Barry University
School of Arts and Sciences
Department of Mathematics and Computer Science
Discrete Math Research Project

Comparison between a bidirectional nearest-neighbor algorithm and a brute force algorithm to find the best Hamilton circuit in a complete graph.

Researcher: **Luis Perez Cipollitti**
Fall semester 2022

Abstract

This research evaluates the application of a bidirectional nearest-neighbor algorithm to search for the optimal Hamilton circuit in a weighted complete graph. The result is then compared with the findings given by a brute-force search. This is used to understand the tradeoff in time complexity and efficiency between both algorithms. The research takes the approach of the traveling salesman problem, where the goal is to find the cheapest way to travel through a given set of cities visiting all of the cities only once. For this, data on the average domestic flight prices between cities from the U.S. Department of transportation is used to create the complete graph to be evaluated.

Mathematical subject classification: 68 Computer science
Keywords: Graph Theory, Network Analysis, Hamilton Circuits, Complete Graph, Time Complexity, Traveling Salesman Problem (TSP), Nearest-Neighbor Algorithm.

1. Introduction

Imagine a salesman has a set of 6 cities to visit and he is looking for the cheapest way to visit all of them only once, starting and ending in his home city. It is assumed that there are flights available between all the city-pairs. He would have **(n-1)!** route options, in this case 120, so he would have to calculate all the possibilities to find the cheapest one. This is the famous traveling salesman problem and what the salesman is looking for are called Hamilton circuits. In this example, the description is that of a complete graph since all the vertices are interconnected, the flight prices would be the edge weights between the vertices. Therefore, we can say that given a weighted complete graph ***G*** with **n** vertices, the number of possible Hamilton circuits in a such graph is given as **(n -1)!** This is a big problem because it implies a computational complexity of **O(n!),** which means that the time it takes to solve the algorithm grows at factorial speed as the number of vertices (**n)** increases. Factorial speed is one of the worst cases in computational complexity, computing such an algorithm for a large **n** is challenging even for the most advanced computers. Solving this problem in a brute force manner, that is calculating all the possible routes, is not time efficient and it is very hard to do for big **n** cases. Consequently, another approach is to use an algorithm that makes certain choices to optimize the computational time. For instance, a nearest-neighbor algorithm could be used to find a solution in linear time **O(n),** with the tradeoff that the solution might not be the best one (cheapest). Ideally, the solution would be close to the best result. The intention of this research is to measure the runtime and optimal solution for both approaches, brute force and nearest-neighbor, to understand the relationship of this tradeoff. For this, the traveling salesman problem will be recreated using Python, and the dataset for the flight prices used in this model comes from the historical collection of domestic prices from the U.S. Department of Transportation.

2. Background

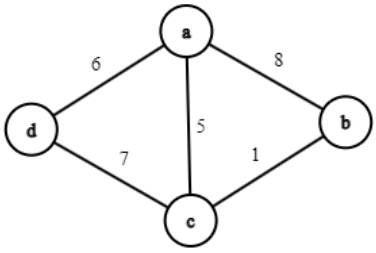
a. Graph Theory Fundamentals

A graph **G** is a set of vertices **V** and edges **E**. **"**In graph theory, the term ***graph*** refers to an object built from ***vertices*** and ***edges*** in the following way. A ***vertex*** in a graph is a node ... The ***edges*** of a graph connect pairs of vertices." (Hayes, Gravelle, Seideman, & Homp, 2019). We refer to a graph as:

$$G = (V, E)$$

The edges that connect two vertices can have a weight, that could represent a value like distance, for instance. Graphs with weighted edges are called weighted graphs.

Additionally, we can say two vertices are adjacent if they are connected by an edge. From this, we can build a square matrix called an adjacency matrix, where we represent the number of connections between the set of vertices. We can also build a weighted adjacency matrix using the weights as cell values for the matrix.



Weighted Adjacency Matrix

	a	b	c	d
a	0	8	5	6
b	8	0	1	0
c	5	1	0	7
d	6	0	7	0

b. Complete graphs

A complete graph is a graph where each vertex is connected by an edge with all the other vertices of the graph. It's perfectly interconnected.

c. Hamilton circuits

A Hamiltonian circuit "is a circuit that visits every vertex once with no repeats. Being a circuit, it must start and end at the same vertex." (Hayes, Gravelle, Seideman, & Homp, 2019). For a complete graph, the number of Hamilton circuits equals the number of permutations of all the vertices **n**, without counting the origin vertex. This result in the number of permutations of **n** minus **1.**

$$Number\ of\ Hamilton\ Circuits = (n-1)!$$

d. K Nearest-Neighbor algorithm (KNN)

This algorithm searches the closest neighbor of a vertex, then moves to this vertex where it repeats the search until it finds its goal or the options are exhausted. Its core mechanism is simple, "a naive approach to finding the k-nearest-neighbors for each query would sort the n candidates by distance and then return the first k." (Li & Amenta, 2015). The KNN algorithm is a greedy algorithm, "an algorithm follows the Greedy Design Principle if it makes a series of choices, and each choice is locally optimized; in other words, when viewed in isolation, that step is performed optimally." (Suri, 2019). This means that the overall result depends on the result at a local level every time the algorithm has to make a choice, in this case, it chooses to follow the shortest path even though it might not be the best decision in the long run.

3. Methodology

a. Data Collection

The first step was to collect data for domestic flights in the US. For this, the collection was done using the API of the U.S. Department of Transportation (DOT) to its database of Consumer Airfare Reports, "Table 1a - All U.S. Airport Pair Markets" (2022). The data was filtered leaving only the flights between the biggest cities per state, then a total of 28 cities were handpicked based on the number of connections between one city to the others (vertex degree). Afterward, the cities were used as vertices and the city-pair airfares as edges to create a graph, this graph is represented in a weighted adjacency matrix. Here the first issue was found, the graph was not a complete graph since the data offered by the Department of Transportation (DOT) have city-pair fares (edges) missing. To solve this, random numbers were generated for the missing city-pair fares. To minimize the distortion inserted by random numbers, the random generator used is based on a gaussian distribution from the sample of edge weights that already exist for the given vertex (city) to be computed.

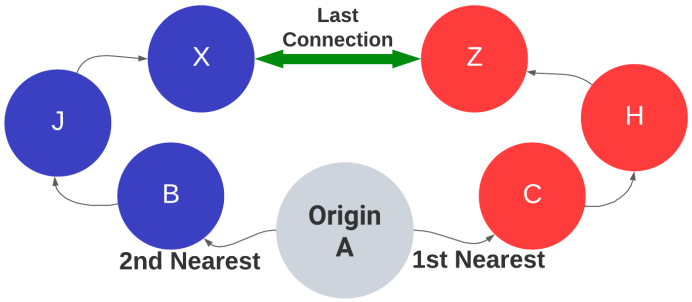
b. Brute Force Algorithm (BF)

The brute force algorithm is simple. It creates a list of all the vertices without the origin vertex, since we don't want to travel to our start point, and then It looks for all the permutations of this list, which are the Hamilton circuits. Subsequently, the algorithm proceeds to add all the weights for each permutation, generating the total weight for each Hamilton circuit. Finally, just choose the circuit with the minimum total weight.

c. Bidirectional Nearest-neighbor Algorithm (BNN)

The idea is to use a nearest-neighbor algorithm to look for the cheapest city connections. However, a problem that this search could have when looking for city routes in the US is that it could start exhausting all the cities close to the origin until it reaches the farthest city away, and the weight from the farthest city away to the origin could outweigh the optimization obtained in the previous cities, distorting our result. For instance, the algorithm could go from Miami-Chicago, Chicago-LA, LA-Alaska, and Alaska-Miami, where the travel back from Alaska could be very expensive. A more natural way to approach this problem could be Miami- Chicago, Chicago -Alaska, Alaska -LA, LA-Miami. This could be especially interesting when looking at large sets of cities.

Consequently, the approach taken in this research is to use a bidirectional nearest-neighbor algorithm, where on the first level of the search the question is: what are the two nearest vertices? Instead of just asking for one, let us call these vertices **B** and **C**. Then, a nearest-neighbor search link starts from both vertices. When all the vertices of the graph have been visited there will be two paths extending from **B** and **C** respectively. Let us say that **path 1** extends from **B** to **X** and **path 2** extends from **C** to **Z.** then the last step is to connect both pats finding the edge that joins **Z** and **X**. This is how the algorithm will calculate the best route.



d. Testing

The variable of interest for the complexity of this algorithm is the number of vertices **n.** Therefore, the different tests are classified based on the number of vertices (cities) per sample. From the main matrix of 28 cities collected from de DOT, the testing program creates 1000 samples of random combinations between cities, each sample is a weighted adjacency matrix of dimension **n** x **n.** The tests consist of running both algorithms for each sample and collecting the results to compare. For the brute force algorithm, the results collected include the optimal circuit

(cheapest), mean of all circuits weights, and runtime of the algorithm for each sample. On the other hand, for the BNN algorithm, the results collected were the optimal circuit and the runtime for each sample. The tests were run for each **n** in **n** = 4, 5, 6, 7, 8, 9, 10, 11.

4. Results

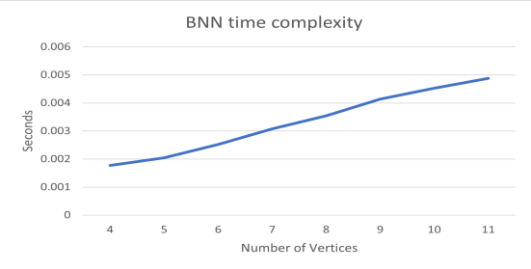
a. Time complexity

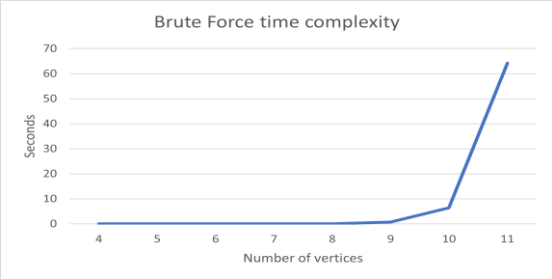
The time complexity of the brute force algorithm is factorial time **O(n!),** which is a terrible time complexity. Just as an example of that, while running the tests (1000 samples per test) for the different numbers of vertices **n**, the maximum **n** that could be tested was 11, beyond that point the time to calculate each test surpassed the 24 h of calculations. On the other hand, the bidirectional nearest-neighbor algorithm takes linear time **O(n).** This means that time complexity grows at a constant speed proportional to the size of input value **n.** The BNN is linear time because it only performs a set of operations for each row of the adjacency matrix. Hence if **n** grows because a vertex is added, the algorithm will only do one more set of operations. Therefore, the complexity grows constant and proportional to **n.** To evaluate this, the runtime was measured for both algorithms on each sample per test, then the runtime mean was calculated per test. The results are in table 1.

Mean runtime per test (seconds)			
n	BF	BNN	Difference
4	0.00188	0.00177	-0.00010
5	0.00093	0.00204	0.00111
6	0.00200	0.00252	0.00052
7	0.01226	0.00307	-0.00918
8	0.08618	0.00354	-0.08264
9	0.69645	0.00413	-0.69232
10	6.37777	0.00452	-6.37325
11	64.24745	0.00488	-64.24257

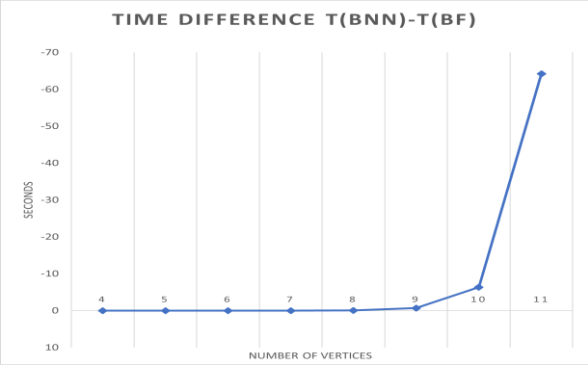
Table 1

In the following two charts, it's plotted the mean runtime per test for each algorithm, where it can be observed that the BNN follows a linear time complexity, and the BF follows a factorial time complexity.





As can be observed in the table, the runtime difference between the BNN minus the BF algorithms grows factorially, so as **n** grows the BNN algorithm drastically outperforms the BF approach. This is reflected in the runtime difference graph.



b. Result efficiency

Since the BNN algorithm looks for the nearest-neighbor it often misses the best path, the path it gets could not even be close to the best path, therefore we need to compare both solutions to understand how accurate is the result. For each sample on the tests, the results for each algorithm were collected. In the case of the brute force approach, the mean of the total weight for all the Hamilton circuits was also collected. To compare these results let's calculate the BNN **efficiency**, for this the following assumption is made: the best circuit from the brute force approach is the minimum value of the distribution of results, therefore between the BF result (minimum value) and the mean there is half of the range of the distribution of results for that sample. If this half range is doubled then the full **range** of the result distribution is obtained. This will tell us what is the range of possible values the BNN algorithm could've produced and how far away they are from the best value. Secondly, let's take the **result difference** between the BNN result minus the BF result, this difference will say how far away are both results. Afterward, let's take the ratio of the **result difference** divided by the **range** of possible values. This will place the BNN result in the scale of the range of possibilities and tell us how far away is the BNN result from the best result as a percentage. Lastly, to calculate the **efficiency**, we take the value of 1 minus the **ratio** result, to get how much the BNN result covered from the worst possible value until its current position. For instance, if the difference between the BNN result and the best result is 10, and the range of possibilities is 100, then the ratio is 0.1, which means the result is 10% away from the best result. Then, to get the efficiency, let's take 1 - 0.1 = 0.9, this will result in an efficiency of 90%. The formulas to calculate efficiency are the following:

$$\boldsymbol{Range = (Mean\ of\ Hamilton\ circuits - best\ hamilton\ circuit) * 2}$$

$$\boldsymbol{Difference = BNN\ result - BF\ result}$$

$$\boldsymbol{Difference\ ratio = \frac{Difference}{Range}}$$

$$\boldsymbol{\%\ Efficiency = (1 - Difference\ ratio) * 100}$$

Finally, after calculating all the rates of efficiency for each sample, let's take the average of these efficiency rates to get the overall efficiency for the test. In the case of the data of this research the results look as follows:

n	BF	Mean BF	BNN	Difference	Range	Efficiency
4	585.36	622.30	604.13	18.77	73.89	74.60%
5	708.52	777.22	738.09	29.57	137.41	78.48%
6	827.40	928.33	865.23	37.83	201.85	81.26%
7	955.66	1087.64	1001.87	46.21	263.95	82.49%
8	1076.81	1240.09	1130.67	53.86	326.56	83.51%
9	1199.82	1395.77	1261.26	61.44	391.91	84.32%
10	1323.05	1551.59	1390.76	67.71	457.10	85.19%
11	1432.70	1698.61	1509.12	76.42	531.82	85.63%
					Total	**81.93%**

As it can be noted, the average efficiency rate for the BNN algorithm in this research is about 81.93%. Something to note about this rate of efficiency is that depending on how the results for the Hamilton circuits are distributed this rate could be more or less accurate. For this research, the results were evenly distributed almost following a normal distribution, so this efficiency rate is a good measure of comparison.

5. Conclusion

All in all, the objective of this research was to offer a comparison frame to understand the tradeoff between computing speed and losing efficiency in finding the best result. As it can be observed the BNN algorithm performs drastically better time-wise, it has a linear time complexity which is significantly better than the brute force factorial time. However, some efficiency in finding the best result is lost, having an average of 81.93% efficiency for the BNN algorithm. Something to note is that the efficiency results highly correlate to how are the weights of the edges of a graph distributed, there could be data distributions that could bring the BNN efficiency down significantly. However, this test fits the data distribution for the data set used in this research.

A way to optimize the efficiency of this algorithm could be to apply the BNN algorithm to each vertex of the sample and select the best cycle, that way the opportunity of losing a short cycle is minimized. The verdict for this research is that the tradeoff of time and efficiency for this example could be worth it, depending on how sensitive to error is the system using this algorithm.

6. Bibliography

Hayes, A., Gravelle, S., Seideman, A., & Homp, M. (2019). *Contemporary Mathematics at Nebraska.* University of Nebraska.

Li, S., & Amenta, N. (2015). *Brute-Force k-Nearest Neig.* University of California.

Suri, S. (2019). Greedy Algorithms.

U.S. Department of Transportation. (2022). *Consumer Airfare Report: Table 1a - All U.S. Airport Pair Markets.* Office of the Secretary of Transportation.

Wainwright, T. (1997). SOLVING PROBLEMS INVOLVING HAMILTON CIRCUITS. In *The Journal of Mathematics and Science: Collaborative Explorations Volume 1 No 1.* Tulsa: The University ofTulsa,.