Minimum k-Chinese Postman Problem

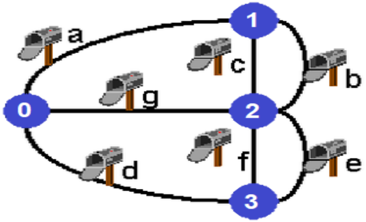
Final Presentation

Akif Kartal

Advisor: Prof. Dr. Didem GÖZÜPEK

15 June 2022

Problem Definition



Given a multigraph $G = (V, E)$ initial vertex $s \in V$ length $l(e) \in N$ for each $e \in E$ the *minimum k-Chinese postman problem* is to find k tours(cycles) such that each containing the initial vertex s and each edge of the graph has been traversed at least once and the most expensive tour is minimized.[1]

What we did?

- Making literature research and understanding the problem.
- Choosing and designing algorithms.
- Implementing both heuristic and exhausted search algorithms.
- Making complexity analysis of the algorithms.
- Testing with different parameters and performance evaluation.
- Making comparison and numerical evaluation of algorithms.
- Showing the comparison average results on the charts.
- Creating a GUI and running algorithms on that GUI.

Tools and Technologies

In order to make this project, I have used following tools and technologies.

python™

Qt

igraph

LaTeX

PyCharm

Windows 10

git

GitHub

Heuristic Algorithm

In order to solve this problem, I have implemented a heuristic augment-merge algorithm.[2] Steps of this algorithm are following.

1. Sort the edges e in decreasing order according to their weight.
2. In decreasing order according to $w(C_e)$, for each $e = v_i, v_j \in E$, create the closed walk $C_e = (SP(v_1, v_i), e, SP(v_j, v_1))$, if e is not already covered by an existing tour.
3. Let $C = (C_1, \ldots, C_m)$ be the resulting set of tours. If $m = k$ we are done and have computed an optimal k -postman tour.
4. If $m < k$ we add $k - m$ “dummy” tours to C, each consisting of twice the cheapest edge incident to the depot node.
5. While $|C| > k$ we merge tour C_{k+1} with a tour from $C_1, ..., C_k$ such that the weight of the merged tour is minimized.

Heuristic Algorithm Complexity Analysis

* $n = |E|$

Complexity	Algorithm Step
$\mathcal{O}(n^2)$	Sort the edges e in decreasing order according to their weight.
$\mathcal{O}(n^3)$	For each $e = v_i, v_j \in E$, create the closed walk.
$\mathcal{O}(n^2)$	If number of cycle(m) $< k$ add $k - m$ "dummy" tours.
$\mathcal{O}(n^4)$	If number of cycle(m) $> k$ merge tour C_{k+1} with a tour from $C_1, ..., C_k$.

Table: Complexity Analysis of Heuristic Algorithm Steps

Overall Complexity of Heuristic Algorithm

Best Case: $\mathcal{O}(n^3)$
Average Case: $\mathcal{O}(n^3)$
Worst Case: $\mathcal{O}(n^4)$

Exhaustive Search Algorithm

In this project, **to make a comparison**, we need to implement another algorithm. For this purpose, I have implemented a simple exhaustive search algorithm. Steps of this algorithm are as follows.

1. Firstly, it finds all possible cycles in a graph by using a simple recursive algorithm like Depth-first search.[3]
2. Then, for the cycles found in the previous step, it finds all distinct combinations of a given length k.[4]

   C(all possible cycles, k)
3. Then, it finds all set of cycles that satisfy and holds the problem conditions in found combinations.
4. Lastly, we choose the k cycle that has a minimum length in set of cycles that have been found in the previous step.

Exhaustive Search Complexity Analysis

* $m = |V|, n = |E|$

Complexity	**Algorithm Step**
$\mathcal{O}(m+n)$	Finding all cycles in undirected graphs.[5]
$\mathcal{O}(n^n)$	Find all k combinations of found cycles in the previous step. [4]
$\mathcal{O}(n^3)$	Finding all cycles that satisfy the problem conditions in found combinations.
$\mathcal{O}(n^4)$	Choosing the cycle that has a minimum length in cycles that have been found.

Table: Complexity Analysis of Exhaustive Search Algorithm Steps

Overall Complexity of Exhaustive Search Algorithm

Best Case: $\mathcal{O}(n^n)$
Average Case: $\mathcal{O}(n^n)$
Worst Case: $\mathcal{O}(n^n)$

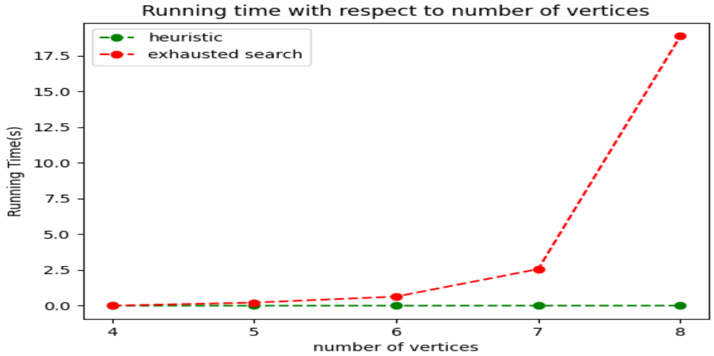
Comparison and Numerical Evaluation

In this comparison, we will change the both number of nodes and the number of edges which means we will have a bigger graph. The k value will be constant. We will create **20 different graphs** and we will compare the average of the results.

Parameters will be as follows.

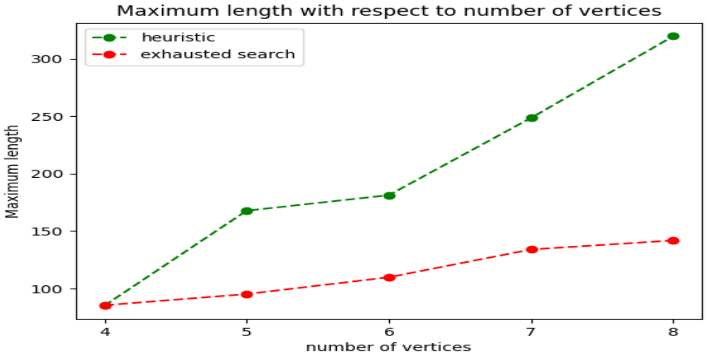
- **initial vertex** = 0
- **k** = 3
- **number of nodes** = 4, 5, 6, 7, 8
- **number of edges** = 6,9,10,12,12

Running Time Comparison Results



In the above chart when the graph is growing running time of the exhausted search algorithm is increasing exponentially. This is an expected result because **in bigger graphs we have more cycles** and to get k combination of that cycles **we need more time.**

Maximum Length Comparison Results



In the above chart, the heuristic algorithm gets worse results. This is an expected result because when the **k value is small** we have to **merge** the found tours. **After merging in the heuristic algorithm, the maximum length is increasing.**

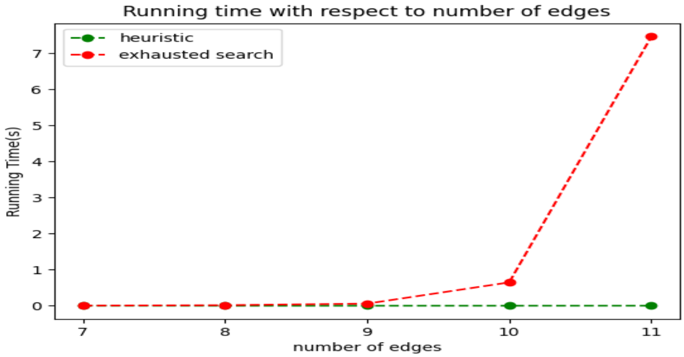
Comparison and Numerical Evaluation

In this comparison, we will change only the number of edges which means the density of the graph will change. The k value and number of nodes will be constant. We will create **20 different graphs** and we will compare the average of the results.

Parameters will be as follows.

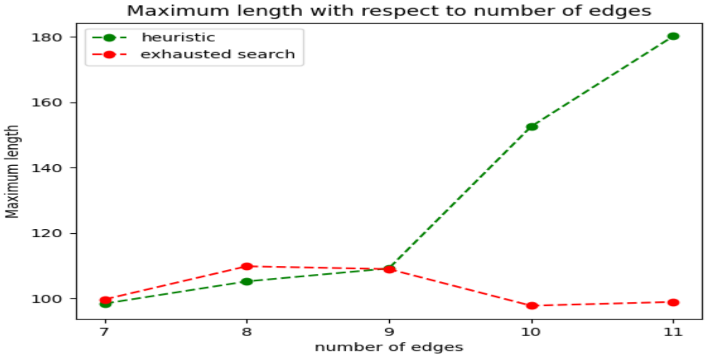
- **initial vertex** = 0
- **k** = 4
- **number of nodes** = 6
- **number of edges** = 7, 8, 9, 10, 11

Running Time Comparison Results



In the above chart **when the graph density is growing** running time of the exhausted search algorithm **is increasing exponentially.** This is again an expected result because in the dense graphs **we have more cycles** and to get the k combination of that cycles **we need more time.**

Maximum Length Comparison Results



In the above chart, when the graph density is growing the heuristic algorithm gets better results, if the k value is proportional to graph size. But, **if the k value is not proportional to graph size and constant** as you can see exhausted search gets better results.

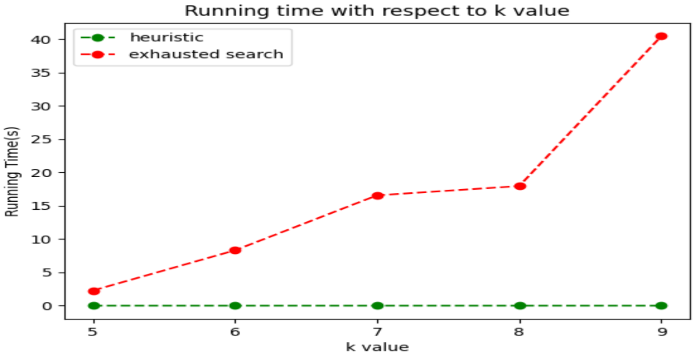
Comparison and Numerical Evaluation

In this comparison, we will change only the k value. Number of node and number of edge will be constant. We will create **20 different graphs** and we will compare the average of the results.

Parameters will be as follows.

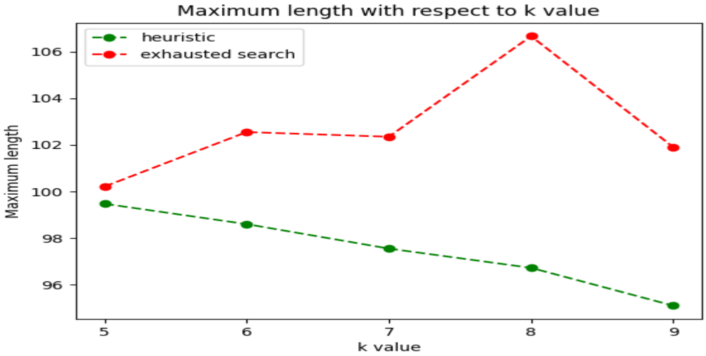
- **initial vertex** $= 0$
- **k** $= 5, 6, 7, 8, 9$
- **number of nodes** $= 6$
- **number of edges** $= 10$

Running Time Comparison Results



In the above chart when the k value is increasing running time of the exhausted search algorithm is increasing. Because the **exhausted search algorithm has to get the k combination in any case and this operation takes time.**

Maximum Length Comparison Results



In the above chart when the **k value is increasing** heuristic algorithm gets **better results.** Because **when the k value is big heuristic algorithm doesn't need to merge tours** therefore it gets better results as we expect in this project.

Summary of Comparison Results

As we have seen in the comparisons unfortunately in any case **exhausted search takes a long time.** But still, it can produce very good results. On the other hand heuristic algorithm is good for running time. But it produces bad results when the k value is small and not proportional to graph size. Because in that case, it has to make merge operation after merging the result is not good according to the exhausted search algorithm that means we can optimize the merging operating. But **when the k value is big and proportional to graph size heuristic algorithm produces better results** than the exhausted search algorithm in a very short time as we expect in this project.

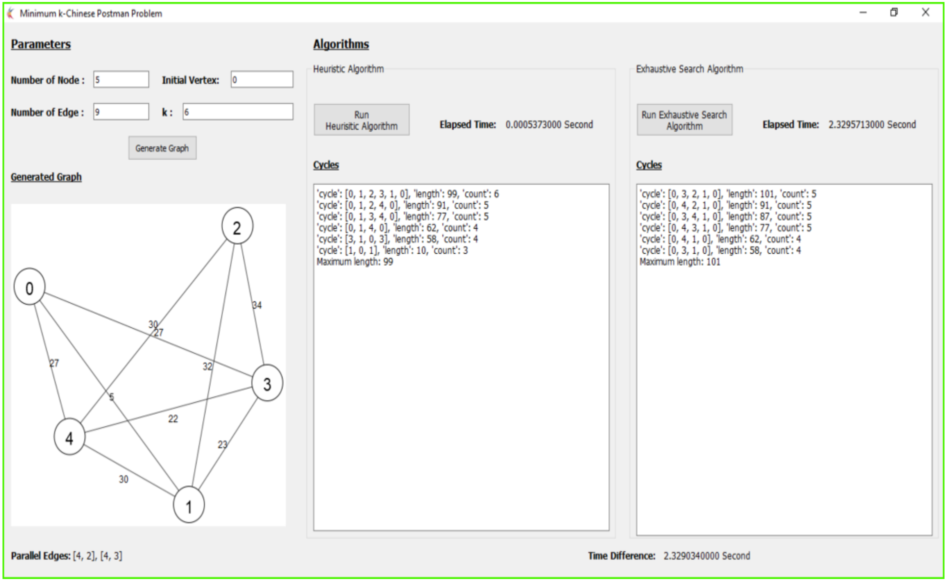
Graphical User Interface

For this project in order to show the results and how algorithms work, we need to create a Graphical User Interface.

Features of my graphical user interface are as follows.

- User can enter the parameter values.
- User can generate and see graph.
- User can see parallel edges as text.
- User can run the algorithms.
- User can see the algorithm results.
- User can see the elapsed time for each algorithm.
- User can see the elapsed time difference between algorithms.

Graphical User Interface



Minimum k-Chinese Postman Problem

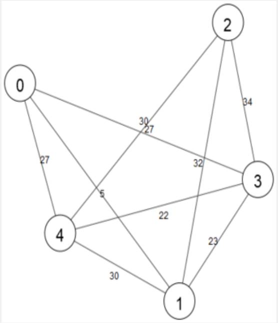
Parameters

Number of Node : 5 **Initial Vertex:** 0

Number of Edge : 9 **k :** 6

Generate Graph

Generated Graph



Parallel Edges: [4, 2], [4, 3]

Algorithms

Heuristic Algorithm

Run Heuristic Algorithm **Elapsed Time:** 0.0005373000 Second

Cycles

```
'cycle': [0, 1, 2, 3, 1, 0], 'length': 99, 'count': 6
'cycle': [0, 1, 2, 4, 0], 'length': 91, 'count': 5
'cycle': [0, 1, 3, 4, 0], 'length': 77, 'count': 5
'cycle': [0, 1, 4, 0], 'length': 62, 'count': 4
'cycle': [3, 1, 0, 3], 'length': 58, 'count': 4
'cycle': [1, 0, 1], 'length': 10, 'count': 3
Maximum length: 99
```

Exhaustive Search Algorithm

Run Exhaustive Search Algorithm **Elapsed Time:** 2.3295713000 Second

Cycles

```
'cycle': [0, 3, 2, 1, 0], 'length': 101, 'count': 5
'cycle': [0, 4, 2, 1, 0], 'length': 91, 'count': 5
'cycle': [0, 3, 4, 1, 0], 'length': 87, 'count': 5
'cycle': [0, 4, 3, 1, 0], 'length': 77, 'count': 5
'cycle': [0, 4, 1, 0], 'length': 62, 'count': 4
'cycle': [0, 3, 1, 0], 'length': 58, 'count': 4
Maximum length: 101
```

Time Difference: 2.3290340000 Second

Success Criteria

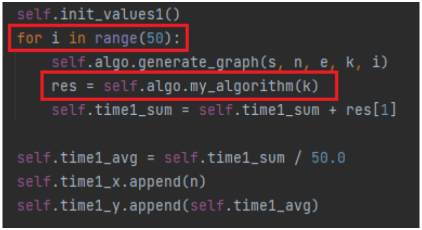
1. Heuristic algorithm complexity will be better than $\mathcal{O}(|E|^4)$

I have accomplished this criterion **successfully**. In heuristic algorithm I have $\mathcal{O}(|E|^3)$ complexity as we have seen complexity analysis.

Success Criteria

2. Creating 50 different random graphs for each test case in performance testing and creating 20 different random graphs for each comparison case and taking the average of them.

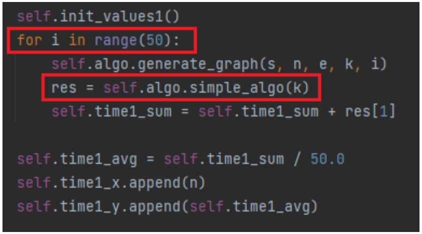
I have accomplished this criterion **successfully**.



```
self.init_values1()
for i in range(50):
    self.algo.generate_graph(s, n, e, k, i)
    res = self.algo.my_algorithm(k)
    self.time1_sum = self.time1_sum + res[1]

self.time1_avg = self.time1_sum / 50.0
self.time1_x.append(n)
self.time1_y.append(self.time1_avg)
```

(a) Graph Generation for Heuristic Algorithm



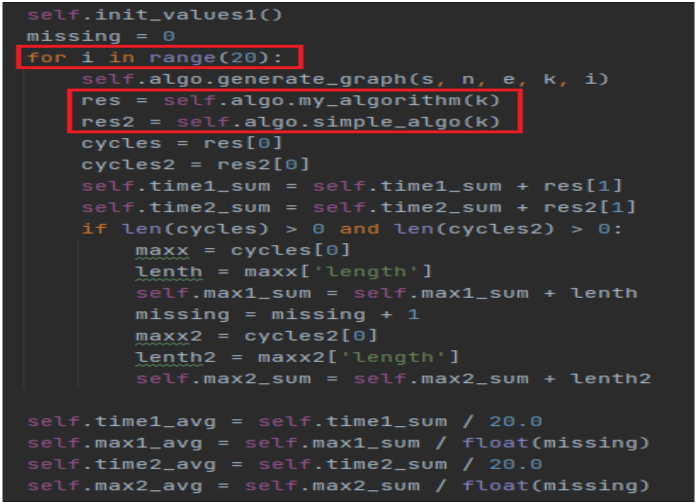
```
self.init_values1()
for i in range(50):
    self.algo.generate_graph(s, n, e, k, i)
    res = self.algo.simple_algo(k)
    self.time1_sum = self.time1_sum + res[1]

self.time1_avg = self.time1_sum / 50.0
self.time1_x.append(n)
self.time1_y.append(self.time1_avg)
```

(b) Graph Generation for Exhausted Search Algorithm

Figure: 50 Random Graph Generation for Algorithms

Success Criteria



```
self.init_values1()
missing = 0
for i in range(20):
    self.algo.generate_graph(s, n, e, k, i)
    res = self.algo.my_algorithm(k)
    res2 = self.algo.simple_algo(k)
    cycles = res[0]
    cycles2 = res2[0]
    self.time1_sum = self.time1_sum + res[1]
    self.time2_sum = self.time2_sum + res2[1]
    if len(cycles) > 0 and len(cycles2) > 0:
        maxx = cycles[0]
        lenth = maxx['length']
        self.max1_sum = self.max1_sum + lenth
        missing = missing + 1
        maxx2 = cycles2[0]
        lenth2 = maxx2['length']
        self.max2_sum = self.max2_sum + lenth2

self.time1_avg = self.time1_sum / 20.0
self.max1_avg = self.max1_sum / float(missing)
self.time2_avg = self.time2_sum / 20.0
self.max2_avg = self.max2_sum / float(missing)
```

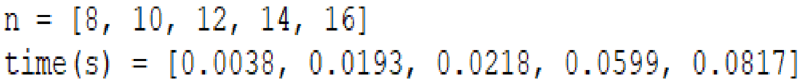
Figure: 20 Random Graph Generation for Comparison

Success Criteria

3. Getting results with the heuristic algorithm in less than 1 second when number of nodes < 25 and k is not constant.

I have accomplished this criterion **successfully**. As you can see in results, we have less than 1 second as running time.

- **initial vertex** $= 0$
- **number of nodes** $=$ 8, 10, 12, 14, 16
- **number of edges** $= ((n * (n-1))/2) - 5$
- **k** $= n * 5$



```
n = [8, 10, 12, 14, 16]
time(s) = [0.0038, 0.0193, 0.0218, 0.0599, 0.0817]
```

Figure: Exact running time results with respect to above parameters

Extra Success Criterion

4. When *k* is big and proportional to the number of edges, the results of the heuristic algorithm is at least %4 better than the exhaustive search algorithm.

I have accomplished this criterion **successfully** as we have seen in comparison.

$$
\begin{aligned}
&= \frac{V_{observed} - V_{true}}{V_{true}}\\
&= \frac{102.35 - 97.55}{97.55}\\
&= \frac{4.8}{97.55}\\
&= 4.9205535622758\%
\end{aligned}
$$

(a) When k is 7

$$
\begin{aligned}
&= \frac{V_{observed} - V_{true}}{V_{true}}\\
&= \frac{106.66 - 96.72}{96.72}\\
&= \frac{9.94}{96.72}\\
&= 10.277088502895\%
\end{aligned}
$$

(b) When k is 8

$$
\begin{aligned}
&= \frac{V_{observed} - V_{true}}{V_{true}}\\
&= \frac{101.9 - 95.1}{95.1}\\
&= \frac{6.8}{95.1}\\
&= 7.1503680336488\%
\end{aligned}
$$

(c) When k is 9

Figure: Difference Calculation of Exact Results

Conclusions

In this project, we have tried to solve the Minimum k-Chinese Postman Problem and we have created 2 different algorithms.

While making this project, I have experienced,

- How to make a literature search
- How to develop and optimize algorithms
- How to make complexity analysis and performance evaluation
- How to compare two different algorithms and how to interpret the results

Contributions

In literature, there are a reasonable number of algorithms for this problem but finding implemented one is nearly impossible because these types of algorithms are NP-hard. Therefore implementing such algorithms is very important.

By making this project, we can help the people who are interested in these types of problems. After publishing this project people can see how to solve these types of problems, how to make a complexity analysis, and how to make a comparison between different algorithms. Also, they can use these solutions and improve them.

Project Timeline

1st Meeting	Making literature research. Understanding the problem.
2nd Meeting	Continue literature research. Determine the steps of the heuristic algorithm. Start to implement heuristic algorithm.
3rd Meeting	Finish heuristic algorithm implementation. Make complexity analysis of heuristic algorithm. Test with different parameters. Start to implement exhaustive search algorithm.
4th Meeting	Finish exhaustive search algorithm impl. Make complexity analysis of e.s. algorithm. Making comparison and numerical evaluation. Create a graphical user interface.

References

[1] A. Hölscher, *A cycle-trade heuristic for the weighted k-chinese postman problem*, 2018.

[2] D. Ahr and G. Reinelt, *New heuristics and lower bounds for the min-max k-chinese postman problem*, 2002.

[3] www.stackoverflow.com/questions/12367801/finding-all-cycles-in-undirected-graphs.

[4] https://www.techiedelight.com/find-distinct-combinations-of-given-length.

[5] D. B. Johnson, *Finding all the elementary circuits of a directed graph*, 1975.

[6] https://igraph.org/.

[7] https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/.

Thank You