Recursive Approach toward QSAP

Bangchen Gong
bgong@oberlin.edu

Bill Huang
chuang@oberlin.edu

December 17, 2016

Abstract

In this paper, we are going to look at the quadratic semi-assignment problem (QSAP), and present an new algorithm that is more efficient in terms of time than the exact algorithm (simplex method with branch-and-bound and cutting planes) with recursive steps in solving the QSAP. For a real world problem that cluster 30 cities into 6 groups, it took our algorithm 1 second to get a solution that is within 9% error to the optimal solution, while it took Gurobi around 12 hours to solve the same QSAP by using the exact algorithm.

1 Introduction

The quadratic assignment problem (QAP) is one of the most difficult problems in the NP-hard class. After decades of study, there is still no algorithm for solving it in polynomial time. Even a small size QAP as compared to other integer program problems takes a great amount of time to get the optimal solution by exact algorithm (branch-and-bound, cutting planes, or the combinations of both). Loiola et al. (2007) [1] present a excellent survey for different formulations of the quadratic assignment problems and discuss about the progress made in both exact and heuristic solution methods. In section 2, we will show the LP systems with and without linearization for the QSAP. In section 3 and 4, we will introduce our algorithm, and apply it to one real life problem to discuss the performance of it. We will summarize and draw conclusions in Section 5.

2 Quadratic Semi-Assignment Problem

The quadratic semi-assignment problem (QSAP) is used for modeling cluster and partition. The QSAP we will address here is a cluster problem. Given a traveling cost matrix C, in which entry $c_{i,j}$ is the traveling cost from city i to city j, we want to separate n cities evenly into k divisions, each of which contains exactly m ($m = \frac{n}{k}$, integer) cities, so that the total intradivisional traveling cost across all the clusters is minimal [1] . We assume the cost is symmetry here: $\forall i, j : c_{i,j} = c_{j,i}$, and the distance from one city to itself is 0: $\forall i : c_{i,i} = 0$. Let

$$x_{i,d} = \begin{cases} 1 & \text{if city i is in cluster d} \\ 0 & \text{if not} \end{cases}$$

Then the formulation for the problem is the following:

$$\begin{aligned} \text{Minimize} \quad & \sum_{d=1}^{k} \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{i,j} x_{i,d} x_{j,d} & \\ \text{Subject To} \quad & \sum_{d=1}^{k} x_{i,d} = 1 & \forall\ i \\ & \sum_{i=1}^{n} x_{i,d} = m & \forall\ d \\ & x_{i,d} \in \{0, 1\} & \forall\ i, d \end{aligned}$$

[1] We only consider travels between cities in the same division and the trip between every city pair could only be made once.

However, the objective function above is not linear. To linearize it, we need a set of new variable $y_{i,j,d}$ where

$$y_{i,j,d} = \begin{cases} 1 & \text{if both city } i,\ j \text{ in division d} \\ 0 & \text{if not} \end{cases}$$

and the linearized version of the integer program is the below

$$\begin{aligned}
\text{Minimize} \quad & \sum_{d=1}^{k} \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{i,j} y_{i,j,d} & \\
\text{Subject To} \quad & \sum_{d=1}^{k} x_{i,d} = 1 & \forall\ i \\
& \sum_{i=1}^{n} x_{i,d} = m & \forall\ d \\
& x_{i,d} + x_{j,d} \leq 1 + y_{i,j,d} & \forall\ i, j, d \\
& y_{i,j,d} \leq x_{i,d} & \forall\ i, j, d \\
& y_{i,j,d} \leq x_{j,d} & \forall\ i, j, d \\
& x_{i,d},\ y_{i,j,d} \in \{0, 1\} & \forall\ i, j, d
\end{aligned}$$

In the linearization step, we introduce additional $d \times \binom{n}{2}$ binary variables and $3d \times \binom{n}{2}$ linking constraints into the formulation. Suppose we are dealing a medium size problem with 30 cities and 6 divisions, the original formulation will have 180 binary variables and 36 constraints. By doing linearization, we introduce another 2610 binary variables and 7830 constraints, which make the numbers of new system rise up to 2790 (variables) and 7866 (constraints), and pump the size of matrix A in the LP relaxation step into 7866×10656. Moreover, since there are 2790 binary variables, we can expect that it would cost considerable amount of time to do branch-and-bound. But this is a problem for only 30 cities! Clearly, with gigantic number of variables and constraints in the linearized formulation, simplex method with branch-and-bound and cutting plane is inefficient in terms of time. We want a more time-saving algorithm. After brainstorming, we failed to make it happen without any error. So we determined that for our algorithm, a little sacrifice of optimality is acceptable.

3 Recursive Algorithm

3.1 The Block Diagonal Structure

Suppose we want to separate the nine cities (shown in Figure 1) into three groups and the cost for traveling between two cities is their distance. We can easily figure out a

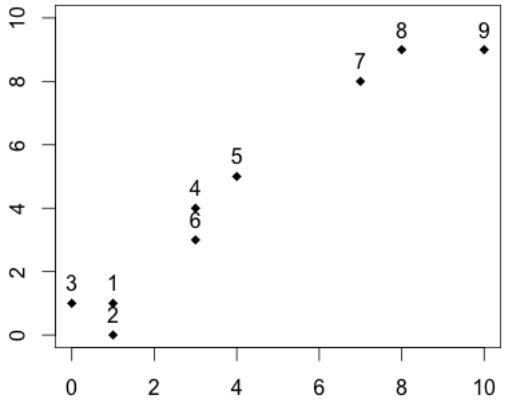


Figure 1: Separate 9 cities into 3 clusters

proper strategy by simply looking into the plot. Suppose group 1 contains cities 1, 2, 3; group 2 contains cities 4,5,6; and group 3 contains cities 7,8,9. Now let us construct the optimal solution matrix Y where $y_{i,j} = 1$ if $y_{i,j,d} = 1$ for any d from the optimal solution and otherwise 0. In this case, we have the following matrix.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Here, we can clearly see a block diagonal structure. Actually, not only this particular problem, but also other QSAP's have the same characteristic. For most of the time, however, it is not a easy job to recognize the block diagonal structure right after constructing the optimal solution matrix Y. We need to do some symmetrical row and column permutations to turn the original matrix into the special block diagonal one matrix, as shown below.

$$\begin{bmatrix} W & 0 & \cdots & 0 \\ 0 & W & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W \end{bmatrix}$$

where W is a $m \times m$ matrix with all entries being 1. The key of our algorithm is to search for a re-permuted block diagonal matrix with the least nontrivial entries, rather than approach the optimal solution purely in a algebraic way.

3.2 Elimination Step

Recall that our problem is to separate n cities evenly into k divisions, and our goal is to minimize the total intradivisional traveling cost. If the traveling cost going from city i to city j is unpleasantly large, it is very likely that the two cities will not be seated in the same group in the optimal solution, i.e., $\forall d : y_{i,j,d} = 0$, and in the optimal solution matrix Y, entry $y_{i,j}$ and $y_{j,i}$ will show up as 0. By repeating the above step for some biggest costs in the traveling cost matrix C, we are eliminating some of the potential clusters that are not likely to end up in the optimal solution. The basic elimination step can be summarized as

- Start with a $n \times n$ potential solution matrix Y' where all the entries of Y' are 1.
- Set a threshold T. If the traveling cost between cities i and j is greater than T, set entry $y'_{i,j}$ equal to 0. For traveling cost between cities i and j that is less or equal to T, we keep $y'_{i,j}$ as 1.

The best we can do is to set threshold T equal to 0 so that we will eliminate all the costs except for the cost of going from one city to itself, which is essentially 0. This is the case that when $n = k$, and the objective value is 0. However, for $k \neq n$, it takes some amount of work to find a proper T such that the matrix after threshold elimination is still feasible to the system. In other word, we want it possible that the reduced potential solution matrix could turn into the following matrix by doing some symmetrical row and column permutations:

$$\begin{bmatrix} W & G & \cdots & G \\ G & W & \cdots & G \\ \vdots & \vdots & \ddots & \vdots \\ G & G & \cdots & W \end{bmatrix}$$

where W is again a $m \times m$ matrix with all entries being 1 and G can be any $m \times m$ matrix. If we do not see W on the diagonal, we would definitely not able to provide all k feasible groups here, and we are violating the cluster constraint by applying a threshold T that is too intense to use. Hence, the full algorithm for the elimination step is

- Start with a $n \times n$ potential solution matrix Y' where all the entries of Y' are 1.
- Set an arbitrary large threshold T (e.g. the largest traveling cost across all cities), set $y'_{i,j}$ equal to 0 for cities i and j with travelling cost greater than T.
- Permute the rows and columns by using the algorithm from Ciprian Zavorianu [2] and check whether the permuted potential matrix maintain the cluster constraints.
- Decrease the threshold T gradually and repeat the above steps up to a point that by eliminating one more travelling cost, we will violate the cluster constraints.

Below is the cost matrix for the cities in Figure 1 (recalled that cost is distance in this example),

$$\begin{bmatrix} 0.00 & 1.00 & 1.00 & 3.61 & 5.00 & 2.83 & 9.22 & 10.63 & 12.04 \\ 1.00 & 0.00 & 1.41 & 4.47 & 5.83 & 3.61 & 10.00 & 11.40 & 12.73 \\ 1.00 & 1.41 & 0.00 & 4.24 & 5.66 & 3.61 & 9.90 & 11.31 & 12.81 \\ 3.61 & 4.47 & 4.24 & 0.00 & 1.41 & 1.00 & 5.66 & 7.07 & 8.60 \\ 5.00 & 5.83 & 5.66 & 1.41 & 0.00 & 2.24 & 4.24 & 5.66 & 7.21 \\ 2.83 & 3.61 & 3.61 & 1.00 & 2.24 & 0.00 & 6.40 & 7.81 & 9.22 \\ 9.22 & 10.00 & 9.90 & 5.66 & 4.24 & 6.40 & 0.00 & 1.41 & 3.16 \\ 10.63 & 11.40 & 11.31 & 7.07 & 5.66 & 7.81 & 1.41 & 0.00 & 2.00 \\ 12.04 & 12.73 & 12.81 & 8.60 & 7.21 & 9.22 & 3.16 & 2.00 & 0.00 \end{bmatrix}$$

By using the elimination step, we will get the smallest threshold T equal to 3.21, and the T will give us the potential solution matrix Y' shown in Section 3.1. Since Y' has the same structure as the optimal solution matrix Y in this case, the clustering shown in Y' is the optimal solution for this problem.

Lemma 1. *If there exists a threshold T that can bring out a eliminated cost matrix Y' that could be turned into a strict block diagonal matrix Y'^p after some symmetrical row and column permutations, the clusters shown in Y'^p is optimal.*

Proof. (by contradiction) Denote the graph we get directly from block diagonal matrix as G_1. Assume there exists another well-clustered graph G_2 such that the value of its objective function is smaller than that of G_1. Since $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are formed for the same problem, we have:

1) $V_1 = V_2$

2) $|E_1| = |E_2|$

3) There exists a biggest edge set C_0 such that $C_0 \subset E_1$ and $C_0 \subset E_2$. (In some cases, C_0 is empty.)

Call C_0's complement in each set C_1 and C_2, i.e., $E_1 - C_1 = E_2 - C_2 = C_0$. With 2) and 3) above, we have $|C_1| = |C_2|$, which means we could generate G_2 from G_1 by remaining E_0 and re-arranging every edge in C_1 to any non-occupied edge slot offered in C_2, and vice versa. Note that G_1 is the graph we get from block diagonal matrix, we know $\forall e \in E_1 : e \leq T$ and $\forall e \in E^c : e > T$, indicating that every re-arrange operation on edge in E_1 would increase the value of its objective function so that the new graph G_2 we get is certainly not as optimal as G_1. Thus, G_1 is the optimal graph. □

The elimination step is much more efficient for solving the well-clustered cost data QSAP than the simplex method with branch-and-bound and cutting planes. However, in most cases, we would not find a threshold T as perfect as expected.

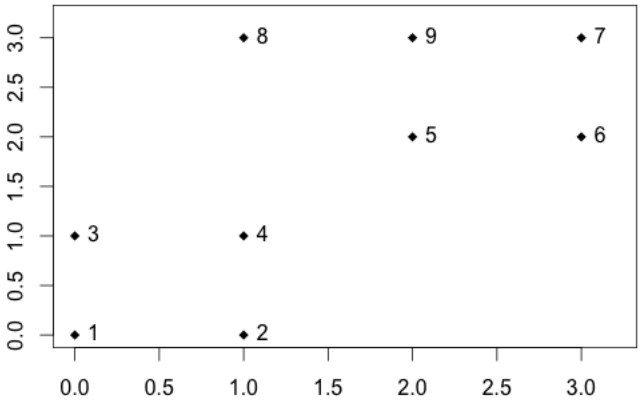


Figure 2: Separate 9 cities into 3 clusters

Consider a problem of separating the nine cities in Figure 2 evenly into three groups, and the cost matrix associated with it is:

$$\begin{bmatrix} 0.00 & 1.00 & 1.00 & 1.41 & 2.83 & 3.61 & 4.24 & 3.16 & 3.61 \\ 1.00 & 0.00 & 1.41 & 1.00 & 2.24 & 2.83 & 3.61 & 3.00 & 3.16 \\ 1.00 & 1.41 & 0.00 & 1.00 & 2.24 & 3.16 & 3.61 & 2.24 & 2.83 \\ 1.41 & 1.00 & 1.00 & 0.00 & 1.41 & 2.24 & 2.83 & 2.00 & 2.24 \\ 2.83 & 2.24 & 2.24 & 1.41 & 0.00 & 1.00 & 1.41 & 1.41 & 1.00 \\ 3.61 & 2.83 & 3.16 & 2.24 & 1.00 & 0.00 & 1.00 & 2.24 & 1.41 \\ 4.24 & 3.61 & 3.61 & 2.83 & 1.41 & 1.00 & 0.00 & 2.00 & 1.00 \\ 3.16 & 3.00 & 2.24 & 2.00 & 1.41 & 2.24 & 2.00 & 0.00 & 1.00 \\ 3.61 & 3.16 & 2.83 & 2.24 & 1.00 & 1.41 & 1.00 & 1.00 & 0.00 \end{bmatrix}$$

Following the elimination step, the smallest threshold T is 2.00, and the potential solution matrix Y' is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Now we permute the rows and the columns in the following order: 6 9 7 5 8 4 3 2 1, which the original row 6 permutes to row 1 and column 6 permutes to column 1. Then

the permuted potential solution matrix Y'^p we get is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

However, given that the permuted potential solution matrix does not possess the special block diagonal structure of the optimal solution matrix Y, we cannot get the optimal clustering directly from the above matrix. Hence, we need to employ the "local" minimizing step to get a solution that is very closed to the absolute optimal.

3.3 "Local" Minimizing Step

From the above permuted potential solution matrix Y'^p,

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

we can get one potential clustering: Group 1 contains city 6,9,7; Group 2 contains city 5,8,4; and Group 3 contains city 3,2,1. This might not be the optimal due to the following two reasons:

- We can permute the rows and columns in a different way that still maintains the block diagonal property. The clustering from the new permutation might be associated with a lower total cost.
- The red region in the above matrix has entries of all 1, meaning that the cost of traveling between any two cities in the set of cities 4, 3, 2, 1 is small. We do not know which three should be in the optimal cluster.

In this question, we might find out putting cities 3, 2, 1 together is premier after a little amount of computation. However, the issue becomes more complicated when n get larger. One way to get around the issue is to minimize the "local" cluster. Continue from the

above, There are two options for us to select three cities out of cities 4, 3, 2, 1 to form a group in the final solution:

- Find three cities out of the four that have a smallest total cost. From Figure 2, this approach might end up having cities 1, 3, 4 be in one cluster, and city 2 need to be in a cluster with other cities that are far away. This is not good.
- Consider the two potential clusters of cities that include cities 4, 3, 2, 1. In this case, it is the following 6 cities: 5 8 4 3 2 1. Now we will use the simplex method with branch-and-bound and cutting planes to separate the six cities into two clusters [2]. Then we take the cluster that contains city 1, the last city in the permutation, to be one of the clusters in the final solution. This approach will avoid the issue of over-fitting for one cluster.

In our algorithm, we will go with the second approach, and we call it the "local" minimizing step. To be more general, for a problem that separate n cities into k clusters that each contains m cities and the resulting permuted potential solution matrix Y'^p after the elimination step does not possess the block diagonal structure of the optimal matrix solution Y, the "local" minimizing step will do the following:

- Find the smallest number a that the a^{th} entry in the last row of Y'^p is 1 [3].
- Let
$$l = \left\lceil \frac{n+1-a}{m} \right\rceil$$
be the number of potential clusters we are considering [4]. Then we use the exact algorithm to separate the last $m \times l$ cities in the order of permutation (from the elimination step) into l clusters.
- The cluster that contains the last city in the order of permutation will be the cluster that we select for being one of the cluster in the final solution of our algorithm.

3.4 The Full Algorithm

Presented in Algorithm 1, our recursive algorithm strives for minimizing local cluster while maintaining a global feasible solution that is not terribly deviated from the true optimal solution given that we eliminate some large costs along the process.

[2] Gurobi uses the simplex method with branch-and-bound and cutting plane and it is efficient in handling some very small size QSAP's.

[3] We start from the last row/column due to the design of the permutation algorithm. Cities that are far away from the many others are the several cities at the end of the permutation order.

[4] For example, if we separate 30 cities into 6 clusters, and we observe the first entry with value 1 in the last row of Y'^p is in the (20, 30) entry, then the l will be 3 in this case. Generally, most of the time we only run into l being 2 due to the design of the algorithm we use for permuting the matrix.

Algorithm 1: Recursive Algorithm

Data: Cost Matrix C, Number of Cities n, Number of Clusters k
Result: Optimal Clusters that Minimize total Intradivisional Cost
Initialization;
while *C is not empty (0×0 matrix)* **do**
    Do elimination step and get Y'^p;
    if $Y'^p = Y$ **then**
        Store optimal clusters for all cities in C;
        Break;
    else
        Do "local" minimizing step and store the selected cluster;
        Delete rows and columns for the cities in the selected cluster from C;
    end
end

For the question in Figure 2, our algorithm gives the same optimal solution that the Gurobi gives.

4 Performance in Real Life Problem

In National Basketball Association (NBA), there are 6 divisions and each has 5 teams. For team that are in the same division, they have to play with each other four times in a regular season [5]. Given that the traveling might hurt the players' performance on the game, we want to find a optimal way to separate the teams into six divisions that the total intradivisional traveling cost can be minimized. This is the quadratic semi-assignment clustering problem with $n = 30$ and $k = 6$. For simplicity, we assume the traveling cost is the distance between the two cities that the two teams locate in.

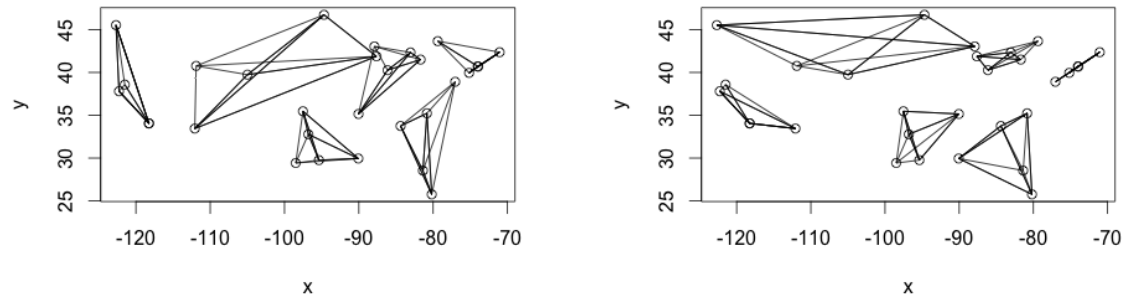
By using Gurobi which use the simplex method with branch-and-bound and cutting plane exact algorithm, it took about 12 hours to get the optimal solution of 29792 [6]. By using our recursive algorithm, we got a solution of 32488 within 1 seconds on a 2013-model Macbook Air. Though our solution is 9% larger than the optimal, our recursive algorithm does express great advantage in term of time. Moreover, it also took Gurobi more than half hour to get a bound that is lower than the solution we got from our algorithm.

Figure 3 [7] visualizes the solution conducted from our algorithm and Gurobi. We can see

[5] They generally play three times with teams in the same conference (East and West) but not in the same division, and twice with teams in different conference in the regular season.

[6] The first 10 hours of the IP is ran on a 2013-model Macbook Air and the last 2 hours is ran on a 2015-model New Macbook.

[7] There are two NBA teams in Los Angeles, so they are overlapped in the plot. There are one team in Manhattan and one in Brooklyn. They are also shown as overlap in the plot.



(a) Recursive Algorithm (b) Optimal

Figure 3: NBA Clusters

that in the optimal solution, one large cost (long distance) is kept in the optimal clusters. However, our algorithm is designed to eliminate the long distance in every recursive step if possible, which partially explains why our algorithm cannot reach the optimal solution in this particular problem and some other cases.

5 Conclusion

In this paper we discuss one special type of the quadratic assignment problem, the quadratic semi-assignment problem, and present a recursive algorithm that is much more efficient in time than the exact algorithm, simplex method with branch-and-bound and cutting plane. In the NBA clustering example, the error of our solution is within 9%, which is tolerable given that it only took a common computer about 1 second to figure it out. Though our solution is not optimal, we can use it as the starting point for the exact algorithm or the other heuristic algorithm such as Ant Colony.

Also, we noticed that in global optimal solution, some large cost (long distance) are kept, while our algorithm is trying to get rid of the long distance at all costs. This provides us an idea to improve our algorithm by including some randomness. In the algorithm presented in the paper, we eliminate a cost c if it is greater than threshold T under the condition that the cluster constraints is still maintained. We can improve this step by introducing a new variable u that follows a uniform distribution between 0 and 1. If $uc < T$, then we will not eliminate the large cost c.

Given that our algorithm is efficient in dealing with QSAP, our future approach is to modify our algorithm so that it can also be used in solving the general quadratic assignment problem. However, general QAP with flow and distance matrix is in its nature very different from the QSAP, which provides an even bigger and more complicated playground for us.

References

[1] Eliane Maria Loiola et al. “A survey for the quadratic assignment problem”. In: *European journal of operational research* 176.2 (2007), pp. 657–690.

[2] Ciprian Zavoianu. *Tutorial: Bandwidth reduction - The CutHill-McKee Algorithm.* `http://ciprian-zavoianu.blogspot.com/2009/01/project-bandwidth-reduction.html`. Accessed: 2016-12-17.