Project 3 Report

Sydney Pierce

CS570 Spring 2023

Part I: Brute Force Algorithm

The Hamiltonian path problem is an np-complete graph problem. The objective of the problem is to find a path that visits every vertex exactly once. To convert this np-complete decision problem into an np-hard optimization problem, I chose to optimize the algorithms to find the *shortest* Hamiltonian path of a given graph.

My brute force algorithm to find the shortest Hamiltonian path uses dynamic programming with a depth first search algorithm. The algorithm works by running all possible depth first searches from every node in the graph. When it finds a path of size V (the number of nodes in the graph), it stores this path in a vector matrix. Once it has iterated through all of the nodes in the graph, it calculates the weights of all the Hamiltonian paths and selects the minimum.

For a graph of ~100 edges, my brute force takes an unreasonable amount of time, and the heuristic is a better algorithm for these size graphs.

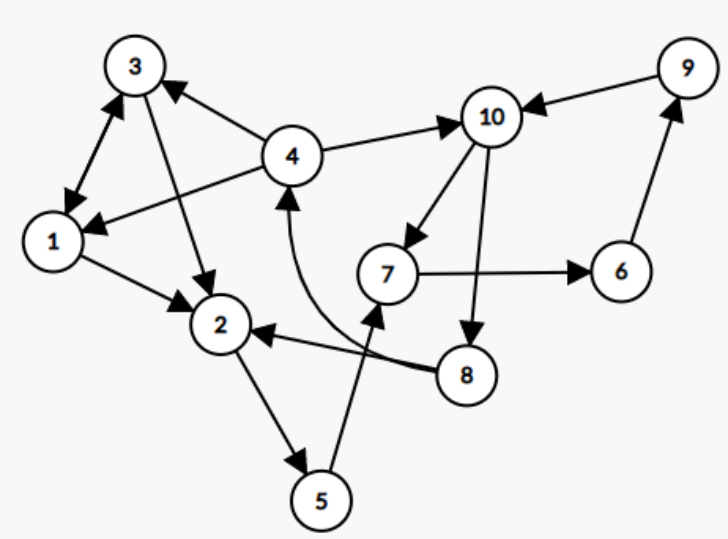
Part II: Heuristic Algorithm

My heuristic algorithm is based on the nearest neighbor heuristic for the travelling salesman problem. The travelling salesman problem seeks to find a shortest Hamiltonian cycle, so I used the general idea of the heuristic and applied to my problem. The algorithm iterates through each node in the graph, and uses each as a starting point exactly once. From each starting node, it checks the node's outgoing edges and selects the shortest one, adding that neighboring node to the path. Then, it checks this node's outgoing edges and selects its nearest neighboring node. This process continues until the path has V nodes or until it reaches a dead end. This weight of this path is then calculated and checked against the current minimum weight. If it is shorter, then it is updated ass the shortest path.

This heuristic is an approximation algorithm – it will not always find the optimal solution. Because it takes the greedy approach of selecting the smallest edge at its current point in execution, it may make a choice that is suboptimal in the long run. However, this approach makes it work for much larger graphs, which can make it much more useful than the brute force algorithm.

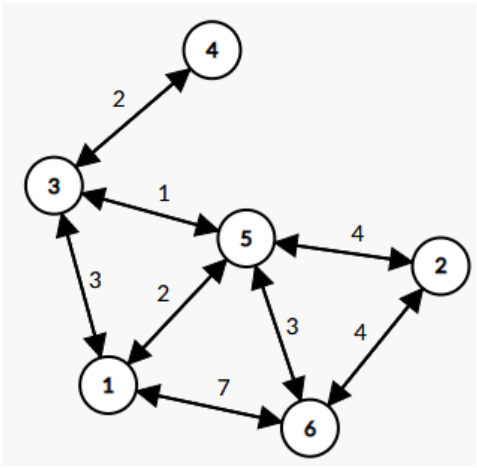
Part III: Examples for Brute Force and Heuristic

This first graph is an example that results in the same answer for both of my algorithms. It has 10 nodes, 16 edges, and is directed. My heuristic takes the path 1, 2, 5, 7, 6, 9, 10, 8, 4, 3 with a weight of 9. My brute force takes the path 1, 3, 2, 5, 7, 6, 9, 10, 8, 4, also with weight 9.



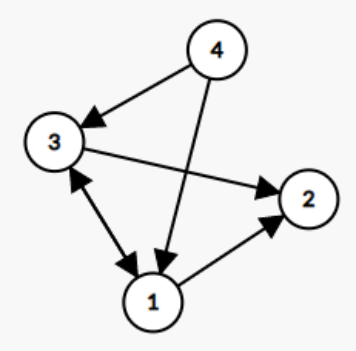

```
$ ./a.out graph1.dat
Heuristic solution:   1 2 5 7 6 9 10 8 4 3 with weight: 9
Brute Force Solution: 1 3 2 5 7 6 9 10 8 4 with weight: 9
```

The second graph is undirected with only 6 nodes and 16 edges. My heuristic get a suboptimal solution of 15, whereas my brute force gets the shortest path weight of 15. Although the heuristic does not get the optimal answer, it is only 1 unit longer.



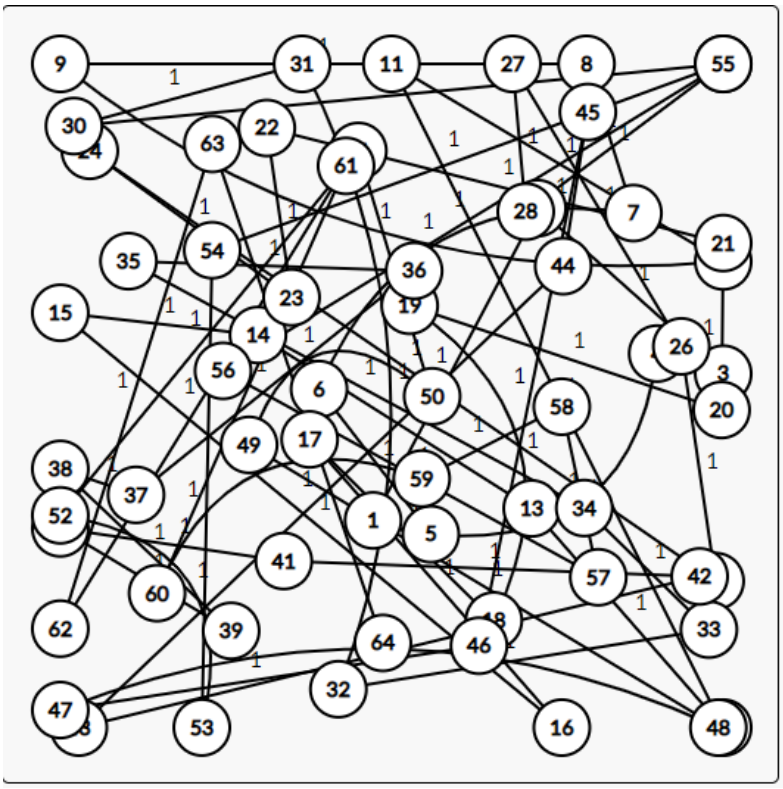

```
Heuristic solution:   4 3 1 5 2 6 with weight: 15
Brute Force Solution: 2 6 5 1 3 4 with weight: 14
```

The third graph is a very small, directed graph with only 4 nodes and 5 edges, but it does not work at all for my heuristic. My brute force gets a solution of path 4, 1, 3, 2 with a weight of 3.




```
The heuristic either fails for this graph or there is no Hamiltonian path.
Brute Force Solution: 4 1 3 2 with weight: 3
```

The fourth graph is undirected and has 64 nodes and 126 edges. This graph was made to test the upper bound of my brute force algorithm. My heuristic results in the optimal solution of weight 63, but my brute force was taking so long that I had to kill the process.



```
Heuristic solution:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 5
1 52 53 54 55 56 57 58 59 60 61 62 63 64 with weight: 63
```

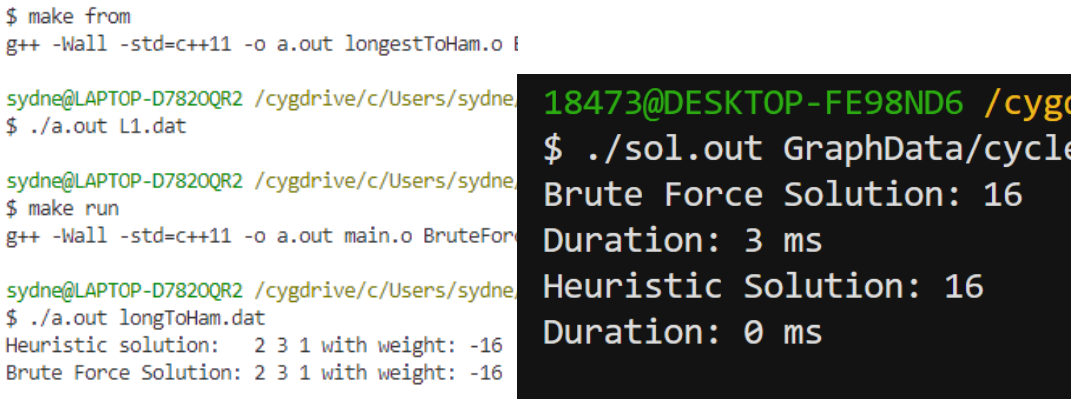
Part IV: Mappings

Mapping from Longest Path to Hamiltonian Path

My first mapping is a mapping from the longest path problem. The longest path problem seeks to find the largest weight path traversing every node in a graph exactly once, which has been shown to be an np-hard problem. My algorithm maps this problem to my shortest Hamiltonian path problem by altering a given graph file that is assumed to describe a graph with positive edge weights. It takes in the data, negates each edge weight, and outputs the altered data to a new file. Then, when this new file is provided as input to my brute force or heuristic algorithm, it outputs a path that is the *longest* path in the graph, and the edge weight of this path.

This mapping works because the negation of the edge weights causes the largest edge weights to be recognized by my algorithm as the smallest, which is what it wants to choose. Because of this, the edge weight of the path that is output by my algorithm will be the negated value of the longest path edge weight. The actual weight can be interpreted as the absolute value of the output value.

To test my mapping algorithm, I asked one of my classmates doing longest path for a few of her graphs and the results from them. The first (L1.dat) was a very simple three node graph. I ran it through my mapping algorithm, then ran the output graph from that through my heuristic and brute force. My results are shown in the left picture below, and hers the right. My mapping causes my output to be negative, but my algorithm resulted in the same magnitude as hers.



I did this same process on another graph with an interesting case. She was having trouble making her algorithms work with negative weights, so I tested her negative weight graph (L5.dat) with my mapping and got the following results. For comparison, her results are shown on the right, and the graph is shown below these. It can be seen that my algorithm worke, resulting in the negated value of the correct answer: -7.

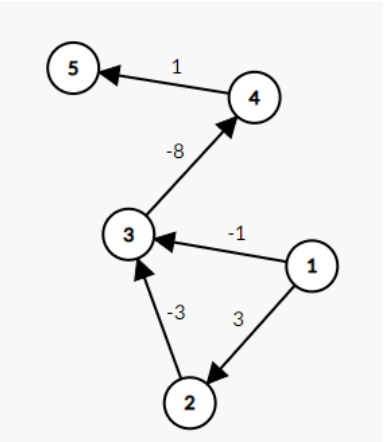
```
$ make from
g++ -Wall -std=c++11 -o a.out longestToHam.o Bru

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/sydne/Or
$ ./a.out L5.dat

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/sydne/Or
$ make run
g++ -Wall -std=c++11 -o a.out main.o BruteForce.

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/sydne/Or
$ ./a.out longToHam.dat
Heuristic solution:   1 2 3 4 5 with weight: 7
Brute Force Solution: 1 2 3 4 5 with weight: 7
```

```
18473@DESKTOP-FE98ND6 /cygd
$ ./sol.out GraphData/negat
Brute Force Solution: 1
Duration: 7 ms
Heuristic Solution: 1
Duration: 2 ms
```



Another interesting case is what occurs when there is not a Hamiltonian path, but there is a (non-Hamiltonian) cycle in the graph (L6.dat). She did not provide a screenshot for the results of this graph, but said that both her brute force and heuristic correctly returned that there was not an answer. However, when I ran it through my algorithm, only the heuristic algorithm worked. As shown below, my brute force got a very strange answer, showing me that graphs without Hamiltonian paths cause issues for my brute force.

```
$ ./a.out L6.dat

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/sydne/OneDrive/Desktop/Spring 2023/
$ make run
g++ -Wall -std=c++11 -o a.out main.o BruteForce.o Edge.o Graph.o Heuristic.

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/sydne/OneDrive/Desktop/Spring 2023/
$ ./a.out longToHam.dat
The heuristic either fails for this graph or there is no Hamiltonian path.
Brute Force Solution: with weight: 2147483647
```

Outside of this strange case, my mapping algorithm seems to find the answer to the longest path problem, proving that my mapping works as expected.

Mapping from Hamiltonian Path to Travelling Salesman

My second mapping is a mapping from shortest Hamiltonian path to the travelling salesman problem. As described previously, the travelling salesman problem seeks to find the shortest cycle in a graph that visits each node exactly once. My algorithm maps between these two problems by taking a graph file as input, and outputting this to a new file, as well as data for a new node that connects directly to every other node in the graph with an edge weight of 1. When this new file is used as input to a travelling salesman problem, it should output a cycle that represents the shortest cycle in the graph.

This mapping works because the added node guarantees that a cycle exits in the graph, a constraint that was not needed for my algorithm, and does not change the weighting of the graph because it has an equal weight to every single node. Because the output of a travelling salesman algorithm will show the shortest *cycle*, the output must be interpreted by removing the newly added node from the cycle to form a simple path. This path represents the shortest Hamiltonian path in the graph and solves my problem.

To test my mapping algorithm, I used the same four graphs on which I tested my brute force and heuristic. In each instance, I ran the graph through my mapping algorithm to generate the new graph. Then this graph was used as input to a travelling salesman heuristic algorithm.

Graph 1 had the following results. It took the path 1, 2, 5, 11, 7, 6, 9, 10, 8, 4, 3, 1. Because the original graph already had a Hamiltonian cycle, the mapping algorithm added a new one, and slightly altered the output from what I expected.

```
$ make to
g++ -Wall -std=c++11 -c hamPathToTSP.cpp
g++ -Wall -std=c++11 -c BruteForce.cpp
g++ -Wall -std=c++11 -c Edge.cpp
g++ -Wall -std=c++11 -c Graph.cpp
g++ -Wall -std=c++11 -c Heuristic.cpp
g++ -Wall -std=c++11 -o a.out hamPathToTSP.o Brut

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/sydne/One
$ ./a.out graph1.dat

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/sydne/One
$ g++ TSP.cpp

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/sydne/One
$ ./a.exe
11
```

I decided to run my original graph through the travelling salesman algorithm to see what would happen. The results are shown in the image below. This result is the answer I expected, as it is only 1 edge greater than the Hamiltonian path. This made me realize that if a graph already had a Hamiltonian cyclr, there was no need to run it through my mapping algorithm.

```
$ ./a.exe
1 -> 2
2 -> 5
5 -> 7
7 -> 6
6 -> 9
9 -> 10
10 -> 8
8 -> 4
4 -> 3
3 -> 1
10
```

Graph 2 had the following results. It traversed the path 1, 3, 5, 6, 2, 7, 4, 1 This is the correct answer, which indicates that the mapping worked as expected.

```
$ make to
g++ -Wall -std=c++11 -o a.out hamPathToTS

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/s
$ ./a.out graph2.dat

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/s
$ g++ TSP.cpp

sydne@LAPTOP-D782OQR2 /cygdrive/c/Users/s
$ ./a.exe
14
```

Graph 3 had the following results. Graph three did not work on my heuristic algorithm, so it is interesting to see this travelling salesman algorithm had no issue with it. The algorithm traversed the path 1, 3, 2, 5, 4, 1, and resulted in the expected answer.

```
$ make to
g++ -Wall -std=c++11 -o a.out ham

sydne@LAPTOP-D782OQR2 /cygdrive/c
$ ./a.out graph3.dat

sydne@LAPTOP-D782OQR2 /cygdrive/c
$ g++ TSP.cpp

sydne@LAPTOP-D782OQR2 /cygdrive/c
$ ./a.exe
5
```

Graph 4 resulted in the following. It took the same path as my heuristic algorithm, except fot the final node to make it a cycle, and ended up with the correct result.

```
sydne@LAPTOP-D782OQR2 /cygdrive/
$ make to
g++ -Wall -std=c++11 -o a.out ha

sydne@LAPTOP-D782OQR2 /cygdrive/
$ ./a.out graph4.dat

sydne@LAPTOP-D782OQR2 /cygdrive/
$ g++ TSP.cpp

sydne@LAPTOP-D782OQR2 /cygdrive/
$ ./a.exe
65
```

These results indicate that my mapping from the Hamiltonian path problem to the travelling salesman problem worked. Outside of the situation in which a Hamiltonian cycle already exists, and the extra node is unnecessary, the algorithm generated the output that I expected.

Part V: Conclusion

With the exception of a few specific cases, all of my algorithms work as desired. My heuristic does not always provide the optimal solution, including for the graphs mapped from longest path, but it works on significantly larger graphs, which makes it worth the sacrifice of accuracy. My mapping from the longest path worked for all but one of the graphs I tested, and it even got better answers than my classmate's algorithm at times. My mapping to travelling salesman was also successful, although sometimes the TSP heuristic produced answers slightly better than mine, indicating they may have made a better heuristic. Overall, my algorithms seemed to be successful and function as intended.