Introduction to Algorithms in Facility Location Problem

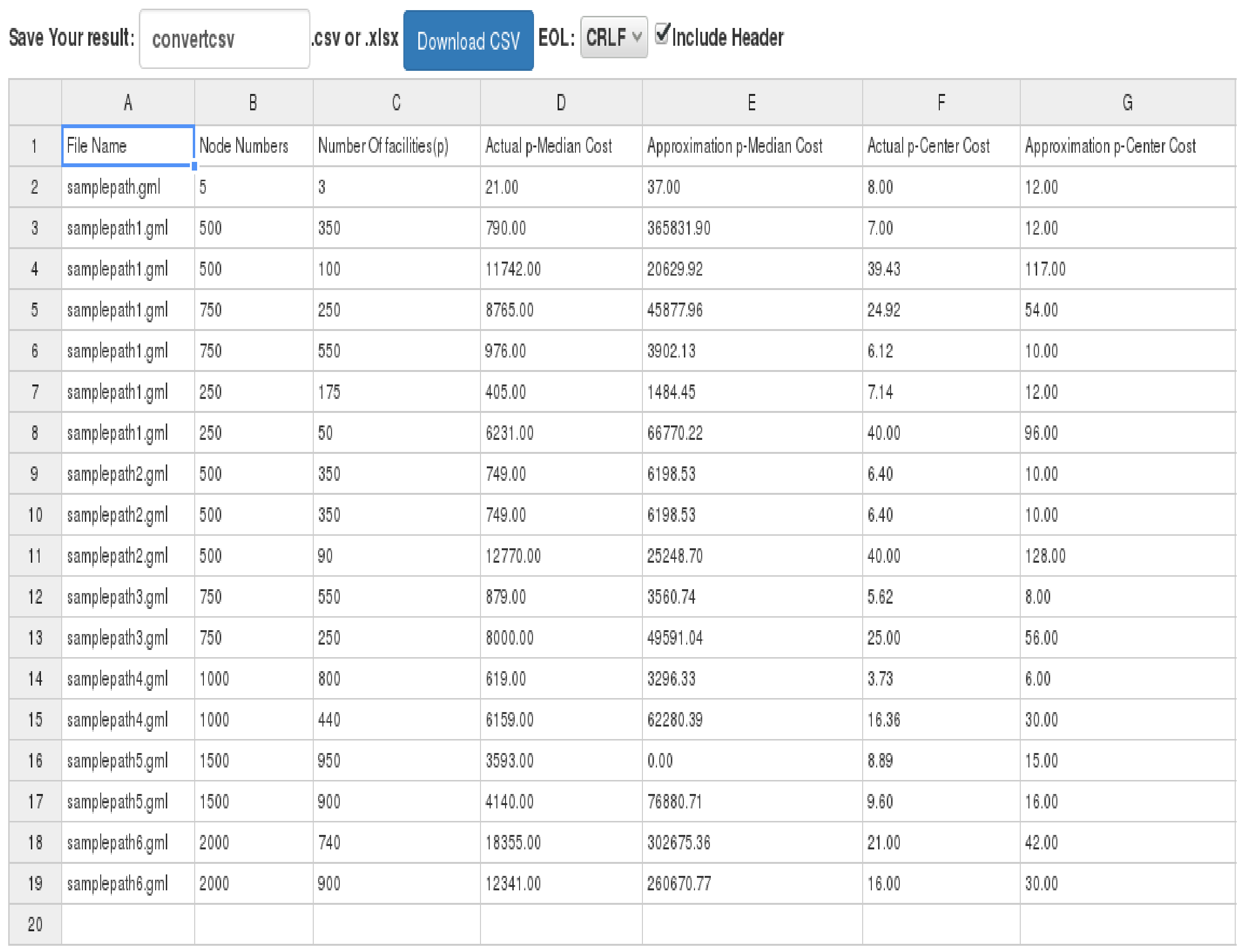
Project Report

Submitted by : Rajib Chandra Das
ID : 001201973

July 14, 2019

An implementation of polynomial time algorithm to determine the weighted p-center and p-median

Results: We tested our algorithm for several cases. Here we enclosed a CSV file where small, medium and large scale instances are shown.



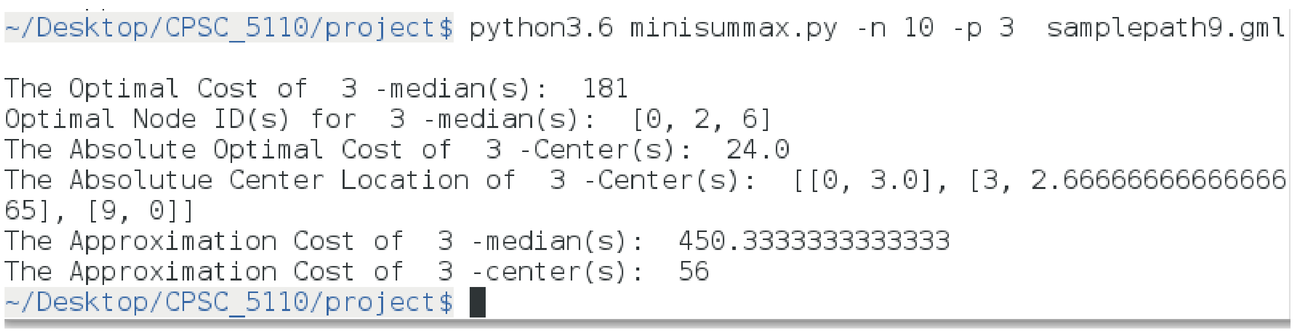
Save Your result: convertcsv .csv or .xlsx Download CSV EOL: CRLF [✓ Include Header]

	A	B	C	D	E	F	G
1	File Name	Node Numbers	Number Of facilities(p)	Actual p-Median Cost	Approximation p-Median Cost	Actual p-Center Cost	Approximation p-Center Cost
2	samplepath.gml	5	3	21.00	37.00	8.00	12.00
3	samplepath1.gml	500	350	790.00	365831.90	7.00	12.00
4	samplepath1.gml	500	100	11742.00	20629.92	39.43	117.00
5	samplepath1.gml	750	250	8765.00	45877.96	24.92	54.00
6	samplepath1.gml	750	550	976.00	3902.13	6.12	10.00
7	samplepath1.gml	250	175	405.00	1484.45	7.14	12.00
8	samplepath1.gml	250	50	6231.00	66770.22	40.00	96.00
9	samplepath2.gml	500	350	749.00	6198.53	6.40	10.00
10	samplepath2.gml	500	350	749.00	6198.53	6.40	10.00
11	samplepath2.gml	500	90	12770.00	25248.70	40.00	128.00
12	samplepath3.gml	750	550	879.00	3560.74	5.62	8.00
13	samplepath3.gml	750	250	8000.00	49591.04	25.00	56.00
14	samplepath4.gml	1000	800	619.00	3296.33	3.73	6.00
15	samplepath4.gml	1000	440	6159.00	62280.39	16.36	30.00
16	samplepath5.gml	1500	950	3593.00	0.00	8.89	15.00
17	samplepath5.gml	1500	900	4140.00	76880.71	9.60	16.00
18	samplepath6.gml	2000	740	18355.00	302675.36	21.00	42.00
19	samplepath6.gml	2000	900	12341.00	260670.77	16.00	30.00
20							

Figure 1: Tested Results in csv file

Here, in our program if the problem is not **t-feasible** then the approximation median cost is zero. A problem instance is said to be tfeasible if and only if every vertex supply can evacuate to a facility within time t and the total number of the facility should not exist more than p.

As we tested for some large instances that is why we did not include the optimal median locations and absolute p-center locations on csv file. But if you run our code then in output screen the locations are shown as follows:



```
~/Desktop/CPSC_5110/project$ python3.6 minisummax.py -n 10 -p 3  samplepath9.gml

The Optimal Cost of  3 -median(s):  181
Optimal Node ID(s) for  3 -median(s):  [0, 2, 6]
The Absolute Optimal Cost of  3 -Center(s):  24.0
The Absolutue Center Location of  3 -Center(s):  [[0, 3.0], [3, 2.6666666666666
65], [9, 0]]
The Approximation Cost of  3 -median(s):  450.3333333333333
The Approximation Cost of  3 -center(s):  56
~/Desktop/CPSC_5110/project$ 
```

Figure 2: Locations for median and centers

Here, in the Absolute center locations we showed a list, the first parameter indicates the node id and the second one represents the distance apart from that node. For example, Let us consider the the first list tuple, that is [0, 3.0], that means, from node zero the facility should be located at 3 unit distance apart.

Description of Our Algorithm: In our program we developed the following four modules:

- Optimal Median Cost
- Absolute Center Cost
- Approximation Median Cost
- Approximation Center Cost

There are also some sub-modules as a requirement of this module, for example, Optimal Median locations and absolute Center locations. We need these locations for computing approximation cost. Now we are going to describe our algorithm briefly:

Details on finding optimal p-median cost: We are dealing with p-median problem on a path, we are using dynamic programming approach based on [1]. The time complexity to find optimal p-median is $O(pn^2)$ as we computed leftSum and rightSum cost for facility in preprocessing step. In code, optimal p-median algorithm is implemented under function named "optPMedian()) and sumLeft and sumRight are pre-processed under array named "sumL" and "sumR". optPMedian() function outputs optimal p-median cost and node ids of the facility location. These node ids are then used in approximating the p-center problem.

Details on finding absolute optimal p-center cost using feasibility test: First we enumerated all the pairs of vertices. And since we have the symmetry on the cost matrix, we just took the lower half of the triangle. So space required to save these enumerated pairs was $O(n^2)$. Also the time taken to calculate these pairs was $O(n^2)$. Then we sorted all the pairs in non-decreasing order of distances. Merge sort was used for the purposes for sorting the weighted distances of the dominating pairs. This took another $O(n^2 logn)$. In code, enumeration and sorting of the pairs can be found in function name ”funcEnumeratePairs()”. Then using binary search, we found the leftmost value where numbers facility required to covered the path was less than the value of p. Binary search is implemented on function named “funcBinarySearch()”. This searching algorithm requires $O(logn)$ time. Then we returned the set of facilities obtained from the coverage algorithm. In our program we implemented coverage algorithm in function named “funcCoverageProblem()” Note: if p is not feasible for the path, then our program returns false as it won’t satisfy the condition mentioned above

Details on finding approximated p-center cost: After finding the locations of facilities for p-median, we used that locations to calculate the approximate p-center. In our program, the approximated p-center cost is determined under the function named ‘approxiCenter()’. For every vertex, we checked it’s left facility location and the right facility location and compared the weighted distance and which facility gave minimum weighted distance, we took that value. And the approximate p-center was the maximum of those weighted distances. The complexity for performing this algorithm needs $O(n^2)$ time.

Details on finding approximated p-median cost: Once we had the absolute locations of facilities for p-center, we used that locations to calculate the approximate p-median. We calculated the approximates p-median cost i the function named ”approxiPMedian()”. The variable ’Opt-p-center’ cost is tracked to find which facility location should go for each vertex. The idea is to find a cost of vertex to the current pointed facility location, and if the cost is above the ’opt-p-center’ cost, we know that the vertex is far from the current pointed facility location. So the current facility marker should be pointed to another facility which would be the nearest facility for that vertex. So in this fashion we calculate the cost of each vertex to its nearest facility location. And the sum of all cost is the approximate p-median cost. This problem is implemented in O(n) time.

Challenges that addressed:

At first we developed an algorithm as it was discussed a brute force method by Higashikawa et al. [2] for placing multiple sink on path. We implemented that algorithm on the grounds that if we do not consider edge capacity and travel time per unit of distance then it can reduce to our center problem. here, we assumed the first location would be on the first vertex and the rest of the centers will be located in between the remaining vertices. And we placed the first on every vertex and edge and then calculate for rest of them. When we run this application for smaller instances then we did not face any problem. But for larger cases, for example for 200 nodes and 90 centers we found out that our program can not produce output even running for long time.

To overcome this problem, we discussed with course teacher and came up an idea of feasibility test by enumerating pairs and using binary search fashion. We developed this algorithm and found a better result. Now we can compute for 1500 nodes and 950 facilities. We are very proud of this improvement.

Challenges that need to be addressed: After finding the optimal positions for center and median we are eligible to compute the approximation cost for both of them. In our program, for given nodes and facilities number at first we calculated actual median cost, absolute center cost, then approximation median cost and finally approximation center cost. And this four four modules run one after another. That is why it took so many times to get the final output. For this reason, our future scope will be to to implement threading in python code at least to determine the approximation cost part.

Moreover, by using code re-factoring, addressing some redundant calling, preprocessing the weighted distance and implementing thread we can resolve the time complexity issue.

Compile Instructions:

- First, from terminal go to the folder where our code is available.
- Run the following command if you want to create a new gml file with random vertex weights and edge length.

  ```
  python3.6 minisummax.py -n 1500 -p 1000 sample.gml
  ```

  Here,

  - **python3.6**, indicates the command to run the python code with version (3.6).
    minisummax.py, is our program file which contains minisum and minimax problem in one instance.
  - **-n 1500**, to get the number of nodes from user. Here, we gave 1500 as a node numbers.
  - **-p 1000**, represents the number of medians. In this example we used 1000 as facilities number.
  - **sample.gml**, would be the gml file name where our program will write the network path. While the writing on path is done then our program will read

  If you want to use the existing file and try with different number of facilities then run the following command:

  ```
  python3.6 minisummax.py -p 199 sample.gml
  ```

  All notations are the same as in previous one.

References

[1] Refael Hassin and Arie Tamir. Improved complexity bounds for location problems on the real line. *Operations Research Letters*, 10(7):395–402, 1991.

[2] Yuya Higashikawa, Mordecai J Golin, and Naoki Katoh. Multiple sink location problems in dynamic path networks. *Theoretical Computer Science*, 607:2–15, 2015.