A Comparison of Mixed Integer Linear Programming Approaches to the K-Center Problem

Adhish Kancharla

October 2023

1 Abstract

The K-Center problem is an NP-Hard combinatorial optimization problem [1] with real world applications previously explored [2] and described in this paper. We give an overview of various linear programming approaches to this problem and analyze their experimental run-times. Additionally, we cite a few heuristic and approximation algorithms which are practically quite efficient and should be researched further.
Section 3 presents the graph theoretic formulation for the practical applications of the K-center problem in electric cars and online grocery.
There have been many previous linear programming approaches to this problem. Section 4 notes the standard integer LP formulations as well as heuristics to make them faster in practice.
Sections 5 and 6 compare the experimental performances of the standard LP and another LP with heuristics.

2 Introduction

I first came across a simplification of the problem in my country's IOI (International Olympiad in Informatics) training camp. We were given a set of possible locations to place charging points for electric cars and we needed to choose one of them so that the maximum distance of a car to its nearest charging point was minimized.
I thought about how this problem could be extended to a place a fixed number of centers such that the maximum distance of a car to its nearest center was minimized.
I discovered that this was called the K-center problem, an NP hard problem in theoretical computer science.
The K-center problem can be reduced from the dominating set problem as described in [3] and [4]

This shows the NP hard nature of the K-center problem, making exact solutions theoretically impossible to attain for large graphs. However, many approximation algorithms and heuristics have been previously researched. Here, we consider the mixed integer linear programming approaches to the problem.

3 Possible Applications

The application of the k-center problem in social network graphs, used for maximizing the spread of influence or minimizing the dissemination time, has been explored in [2]

3.1 Electric cars

With the arrival of electric cars, everyone needs to find their nearest electric charging point. However, there are a limited number of charging stations that can be placed. We can mathematically formulate this optimization problem as the K-center problem.
Let the vertices of the graph be all possible charging points located at the junctions of roads. Let the edges of the graph be the direct roads connecting the various junctions. The edge weights represent the time taken to go from one junction to an adjacent one. Now, solving the K-center problem on this un-directed, weighted graph will give us the optimal location for the K charging points.

3.2 Online grocery

Another possible application is in the field of online grocery. Companies would like to choose their warehouses such that the maximum distance of a client from its closest inventory is minimized.
Let the vertices of the graph be all possible locations for the inventory. Let the edges be the roads connecting the various physical locations. The edge weights represent the time it takes to go from one location to its neighbouring location. Now, solving the K-center problem on this un-directed, weighted graph will give us the optimal location for the K inventories.

3.3 Healthcare access

In regions with limited access to healthcare facilities, the k-center problem can be applied to determine the optimal locations for medical clinics and vaccination centers. This ensures that healthcare units are distributed efficiently, improving overall health outcomes.

4 Linear Programming Formulations

Consider an un-directed, weighted graph with nodes U. Given a node u in U and a subset V of U, consider the distance $d(u, V)$ = $\min \{d(u, v)|\ v$ in $V\}$
The objective is to find the the subset V of U with cardinality K that minimizes $\max \{d(u, V) \mid u$ in $U\}$

4.1 Standard Integer Linear Programming

The classical integer linear programming formulation is given below:

Define decision variables:
$x_{i,j} = 1$ if client i is assigned to facility j, otherwise $x_{i,j} = 0$
$y_j = 1$ facility j is open, otherwise $y_j = 0$

Minimize z
subject to

$$\sum_{j=1}^{n} x_{i,j} = 1 \forall i \in V \tag{1}$$

$$x_{i,j} \leq y_j \forall i, j \in V \tag{2}$$

$$\sum_{j=1}^{n} y_j \leq k \tag{3}$$

$$\sum_{j=1}^{n} d_{i,j} x_{i,j} \leq z \forall i \in V \tag{4}$$

$$x_{i,j}, y_i \in 0, 1 \forall i, j \in V \tag{5}$$

Constraint (1) ensures that each client is assigned to exactly one facility.
Constraint (2) ensures that clients are assigned to only open facilities.
Constraint (3) ensures that at most k facilities are opened.
Constraint (4) represents the sum of distances to open facilities for each client
Constraint (5) is the integer programming constraint which ensures that each facility is either fully opened or fully closed.

4.2 Another Integer LP

Another linear programming formulation is highlighted below [5]:

We solve the k-center problem by solving a series of set-covering problems. Each time choose a threshold distance as radius and check whether all clients can be covered within this radius using no more than k facilities. This function is monotonically

increasing, because if any radius can cover k facilities, so can a larger radius. Thus, binary search is applicable.

The feasibility problem for a specific radius [5]:

Define decision variable:
$w_i = 1$ if facility i is open, otherwise 0

$$\sum_{j=1}^{n} b_{i,j} w_j \geq 1 \; \forall i \in V \text{ (1)}$$

$$\sum_{j=1}^{n} w_j \leq k \text{ (2)}$$

where $b_{i,j} = 1$ if $d_{i,j} \leq r$

Constraint (1) ensures that each location is assigned at least 1 facility
Constraint (2) is the integrality condition which ensures each facility is either open or not.

4.3 Two Heuristics

To speed up the practical run-time of this formulations, two heuristics have been discovered [6]:

Modification 1:
Select the lower-bound as the $k - th$ minimum of the distance in the distance matrix. This guarantees the optimal covering radius is $\geq LB$
Select the upper-bound as the minimum from the set of maximum distance values in each row. This guarantees the optimal covering radius is $\leq UB$

Modification 2:
Use the golden section method instead of a binary search. Instead of binary searching on an interval $[a, b]$, use two interior points:
$x_1 = a + (b - a)/(\lambda)^2$
$x_2 = a + (b - a)/(\lambda)$
where λ is the golden ratio approximately equal to 0.618
The modifcation uses x_1 and x_2 as the converge distance of the set covering problem.

4.4 Summary of the three algorithms

We first compute the shortest path between every pairs of nodes as it is required in all three algorithms.
Algorithm 1: Call the standard integer linear program with the matrix of distances
Algorithm 2: Binary search on the minimum feasible radius using the range of radii obtained from the first modification
Algorithm 3: Golden search using the second modification on the range of radii obtained from the first modification

We hypothesize that algorithms 2 and 3 would take lesser time than the first algorithm, although they might use more memory.
Algorithm 3 might take lesser time when the upper bound is large since the golden search might check lesser number of radii than a binary search

5 Experimental Methods and Results

We generate random undirected connected graphs with a specified number of vertices (n) and a specified number of edges (m). To do so, we first generate a random spanning tree of $n-1$ edges and then we add $m-n+1$ random edges. The weights of edges lie between $[0, UB]$ where UB is the maximum edge weight that is varied in our experiments.
Then, we find distances between every pair of nodes using Dijkstra's shortest path algorithm, and run the three algorithms for different values of k.
The table on the last page shows the results for 15 experiments.

6 Discussion

As we can see, algorithms 2 and 3 seems to take lesser time than algorithm 1 on these data. This could be because the second linear programming formulation has considerably less number of variables, resulting in lesser operations done by the branch and bound. Additionally, the JuMP API remembers each previous LP solution and the radius modifications. This further reduces the time for the operations in the second LP.
The rows where "TIMED OUT" is written indicate instances where algorithm 1 took more than a few minutes to run. For these data, algorithm 1 is inefficient in practice.
Although the binary search of algorithm 2 seems to query less radii for small weights, the golden search in algorithm 3 seems to take less time for larger weights.
In conclusion, we see that for large graphs with high weights, algorithm 3 seems to work fast; for large graphs with low weights, algorithm 2 works fast and for small graphs algorithm 1 uses the least memory while executing within a reasonable time.

7 Conclusion and Further Research

In this paper, we analyzed three algorithms that found the exact solution to the uncapacitated vertex K-center problem.
We discovered that the golden search method works best for graphs whose edges have high weights, whereas binary search is better for graphs with edges of lesser weights.
In graphs of small sizes, the standard integer linear program works fast enough and uses the least amount of memory, whereas for graphs of larger sizes, the binary/golden search methods should be used.
It would be interesting to analyze the capacitated version of the K-center problem, where there is a limit on the number of locations that can be assigned to a facility. Perhaps the same three algorithms could be used with N more linear constraints for the capacities.

8 Acknowledgements

I would like to thank professor Benoit Legat, who guided me as a part of his course on the Applications of Mathematical Optimization: Mixed Integer Linear Programming at the Cambridge Center for International Research. It was through his lectures that I learned of the mathematical techniques to approach the K-center problem.

References

[1] "course notes," 2007. [Online]. Available: https://algo2.iti.kit.edu/vanstee/courses/kcenter.pdf

[2] M. Nehéz, "On the k-center problem in social networks," 09 2014.

[3] S. Vishwanathan, "An o (log* n) approximation algorithm for the asymmetric p-center problem," in *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, 1996, pp. 1–5.

[4] J. Garcia-Diaz, R. Menchaca-Mendez, R. Menchaca-Mendez, S. P. Hernández, J. C. Pérez-Sansalvador, and N. Lakouari, "Approximation algorithms for the vertex k-center problem: Survey and experimental evaluation," *IEEE Access*, vol. 7, pp. 109 228–109 245, 2019.

[5] J. Garcia-Diaz, R. Menchaca-Mendez, R. Menchaca-Mendez, S. Pomares Hernández, J. C. Pérez-Sansalvador, and N. Lakouari, "Approximation algorithms for the vertex k-center problem: Survey and experimental evaluation," *IEEE Access*, vol. 7, pp. 109 228–109 245, 2019.

[6] T. Ilhan and M. C. Pinar, "An efficient exact algorithm for the vertex p-center problem," *Preprint.[Online]. Available: http://www. ie. bilkent. edu. tr/ mustafap/pubs*, 2001.

	Values of n, m, k, UB	Algo 1	Algo 2	Algo 3
Computation Time	50, 250, 10, 500	2.449 s	0.073 s	0.110 s
Memory Usage		11.706 MB	5.793 MB	7.838 MB
Computation Time	50, 250, 10, 5×10^5	0.732 s	0.159 s	0.258 s
Memory Usage		11.715 MB	12.219 MB	17.241 MB
Computation Time	50, 250, 10, 5×10^8	36 s	0.256 s	0.449 s
Memory Usage		11.702 MB	18.572 MB	26.295 MB
Computation Time	50, 500, 10, 500	2.164 s	0.108 s	0.118 s
Memory Usage		11.941 MB	7.430 MB	10.195 MB
Computation Time	50, 500, 10, 5×10^5	0.812 s	0.232 s	0.296 s
Memory Usage		11.953 MB	16.311 MB	23.3 MB
Computation Time	50, 500, 10, 5×10^8	2.248 s	0.321 s	0.493 s
Memory Usage		11.947 MB	25.216 MB	35.715 MB
Computation Time	50, 1000, 20, 500	1.522 s	0.160 s	0.109 s
Memory Usage		12.062 MB	7.302 MB	6.331 MB
Computation Time	50, 1000, 20, 5×10^5	0.942 s	0.223 s	0.189 s
Memory Usage		12.062 MB	16.496 MB	15.534 MB
Computation Time	50, 1000, 20, 5×10^8	7.832 s	0.322 s	0.333 s
Memory Usage		12.062 MB	25.803 MB	26.682 MB
Computation Time	500, 1000, 50, 1000	TIMED OUT	12.228 s	10.176 s
Memory Usage			440.594 MB	466.278 MB
Computation Time	500, 1000, 50, 10^6	TIMED OUT	24.950 s	18.069 s
Memory Usage			788.276 MB	777.745 MB
Computation Time	500, 1000, 100, 1000	TIMED OUT	5.032 s	4.203 s
Memory Usage			429.477 MB	453.311 MB
Computation Time	500, 1000, 100, 10^6	TIMED OUT	8.051 s	6.643 s
Memory Usage			794.481 MB	709.276 MB
Computation Time	500, 10^4, 50, 1000	TIMED OUT	27 s	19.179 s
Memory Usage			556.530 MB	343.416 MB
Computation Time	500, 10^4, 50, 10^6	TIMED OUT	42.129 s	35.208 s
Memory Usage			704.500 MB	755.268 MB

Table 1: Experimental run-times and memory usage of the three algorithms