The Max-Cut Problem

Adam Rahman

February 8, 2019

One of the simplest problems that can be formulated in terms of a conic linear optimization problem is finding the maximum cut of a graph. Let $\mathbf{G} = [\mathbf{V}, \mathbf{E}]$ be a graph with vertices $\mathbf{V}$ and edges $\mathbf{E}$. A *cut* of the graph $\mathbf{G}$ is a partition of the vertices of $\mathbf{G}$ into two disjoint subsets $\mathbf{G}_1 = [\mathbf{V}_1, \mathbf{E}_1]$, $\mathbf{G}_2 = [\mathbf{V}_2, \mathbf{E}_2]$, with $\mathbf{V}_1 \cap \mathbf{V}_2 = \varnothing$. The size of the cut is defined to be the number of edges connecting the two subsets. The *maximum cut* is defined to be the cut of a graph $\mathbf{G}$ whose size is at least as large as any other cut. For a weighted graph object, we can also define the maximum cut to be the cut with weight at least as large as any other cut.

Finding the maximum cut is referred to as the **Max-Cut Problem**, and was one of the first problems found to be NP-complete, and is also one of the 21 algorithms on Karp's 21 NP-complete problems ([2]). The Max-Cut problem is also known to be *APX hard* ([3]), meaning in addition to there being no polynomial time solution, there is also no polynomial time approximation.

Using the semidefinite programming approximation formulation of [1], the Max-Cut problem can be approximated to within an *approximation constant*. For a weighted adjacency matrix $\mathbf{B}$, the objective function can be stated as

$$\begin{array}{lrcl} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C}, \mathbf{X} \rangle & & \\ \text{subject to} & & & \\ & diag(\mathbf{X}) & = & \mathbf{1} \\ & \mathbf{X} & \in & \mathcal{S}^n \end{array}$$

where $\mathcal{S}^n$ is the cone of symmetric positive semidefinite matrices of size n, and $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. Here, we define $diag(\mathbf{a})$ for an $n \times 1$ vector $\mathbf{a}$ to be the diagonal matrix $\mathbf{A} = [A_{ij}]$ of size $n \times n$ with $A_{ii} = a_i, \quad i = 1, \dots, n$. For a matrix $\mathbf{X}$, $diag(\mathbf{X})$ extracts the diagonal elements from $\mathbf{X}$ and places them in a column-vector.

To see that the Max-Cut problem is a conic linear optimization problem it needs to be written in the same form as the primal objective function. The objective function is already in a form identical to that of the primal objective function, with minimization occurring over $\mathbf{X}$ of its inner product with a constant matrix $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. There are n equality constraints of the form $x_{kk} = 1, \quad k = 1, ..., n$, where x_{kk} is the k^{th} diagonal element of $\mathbf{X}$, and $b_k = 1$ in the primal objective function. To represent this in the form $\langle \mathbf{A}_k, \ \mathbf{X} \rangle = x_{kk}$, take $\mathbf{A}_k$ to be

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

Now $\langle \mathbf{A}_k, \ \mathbf{X} \rangle = vec(\mathbf{A}_k)^\intercal vec(\mathbf{X}) = x_{kk}$ as required, and the Max-Cut problem is specified as a conic linear optimization problem.

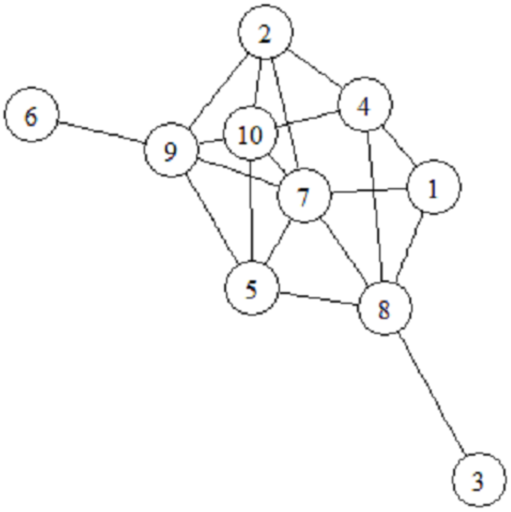
To convert this to a form usable by `sqlp`, we begin by noting that we have one optimization variable, $\mathbf{X}$. Therefore, with $L = 1$, and having $\mathbf{X}$ constrained to the space of semidefinite matrices of size n, we specify `blk` as

```
R> blk <- c("s" = n)
```

With the objective function in the form $\langle \mathbf{C}, \mathbf{X} \rangle$, we define the input `C` as

```
R> one <- matrix(1,nrow=n,ncol=1)
R> C <- -(diag(c(B %*% one)) - B)/4
```

where $\mathbf{B}$ is the adjacency matrix for a graph on which we would like to find the maximum cut, such as the one in Figure 1.



$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 1: *A graph object and associated adjacency matrix for which we would like to find the maximum cut.*

The matrix `At` is constructed using the upper triangular portion of the $\mathbf{A}_k$ matrices. To do this in `R`, the function `svec` is made available in `sdpt3r`.

```
R> #Construct Ak matrices
R> A <- matrix(list(),nrow=1,ncol=n)
R> for(k in 1:n){
R>   A[[k]] <- matrix(0,nrow=n,ncol=n)
R>   diag(A[[k]])[k] <- 1
R> }

R> #Combine to form At
R> At <- svec(blk[1],A,1)
```

Having each of the diagonal elements of $\mathbf{X}$ constrained to be 1, `b` is a $n \times 1$ matrix of ones

```
R> b <- matrix(1,nrow=n,ncol=1)
```

With all the input variables now defined, we can now call `sqlp` to solve the Max-Cut problem

```
R> sqlp(blk, list(At), list(C), b)
```

The built-in function `maxcut` takes as input a (weighted) adjacency matrix `B` and returns the maximum cut of the graph using `sqlp`. If we wish to find to the maximum cut of the graph in Figure 1, given the adjacency matrix $\mathbf{B}$ we can compute the solution using `sqlp` using `maxcut`

```
R> out <- maxcut(B)

R> out$pobj
```

```
[1] -14.67622

R> out$X

      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]  [,10]
V1   1.000  0.987 -0.136 -0.858  0.480  0.857 -0.879  0.136 -0.857  0.597
V2   0.987  1.000  0.026 -0.763  0.616  0.929 -0.791 -0.026 -0.929  0.459
V3  -0.136  0.026  1.000  0.626  0.804  0.394  0.592 -1.000 -0.394 -0.876
V4  -0.858 -0.763  0.626  1.000  0.039 -0.469  0.999 -0.626  0.470 -0.925
V5   0.480  0.616  0.804  0.039  1.000  0.864 -0.004 -0.804 -0.864 -0.417
V6   0.857  0.929  0.394 -0.469  0.864  1.000 -0.508 -0.394 -1.000  0.098
V7  -0.879 -0.791  0.592  0.999 -0.004 -0.508  1.000 -0.592  0.508 -0.907
V8   0.136 -0.026 -1.000 -0.626 -0.804 -0.394 -0.592  1.000  0.394  0.876
V9  -0.857 -0.929 -0.394  0.470 -0.864 -1.000  0.508  0.394  1.000 -0.098
V10  0.597  0.459 -0.876 -0.925 -0.417  0.098 -0.907  0.876 -0.098  1.000
```

Note that the value of the primary objective function is negative as we have defined $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$ since we require the primal formulation to be a minimization problem. The original formulation given in [1] frames the Max-Cut problem as a maximization problem with $\mathbf{C} = (diag(\mathbf{B1}) - \mathbf{B})/4$. Therefore, the approximate value of the maximum cut for the graph in Figure 1 is 14.68 (recall we are solving a relaxation).

As an interesting aside, we can show that the matrix $\mathbf{X}$ is actually a correlation matrix by considering its eigenvalues - we can see it clearly is symmetric, with unit diagonal and all elements in [-1,1].

```
R> eigen(X)$values

[1] 5.59e+00 4.41e+00 2.07e-07 1.08e-07 4.92e-08 3.62e-08 3.22e-08
[8] 1.90e-08 1.66e-08 9.38e-09
```

The fact that $\mathbf{X}$ is indeed a correlation matrix comes as no surprise. [1] show that the set of feasible solutions for the Max-Cut problem is in fact the set of correlation matrices. So while we may not be interested in $\mathbf{X}$ as an output for solving the Max-Cut problem, it is nonetheless interesting to see that it is in fact in the set of feasible solutions.

References

[1] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[2] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer-Verlag, 1972.

[3] Christos H Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, 43(3):425–440, 1991.