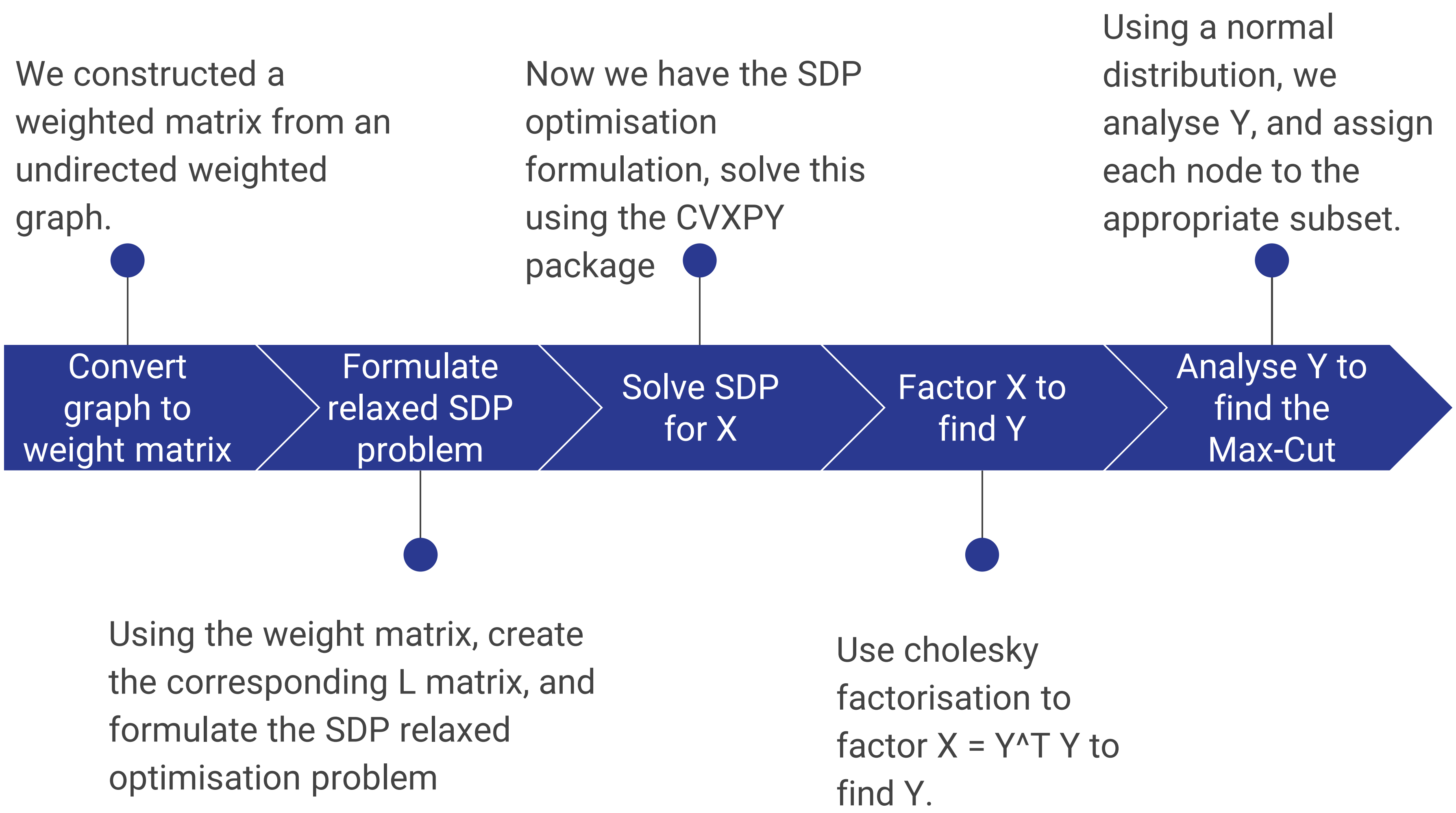
Solution

A python package - CVXPY

"CVXPY is an open source Python-embedded modeling language for convex optimization problems. It lets you express your problem in a natural way that follows the math, rather than in the restrictive standard form required by solvers." CVXPY (2022)

This package proved ideal for our team, due to our proficiency in Python over other coding languages.

Implementation


We constructed a weighted matrix from an undirected weighted graph.
Convert graph to weight matrix
Formulate relaxed SDP problem
Using the weight matrix, create the corresponding L matrix, and formulate the SDP relaxed optimisation problem
Now we have the SDP optimisation formulation, solve this using the CVXPY package
Solve SDP for X
Factor X to find Y
Use cholesky factorisation to factor X = Y^T Y to find Y.
Analyse Y to find the Max-Cut
Using a normal distribution, we analyse Y, and assign each node to the appropriate subset.

Semidefinite Relaxation Approach

Considering the following notation:

$$L_{ij} = \begin{cases} \sum_k w_{ik} & \text{if } i = j \\ -w_{ij} & \text{if } i \neq j \end{cases}$$

```python
def get_L(W,n):

    L = np.zeros((n,n))

    for i in range(0,n,1):
        for j in range(0,n,1):

            if i == j:
                L[i,i] = sum(W[i,:])
            else:
                L[i,j] = - W[i,j]

    return L
```

Semidefinite Relaxation Approach (Cont.)

```python
def semi_definite_solver(W,n):
    """
        Solves the max cut problem by transforming the problem
        into a semidefinite relaxation problem

        Input:
            W : the weight matrix
            n : the number of nodes in the graph
    """

    #constructing the L matrix#
    L = get_L(W,n)

    #defining the variable#
    X = cp.Variable((n,n), PSD = True)

    #getting constraints#
    constraints = []

    for i in range(0,n,1):
        constraints.append(X[i,i]==1)

    #cost function#
    expr = 0.25 * cp.trace(cp.multiply(L,X))
```

```python
#solving#
prob = cp.Problem(cp.Maximize(expr), constraints)
ans = prob.solve()
Y = np.linalg.cholesky(X.value)

#random matrix#
S = []
Sbar = []
W = np.random.normal(size=n)

for i in range(0,n,1):

    ans = np.dot(Y[:,i],W)

    if ans >=0:
        S.append(i)
    else:
        Sbar.append(i)

print(" S : ",S,"\n S': ",Sbar)
```

Semidefinite Relaxation Approach (Cont.)

Defining W:

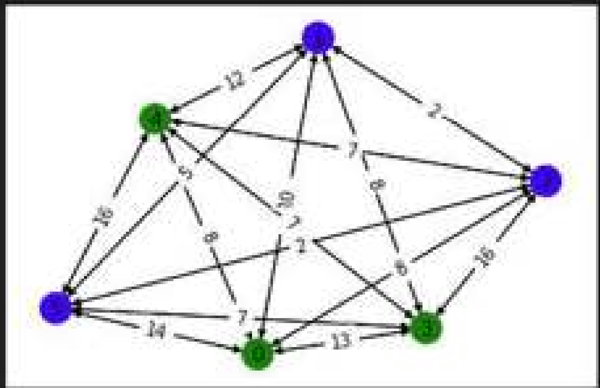
```
W = np.array([[0,4,10,-3,0,0,0],
              [4,0,0,0,6,0,0],
              [10,0,0,7,0,0,0],
              [-3,0,7,0,0,0,8],
              [0,6,0,0,0,4,0],
              [0,0,0,0,4,0,1],
              [0,0,0,8,0,1,0]])
```

The Max-Cut subsets for W:

```
S :  [1, 2, 5, 6]
S':  [0, 3, 4]
```

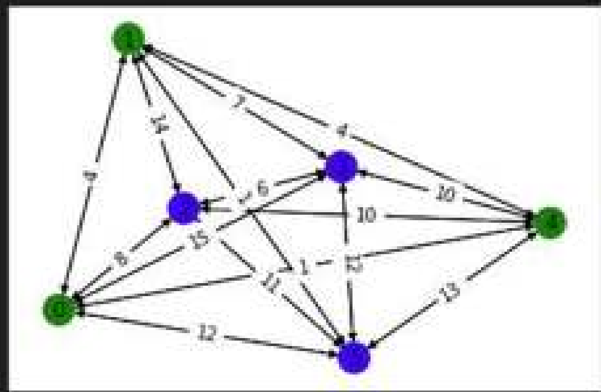
Graphs

```
Weight Matrix :
 [[ 0. 10.  8. 13.  8. 14.]
 [10.  0.  2.  8. 12.  5.]
 [ 8.  2.  0. 16.  7.  2.]
 [13.  8. 16.  0.  7.  7.]
 [ 8. 12.  7.  7.  0. 16.]
 [14.  5.  2.  7. 16.  0.]]
```



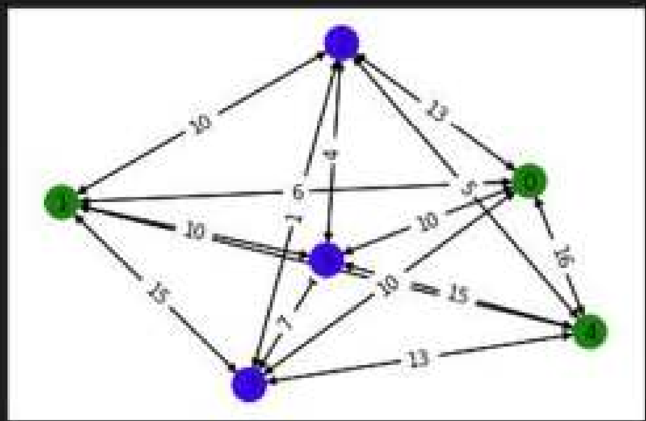
```
Cost:  98.0
 S :  [1, 2, 5]
 S':  [0, 3, 4]
```

```
Weight Matrix :
 [[ 0.  4.  8. 15.  1. 12.]
 [ 4.  0. 14.  7.  4.  1.]
 [ 8. 14.  0.  6. 10. 11.]
 [15.  7.  6.  0. 10. 12.]
 [ 1.  4. 10. 10.  0. 13.]
 [12.  1. 11. 12. 13.  0.]]
```



```
Cost:  90.0
 S :  [2, 3, 5]
 S':  [0, 1, 4]
```

```
Weight Matrix :
 [[ 0.  6. 13. 10. 16. 10.]
 [ 6.  0. 10. 15.  1. 10.]
 [13. 10.  0.  1.  5.  4.]
 [10. 15.  1.  0. 13.  7.]
 [16.  1.  5. 13.  0. 15.]
 [10. 10.  4.  7. 15.  0.]]
```




```
Cost:  101.0
 S :  [2, 3, 5]
 S':  [0, 1, 4]
```

Impact

Max-cut is known to be an NP-hard problem. Solving it with other methods, such as dynamic programming - is not feasible for large graphs. Using a relaxation approach allowed us to solve the problem a great deal faster.

References

Montaz, Ali (2022) *Mathematical Modelling of the Max 2-Cut Problem and Solving the Relaxed Model*

CVXPY (2022) *Welcome to CVXPY 1.1?* [online] available at https://www.cvxpy.org/ (Accessed February 4 2022)

Diamond S, Boyd S, 2016, *CVXPY: A Python-embedded modeling language for convex optimization*, Journal of Machine Learning Research volume 17, pp. 1-5

Agrawal, Akshay, Verschueren, Robin,Diamond, Steven,Boyd, Stephen, 2018, *A rewriting system for convex optimization problems*, Journal of Control and Decision volume 5, pp.42-60