GIS Final Project : Facility Location

Christopher Blackman

December 2020

1 Facility Selection

The Facility Location Problem (FLP) is a branch of operations research and computational geometry concerned with optimal placement of facilities in order to minimize transportation costs, reachability, or avoiding placement near certain constraints [1].

We will be focusing on FLP based on distance. There are two major FLP based on this : the largest empty-circle problem, and the smallest enclosing circle problem. However, when taken to other metric spaces these concepts do not always apply.

Let T be a convex-polygon in the plane of R^2 such that $S = \{P_1, P_2, ..P_k\}$ be a set of k facility locations in T. Denote $|P, Q|$ as the euclidean distance between locations P, and Q. Then we can define the *largest empty-circle* problem as :

$$\max_{P \in T} \min_{P_i \in S} |P, P_i|$$

Where we are trying to find a point P which is maximizes, the minimum distance. Essentially, maximizing the least served area. It is also noted that we are finding a point P that is the center of the largest circle that does not contain another facility P_i. Thus the returned value is also the largest radius of an empty circle. This problem is useful when planning transport stops near population centers, fire stations, hospitals, or even store locations in a particular region.

The second major FLP is the *smallest enclosing disk*. We define it as the following :

$$\min_{P \in R^2} \max_{P_i \in S} |P, P_i|$$

Where we are trying to find the point P that minimized the maximum distance. Or, we are trying to find a point that is closest to all other facility locations. This problem is useful when planning for a distribution center, for other distribution centers. For example, it could be a lab, that all hospitals in the city use for testing, or a transit hub used to connect other transit hubs, often seen in intermodal freight transport hubs. In the survey by Kokichi Sugihara [2] showed this is possible with Farthest Point Voronoi Diagrams.

In this paper we will be focusing on the *largest empty-circle*, but in the graph. Our graph being a network of roads, and intersections. It is easy to see that Euclidean distance may not always apply as one hole will make the graph's shortest path different from Euclidean Shortest path.

Therefore, when we implement most of our algorithms, its true optimal value may not actually be optimal in application. The way that we compare our algorithms is by taking each of the k facilities and finding all closest points associated with the facilities in the graph, and take the sum of all paths belonging to the facilities.

1.1 Problem Definition

In order to keep the problem as simple as possible we define our problem as a graph G, where V are the intersections of the graph, and E are the roads connecting the intersections. We treat the intersections as facility locations, as well as populations centers of density one. One can upscale this where the intersections can have density's larger than one. The idea was to take a residence and map the residence to the closest intersection in the graph. In this way one, could decrease the number of nodes in the graph, at the loss of exact distance traveled. Which is fine in a city where roads are usually short, but not as much in a high highway, or farming community where distance between residences and intersections are much larger, depending on which country you live in.

2 Algorithms

2.1 Voronoi Diagrams (VD)

As discussed before the VD method is based on the *largest empty-circle* method. In order, to solve for the *largest empty-circle* we use the VD. We define our diagram regions satisfying the following:

$$R(S, P_i) = \{P \in R^2 : |P, P_i| < |P, P_j| \text{ for any } i \neq j\}$$

Or anything in the region $R(S, P_i)$ is the shortest path to point P_i. In our case $R(S, P_i)$ is the region in which the facility serves at point P_i. The R^2 plane is partitioned in n regions, where n is the number of currently existing facilities.

As defined before, the facility can only be placed inside the city region T. If we are solving for one new facility that serves the largest distance. Then we must find a point P that has the *largest empty-circle*. These must be in one of the three locations :

1. Voronoi Point
2. Vertex of Polygon T
3. Intersection of a Voronoi edge, and an edge of T

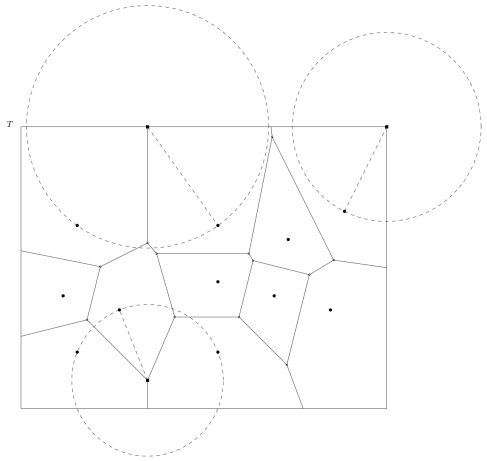


Figure 1: 3 Types Of Optimal placement in the Voronoi Diagram

By definition a Voronoi point is formed by the three regions being equidistant to the point. This results in being an empty circle in the VD.

If it is not a Voronoi Point, then it may be some circle of infinite radius. However, our VD is bounded by the polygon T. Thus we must check the intersection points, as well as when our polygon changes half-planes (vertices). One can prove by using the Pythagorean's theorem that the furthest points lie only in these two locations.

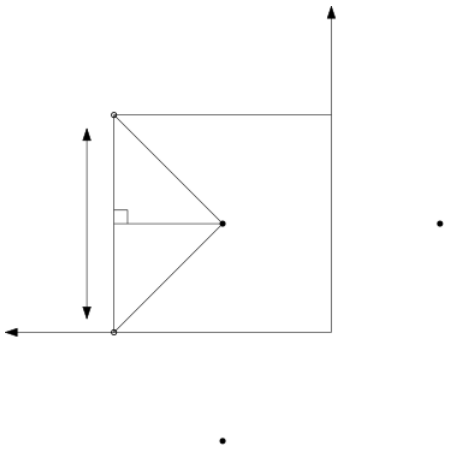


Figure 2: As you move away from the perpendicular line, your hypotenuse will always have larger length.

If we check all three conditions then we would have fulfilled the properties from the introduction. In return we will find the maximal radius of an empty

circle in T.

$$\max_{P \in T} \min_{P_i \in S} |P, P_i|$$

However, this only solves the problem for adding one point to an existing infrastructure. If we were adding in increments as well, then the prior addition may not be the optimal placement for the next facility. Furthermore, this method only works for areas that will have a uniform density, however in practice this is not always the case. We come up with a new method built on the previous. Let us denote our new problem as the following :

$$\min_{P_1,...,P_k} \max_{P \in T} \min_{i \in \{1,...,k\}} |P, P_i|$$

We are essentially solving the same problem, but now we are trying to minimize the set of k facilities. Note this definition does not account for the population density of a region.

Therefore, let us denote $u(P)$ as the population density. We mark this as the number of people living in region $R(S, P_i)$. Furthermore, we assume that all people require the service with equal probability. Then we may define our problem as :

$$\min_{P_1,...,P_k} \int_T u(P) \min_{i \in \{1,...,k\}} |P, P_i| dT$$

Notice, that for a particular facility we get an objective function of $F(P_1, ..., P_k)$ from $max_{P \in T} min_{i \in \{1,...,k\}} |P, P_i|$. This objective function is non-convex and difficult to solve with direct methods. Also notice for each facility, we are solving for one Voronoi region of our problem. Thus we can rewrite the solution as the following.

$$\min_{P_1,...,P_k} \int_{R(S,P_i) \cap T} u(P) |P, P_i| dT$$

Thus when approaching the problem we take the same idea as a gradient decent, where for a particular facility we change its location at iteration l+1 :

$$P_i^{l+1} = \Delta \cdot \left(\frac{\delta F}{\delta x_i^l}, \frac{\delta F}{\delta y_i^l} \right) + P_i^l$$

Where Δ is the rate that we climb the hill at. If we are solving for x_i^l then we are solving for the following (same idea when solving y) :

$$\begin{aligned}\frac{\delta F}{\delta x_i} &= \frac{\delta F}{\delta x_i}|P_1, P| + \frac{\delta F}{\delta x_i}|P_2, P| + ... + \frac{\delta F}{\delta x_i}|P_i, P| + ...\frac{\delta F}{\delta x_i}|P_k, P| \\ &= 0 + 0 + ...\frac{\delta F}{\delta x_i}|P_i, P| + ... + 0 \\ &= \frac{\delta F}{\delta x_i}\sqrt{x_i^2 + y_i^2} \\ &= \frac{x_i}{|xi, y_i|}\end{aligned}$$

Therefore, we solve for each Voronoi region our radius vector. We find the slop of that vector, with respect to our objective function. Then, head in the direction of the slop, and recalculate the VD based on the new positions. We stop when we reach some threshold, or some number of iterations. This approach is outlined by Kokichi Sugihara [2].

For our implementation we have a maximum iteration of 300, with a threshold of 0.001. We also have a delta of 0.1, and have no methods of changing delta based on the inertia of the objective function. Time-complexity wise VD takes $O(k \log k)$ to compute the VD, and $O(n \log k)$ in-order to compute region density, by using Nearest Neighbour search on the Voronoi points.

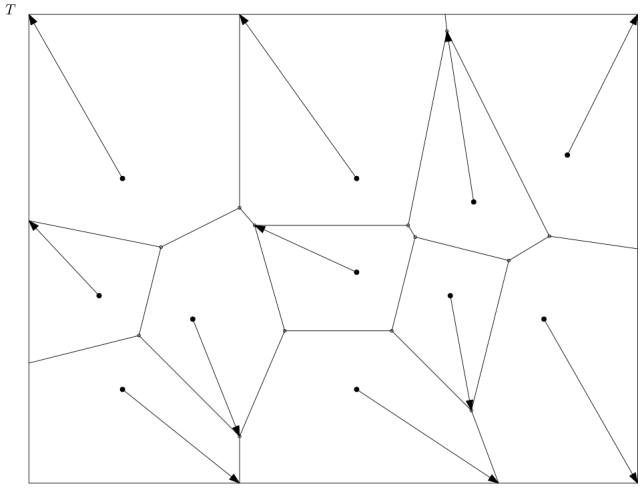


Figure 3: Direction that Voronoi points will travel

2.2 K-Means (KM)

K-Means is a classic cluster algorithm that is robust in many applications. In our case we use it in two possible ways. The first way we assume that the shortest path is Euclidean distance.

How does K-means work. First we choose k random centroids at random. We associate each point with its closest centroid. Then, for each cluster, we move the associated centroid to the average position of that cluster. Finally, if the previous objective function minus the current objective function is less than a certain threshold we stop. Or we stop when we have exceeded a certain number of steps.

We repeat the above a number of times, and choose the centroids with the smallest objective function. The objective function being the sum of all distances, with the associated clusters [3].

For our implementation we choose to have a maximum of 300 iterations per cluster computation, and 10 cluster iterations. Time complexity wise K-Means is $O(n)$ as the number of iterations, and number of clusters are constant with regards to input.

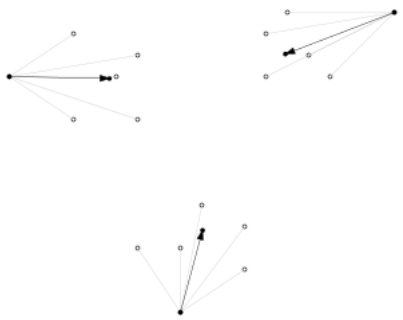


Figure 4: K-Means choosing a new centroid position in one iteration

2.3 K-Means Paths (KMP)

K-Means Paths, is similar to the K-means implementation. The only difference is that instead of assuming Euclidean as the shortest path, we calculate the shortest path in the graph G instead. When choosing a centroid position, we find the Nearest Neighbour vertex to the calculated centroid. Centroid position is based on the Euclidean distance of the cluster.

For our implementation we choose to have a maximum of 300 iterations per cluster computation, and 10 cluster iterations. Time complexity wise K-Means

is $O(n \cdot n \log n)$ since we perform a Dijkstra search per iteration on every intersection. Since the graph is planar, the number of edges in our graph is also bounded by a constant on the number of vertices/intersections.

3 Implementation Details

The graph we tested on was an implementation of a random planar graph by *tinyplasticgreyknight* on Github [4]. We later discovered that on larger graphs, planarity of the algorithm suffered. However, it was sufficient as a demo, in in real life there could be bridges, and tunnels.

The language of choice was python. Although python's not the fastest language, it does have a variety of tools that one can use, which is useful for projects like this where you can pull in a library quickly for experimentation.

As discussed in our Voronoi Model, we needed a city boundary. We defined our boundary as the minimum enclosing axis aligned rectangle containing our point set.

Since many of the algorithms have a quick termination, we experiment 10 times (n=10), each with a unique random seed in order to find a reasonable average. We tested facilities of $k = 3, 4, 5$ on data-sets of $50, 100, 200, 400, 800$, however, we could not finish all test as some would take 2 hours for only one iteration to complete.

4 Results

4.1 Ratio to K-Means Path

In this experiment we took the $l1$ norm of the final costs of the centroids and took the ratio of them when compared to K-Means. What we found was that K-Means Path out performed both Voronoi, and K-Means in general, but not by more than a factor of 2.

K-Means Path is expected to be better since it is calculating more closely to the graph. K-Means outperformed Voronoi, and it kinda makes sense since Voronoi is maximizing area, whereas k-Means is fitting to the current density of the point set.

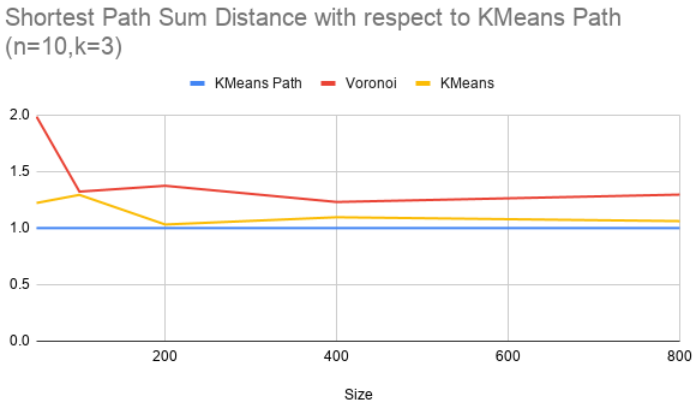


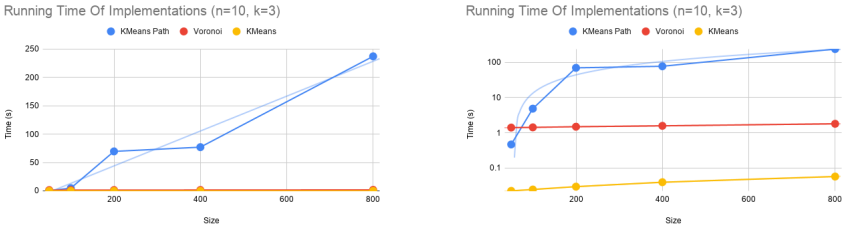
Figure 5

4.2 Run-Time

As one can see, the runtime of Voronoi, and K-Means is far faster than the runtime of K-Means Path. K-Menas Path is definitely not exponential, else it would have been linear in figure b) which it does not appear to be.

As for Voronoi, and K-Means. K-Means appears to be faster than Voronoi, and this is most likely due to the fact that there is less overhead compared to Voronoi. Since in Voronoi, we have to calculate the Voronoi Diagram, ray intersection with edges and other non-trivial calculations. My guess is as the input size gets larger, K-Means will run slower than Voronoi since the $logk$ is essentially a constant, and can be a very small one, compared to the multiple summations in K-Means.

It is noted that the results when running with $k = 5$ is similar to what is previously discussed.



(a) Linear Scale

(b) Log Scale

Figure 6: Run-times Of Algorithms

4.3 Distribution

What we see is the distribution of the objective function costs of each center. We are looking to see that centers have a small Total Cost, and are densely packed around their mean. If centers are not packed around their mean, then it means one of two things. Either one facility has many paths that are shorter/longer than the average, or one facility is taking the most occupants. Ideally we would want all facilities to have equal usage to minimize upgrade costs to a particular facility.

K-Means Path seems to be the best performant, cost centered around the mean. K-Means is similar to K-Means Path, but costs is slightly more expensive. But, this makes sense since K-Means Path performs slightly better than K-Means in terms of cost. Finally Voronoi has a very spread out distribution, which is not optimal.

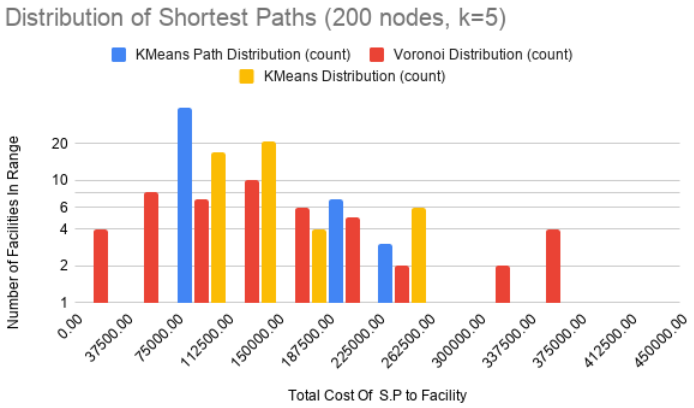


Figure 7: Distribution Of Objective with n = 10

5 Difficulties

Initially we wanted to test this on data from Open Ottawa [7] with their road data, and residential data. However, the time converting that data to a usable format was not worth the time lost in implementation of the algorithm.

The road network in the data they had left to, right to, left from, right from all mapped to IDs. They had two sets of IDs in this format, the first set the IDs were mismatched and did not map to the correct location. The second set of IDs were linking major intersections, but not linking side intersections. It was unclear if all major intersections were properly linked.

So the entire graph needed to be fixed automatically, since a manually fixing at most $(46,000)^2$ intersections is feasibly impossible for one person. We went about finding intersections with distance less than 10 meter radii, and finding endpoints intersecting edges by finding their projection onto the edge. The first operation was a $O(n \log n)$, but the second was $O(n^2)$ since I could not find a good algorithm for calculation.

That is where we learned that any n^2 operation takes too long to calculate. At one point, we calculated it would take a week to complete, which we did not have the time nor resources to make happen. Also, any calculation converting latitude, and longitude to a distance was also more computationally expensive than a 'l2' norm. Avoiding these calculations was sometimes necessary.

We also had some difficulties with uncommon calculations. One being a fast way to check for a ray intersection. Another being a fast way to check the orthogonal distance from a point onto a segment, and compute in a matrix for maximum speedup.

Lastly, translating a Voronoi diagram into a useful format in calculations. There is a lot of overhead that must be done, to get certain properties of the Voronoi diagram.

6 Conclusion

In conclusion, K-means is most likely the simplest, and best performing algorithm of the three. Although, K-Menas Path is closer to optimal the $O(n^2)$ runtime is not good for GIS applications. The approach of K-Means and Voronoi Diagrams are similar but they differ in one way; K-Means is optimizing for points in the data set, while Voronoi Facility Location is optimizing for the largest served area. If I was planning to build a new city, then I would most likely consider using the Voronoi implementation as I can optimize for building that are not currently placed. If I was planning for an existing city, I would most likely consider K-Means, as it is focusing more on the current population of the city.

However, there are some cases where using Voronoi Diagrams are useful. One being when the problem is a uniformly dense area, with many inhabitants. If the area is uniformly dense, then the Voronoi implementation does not need the density lookup, which means that the algorithm would run in $O(k)$ time, which is almost constant considering we don't need many hospitals to serve a population, in comparison to the n number of people that need a hospital. But considering the speed of current day computers, and densest city in the world currently having 606 people per km^2 with a population of 9 million, maximum being 14 million. This should take no more than an hour to compute.

It would be interesting to see the values compared to optimal results in the graph. But, since the problem is NP-complete, we would only be able to test on small data sets. There is also another approximation algorithm which we did not cover, which would be interesting to compare to these [5]. This algorithm grantees 2 times optimal from the solution.

For the project repository please visit [6].

References

[1] *Facility location problem. (2020, May 26). Retrieved from https://en.wikipedia.org/wiki/Facility_location_problem*

[2] *Sugihara K. (2008) Voronoi Diagrams in Facility Location. In: Floudas C., Pardalos P. (eds) Encyclopedia of Optimization. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-74759-0_707*

[3] *Hastie, T., Friedman, J., & Tisbshirani, R. (2017). The Elements of statistical learning: Data mining, inference, and prediction. New York: Springer.*

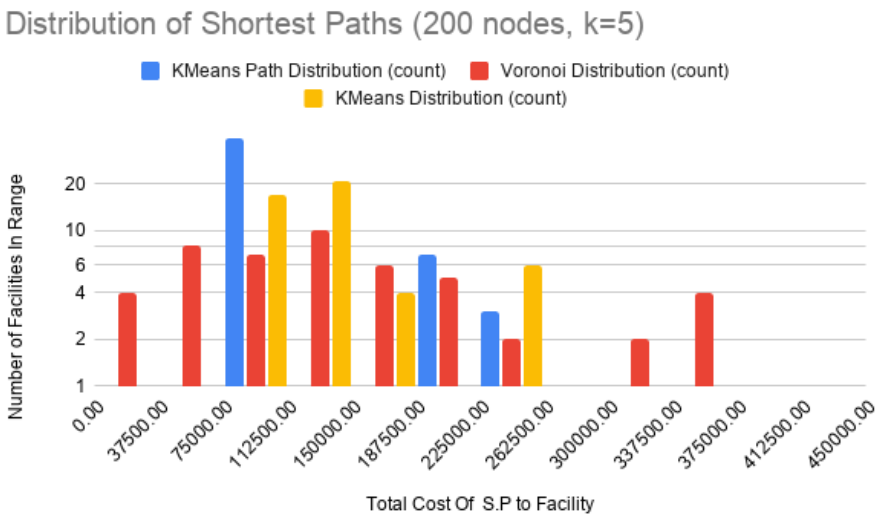
[4] *Tinyplasticgreyknight. "Tinyplasticgreyknight/Random-Planar-Graph." GitHub, github.com/tinyplasticgreyknight/random-planar-graph.*

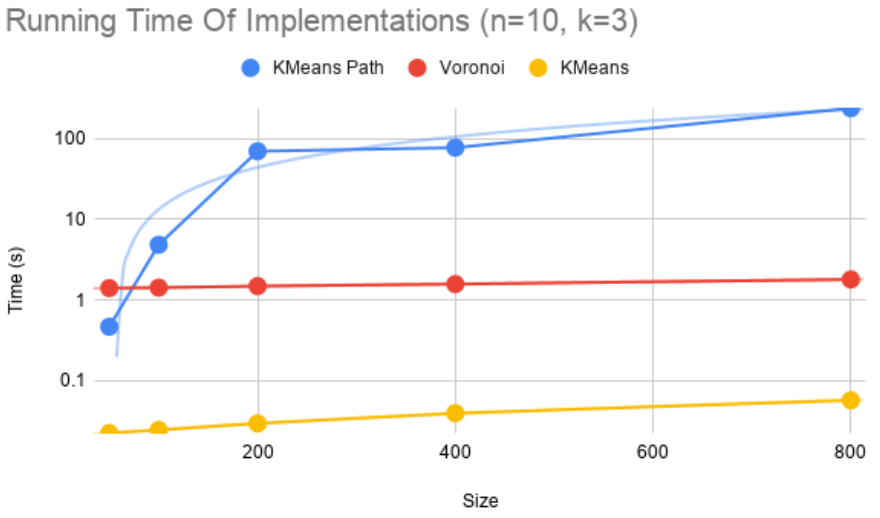
[5] *Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, 38, 293-306. doi:10.1016/0304-3975(85)90224-5*

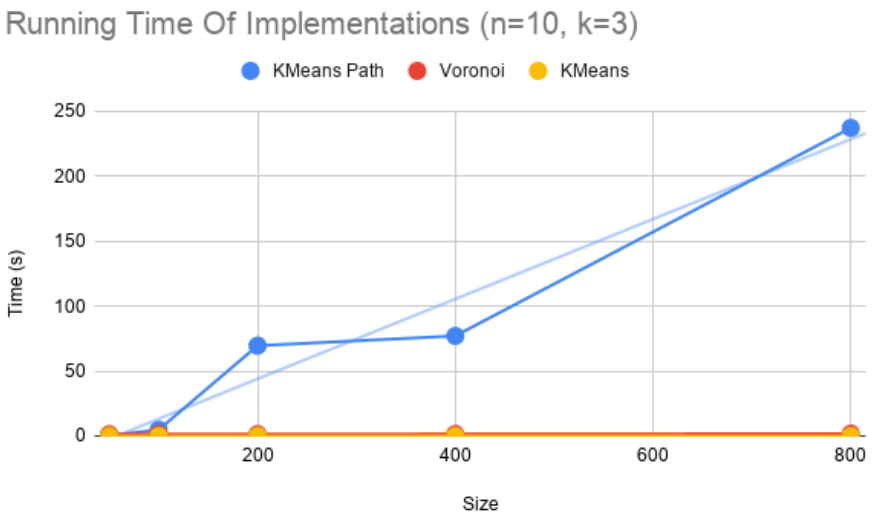
[6] *https://github.com/ChristopherBlackman/Facility-Location-Ottawa*

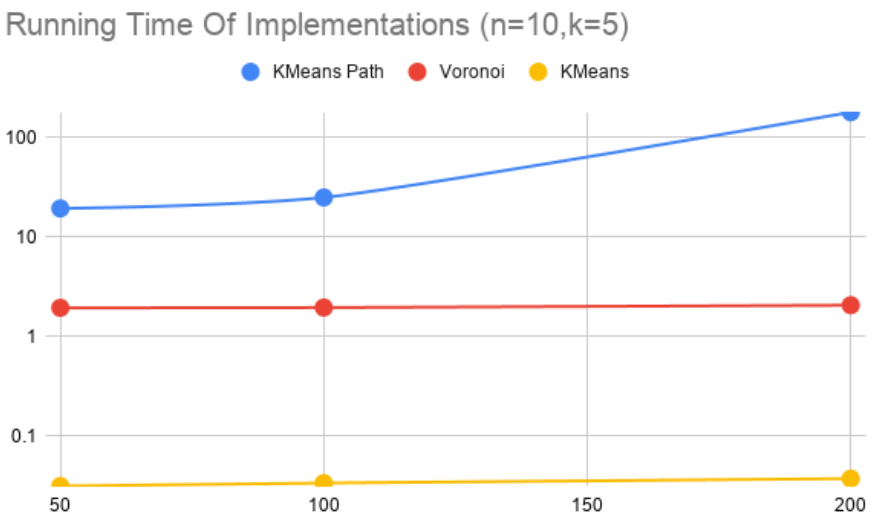
[7] Open Ottawa, open.ottawa.ca

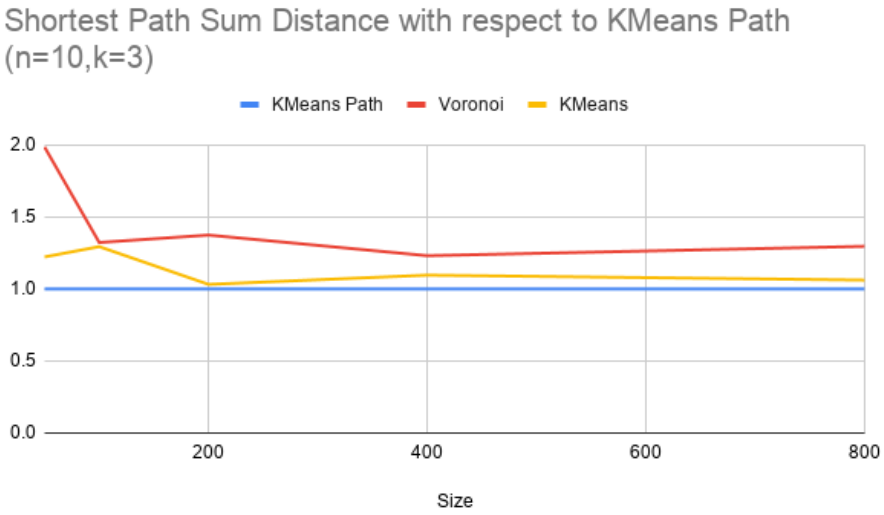
7 Appendix

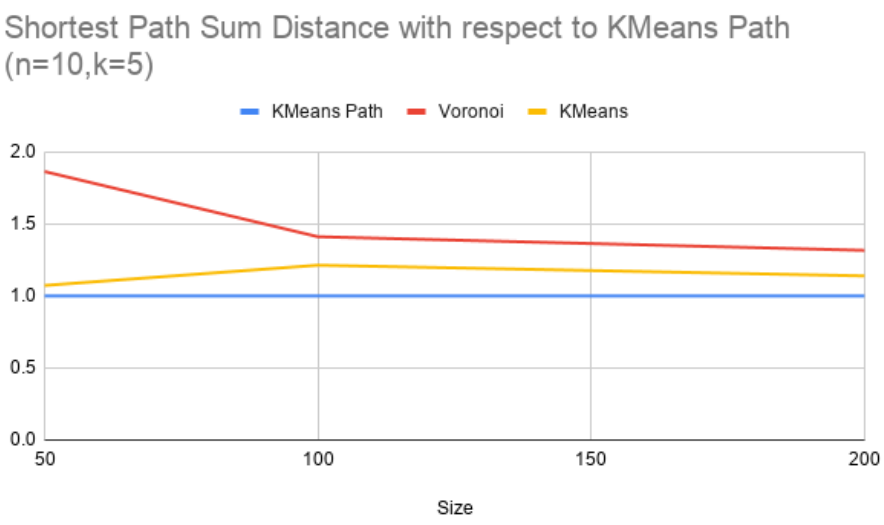

Distribution of Shortest Paths (200 nodes, k=5)
KMeans Path Distribution (count)
Voronoi Distribution (count)
KMeans Distribution (count)
Number of Facilities In Range
20
10
6
4
2
1
0.00
37500.00
75000.00
112500.00
150000.00
187500.00
225000.00
262500.00
300000.00
337500.00
375000.00
412500.00
450000.00
Total Cost Of S.P to Facility


Running Time Of Implementations (n=10, k=3)
KMeans Path
Voronoi
KMeans
Time (s)
100
10
1
0.1
200
400
600
800
Size


Running Time Of Implementations (n=10, k=3)
KMeans Path
Voronoi
KMeans
250
200
150
100
50
0
Time (s)
200
400
600
800
Size


Running Time Of Implementations (n=10,k=5)
KMeans Path
Voronoi
KMeans
100
10
1
0.1
50
100
150
200


Shortest Path Sum Distance with respect to KMeans Path (n=10,k=3)
KMeans Path
Voronoi
KMeans
2.0
1.5
1.0
0.5
0.0
200
400
600
800
Size


Shortest Path Sum Distance with respect to KMeans Path (n=10,k=5)
KMeans Path
Voronoi
KMeans
2.0
1.5
1.0
0.5
0.0
50
100
150
200
Size

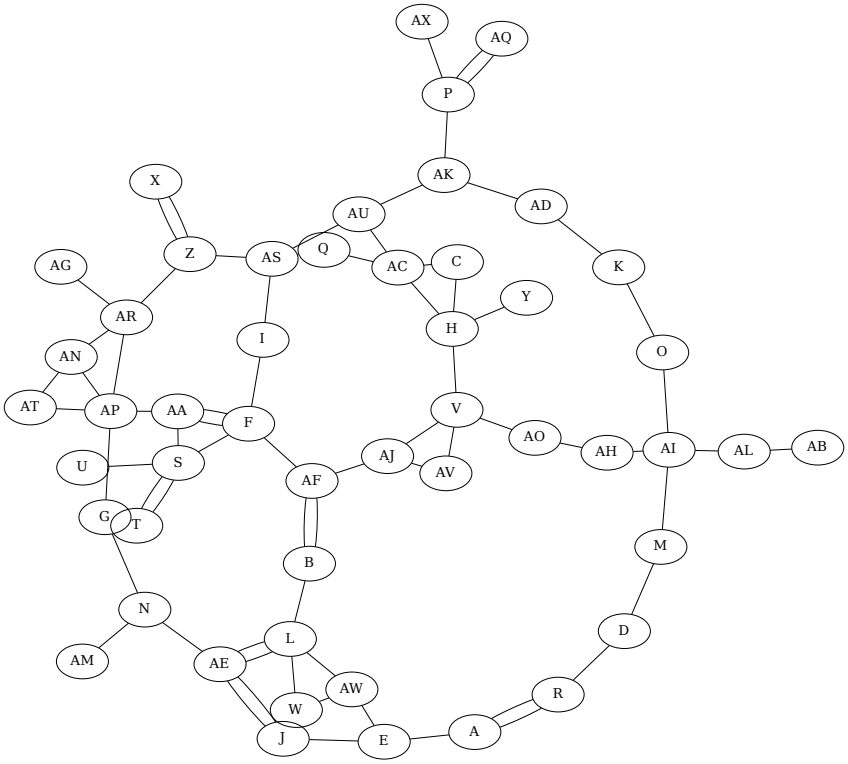


Figure 8: Tested Graph on 50 Nodes

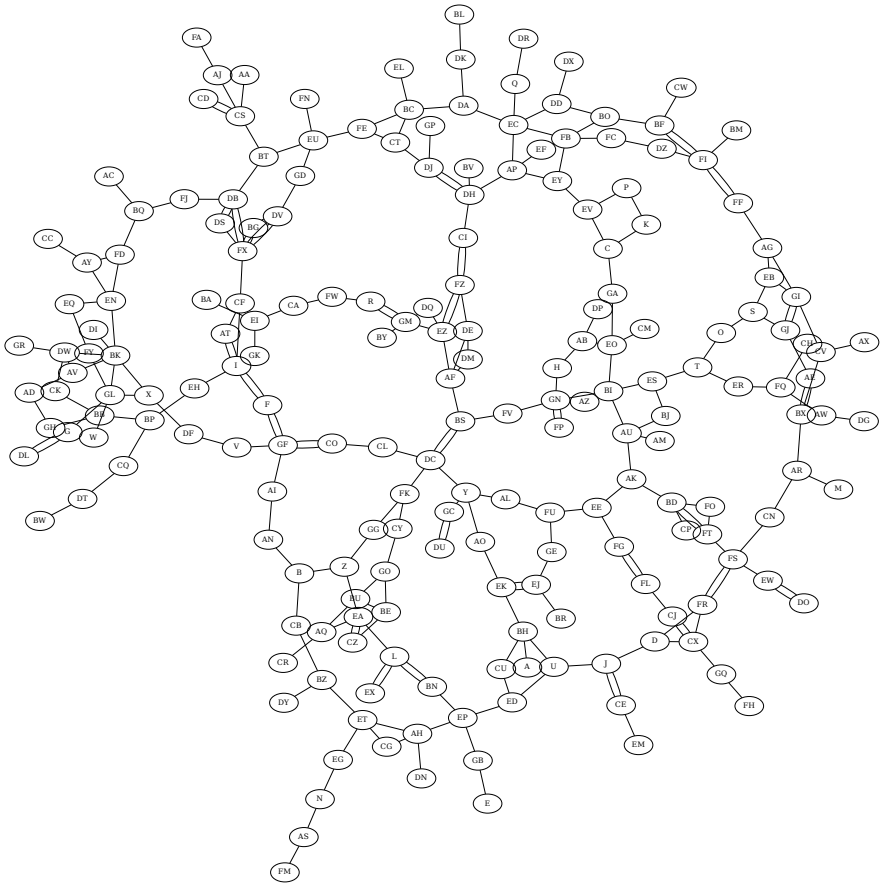


Figure 9: Tested Graph On 100 Nodes