# Using Hungarian Algorithm to Estimate the Traveling Salesperson Problem

Jonathan Armknecht
Dallin Goodsell
Tel Dyer
Pearce Keesling

13 December 2018
CS 312-002
Professor Dan Ventura
BYU, Computer Science

**Abstract**

The Hungarian algorithm is a matrix based algorithm that uses row and column reduction to estimate the best possible overall cost to satisfy the matrix [1, 2]. This is applied to the Travelling Salesperson problem by treating cities as the rows and columns of the matrix and reducing to find the best path. This method is then compared with a random selection, a greedy algorithm, and a branch and bound algorithm. The comparison focuses on both accuracy of the final result and on the time taken to find that result.

**Introduction**

The Traveling Salesperson Problem (TSP) has been proven to be NP-complete. This means that there is no known polynomial time solution that can find the optimal solution. Because it is unreasonable to always solve the TSP using a non-polynomial algorithm (traveling to every airport in the world for example) there have been many polynomial approximations for the problem. The goal of these algorithms is to run quickly but also reduce the error bound for the estimate.

Three solutions are being compared that attempt to estimate a best solution for the traveling salesperson problem. The greedy algorithm is used as a baseline for speed, and the branch and bound as a baseline for correctness (as it is guaranteed to eventually find a correct answer). The simple random solution provides a comparison to prove that our solutions are better than the simplest answer of guessing and checking.

**Greedy Algorithm**

The greedy solver has the best run time of the different algorithms, clocking in at $O(n^2)$. This is typical of greedy algorithms that are often used for their small run-time bounds. The algorithm starts at a city and attempts to find the shortest path by selecting the shortest hop at each node. This is then repeated from each city and the best answer is selected. The run time is $O(n^2)$ because each start city only needs to check each other city one time, so n cities are checking (n-1) other cities.

The shortcoming of this algorithm is that it is not guaranteed to find the optimal solution. In the hardest difficulty it is also not guaranteed to find a solution to begin with (although in practice it reliably will). Based on experimentation it was found that for the average case this algorithm performed much better than the simple random algorithm. It also lived up to performance expectations, proving to be the only algorithm capable of solving 200 cities (and only taking 13 seconds to do so).

**Simple Random Algorithm**

The simple random algorithm does as the name implies, choosing cities at random and checking to see if that path creates a valid route. It makes no attempt to optimize for shortest path. The big-O of this algorithm changes depending on the problem difficulty.

For the easy version of the problem, the simple random algorithm runs in O(n) because all it needs to do is select n cities and verify that they make a route. Technically the verification part of this test is unnecessary because every city connects to

every other city so a hamiltonian route is possible no matter the order of the cities.

For the hard version of this problem the runtime is significantly worse. Since there are unidirectional paths it is possible that if there is only one valid route, every single permutation of the cities must be tried before that valid route is found. This means that the complexity of the algorithm is O(n!) which is substantially worse than any of the other algorithms. The cost of creating and testing routes is so small that at low numbers of cities the algorithm performs very well, even within an order of magnitude of the greedy algorithm. However the runtime quickly explodes, causing 100 cities to take too long to compute.

**Branch and Bound**

The only algorithm of the ones used in this experiment which is guaranteed to find an optimal solution is the Branch and Bound algorithm. As such, it has a non-polynomial time solution. This algorithm is similar to the simple random algorithm in that it can potentially have to check every possible route in order to find the best route. The advantage of this solution is that it attempts to order the routes that it checks in such a way that it can eliminate groups of them without needing to check them.

The way that it does this is by traversing a tree of all possible routes and keeping track of a best route so far. Each node is checked based on the best possible solution it could produce. If it is worse than the best solution so far, then that node and all of its children are ignored because the best solution does not exist down that path.

The less defined detail of the branch and bound algorithm is how to decide the order in which to check the nodes. This is important because finding better solutions earlier means that more of the tree can be cut. This metric for ordering nodes can be different for different implementations. In this experiment a combination of best possible path and the number of cities in the path was used. This was to encourage reaching leaf nodes in the tree while also choosing nodes that have promising results.

The unfortunate flaw in this algorithm is that no matter what metric is used for ordering the nodes, the overall complexity of this algorithm remains at $O(n^3 * (n-1)!)$ in the worst case. This is because no metric has been found which guarantees that significant sections of the tree will be pruned. If no sections are pruned then just like the simple random algorithm, all possible routes might have to be checked. To add to that complexity, it is not free to order and prune the nodes. This ordering and pruning is what adds the $n^3$ to the complexity. That being said, the pruning can drastically reduce the runtime in the average case because many of the nodes do not need to be checked.

**Hungarian**

While the Branch and Bound algorithm has a runtime of $O(n^3(n-1)!)$ the Hungarian algorithm solves the assignment problem on average in polynomial time $O(n^4)$ [2]. However the big-O is O(n(n-1)!) since this is the worst case scenario. The assignment problem is a combinatorial optimization problem and a solution will assign items from one set to items in another set in a one-to-one manner with the least cost [3].

The algorithm can also be used for the traveling salesperson problem because each route is one city assigned to another city in the same one-to-one manner as the assignment problem [4]. The Hungarian algorithm also shares some commonality with the matrix reduction in the Branch and Bound algorithm which is a reason why we chose to use it.

The time complexity and the similarity to the Branch and Bound reduction made the Hungarian algorithm a good candidate for another algorithm to compare the others with. The downside of this algorithm is that while reducing the matrix, it can make arbitrary decisions between one route and another. This means that while it will still produce a solution with a valid tour, the solution isn't guaranteed to be optimal for the problem.

**How The Hungarian Algorithm Works [3, 5]**

1. The Hungarian algorithm starts with the same cost matrix as the Branch and Bound algorithm. The matrix has n rows and n columns where n is the number of cities. The numbers in the matrix are the costs from traveling from i to j, i being the city represented by the row position, and j being the city represented by the column position.
2. The matrix is then reduced by subtracting the lowest value of each row from each element in the row. The same is done for the columns.
3. With the reduced matrix, the algorithm finds the smallest set S of rows and columns that contain all the zeros in the matrix. If the size of this set is equal to the number of cities (n) then the matrix is fully reduced and step five is executed.
4. If the size of S is smaller than n then the matrix is reduced again as follows. The smallest cost element not contained in a row or column in S is subtracted from all elements of the matrix not in a row or column in S. The values are then added to all elements in a row or column in S but not in a row *and* a column in S. The smallest cost element is then subtracted from every element in the matrix.
5. Once the size of S is equal to n the matrix is fully reduced and routes are found and added to the tour in the following way. Each row is searched for an element with a value of zero. Once that element is found the corresponding path (i, j) is added to the tour and the row for the city corresponding to the column j is treated in the same way searching for a zero then adding the route to the tour.

The downfall of the Hungarian algorithm is that the last step can cause the algorithm to take a long time to produce a solution. In practice step five will stop once a tour is found so on average step five is O(n). In the worst case though, step five can try all the possibilities for making a tour ending up in O(n(n-1)!). This made the algorithm run relatively quickly for some larger problems, and very slow for others.

| | Random | | Greedy | | Branch and Bound | | | Hungarian | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # Cities | Time (sec) | Path Length | Path Length | % Improve | Time (sec) | Path Length | % Improve | Time (sec) | Path Length | % Improve |
| 15 | 0.00 | 18375.6 | 11150 | 39.31% | 28.18 | 9423 | 15.49% | .065 | 9725.2 | 12.78% |
| 30 | .04 | 40728 | 17154 | 57.88% | TB | TB | TB | TB | TB | TB |
| 60 | 17.87 | 76109 | 25221 | 66.86% | TB | TB | TB | TB | TB | TB |
| 100 | TB | TB | 35892 | --- | TB | TB | TB | TB | TB | TB |
| 200 | TB | TB | 56083 | --- | TB | TB | TB | TB | TB | TB |

**Empirical Data**

The data above matches what has been discussed in the sections above about the Random, Greedy, Branch and Bound, and Hungarian algorithms. At first glance the fact that the random algorithm started to take more than ten minutes after 100 cities might seem surprising. While the idea of the algorithm is simple, when the scenario is “hard” the random method has to try every combination of paths to find one that makes a valid route. The greedy algorithm, on the other hand, ran fast even up to 200. The Branch and Bound algorithm stopped being useful early on as well as the Hungarian algorithm. The reason the Hungarian algorithm also stops being useful is that the more cities that are added, the more likely it is that the last step of constructing the path will take O(n(n-1)!). Some larger trials ran fast with the Hungarian but as the number of cities increased this happened less and less.

The table below shows how large of a problem the Hungarian can handle in a reasonable amount of time. The algorithm either completed relatively quickly or ran very slow without many trials with results somewhere in the middle. As the problem size increases, the algorithm runs very slow more often until it is unlikely that it will complete quickly. The table below shows runtimes for four different problem sizes with five trials each.

| | n=17 | n=20 | n=25 | n=28 |
|---|---|---|---|---|
| T1 | 2.72 | 9.60 | 54.53 | TB |
| T2 | 14.86 | TB | TB | TB |
| T3 | 6.48 | 6.85 | TB | TB |
| T4 | 4.44 | 329.36 | 171.11 | TB |
| T5 | 5.65 | TB | 109.98 | TB |

**Future Work**

In our implementation of the Hungarian algorithm, we know that the solution isn't always optimal but the algorithm doesn't indicate when it is and when it isn't. The nature of this algorithm would make it difficult to tell definitely whether the solution is optimal or not but the next easiest thing to do would be to indicate that the solution may not be optimal whenever the algorithm is confronted with an arbitrary choice. Although this is better than the current implementation the algorithm isn't that useful in real situations unless the user can know how close the solution is to optimal and the margin of error. If this was the case this algorithm would be a good one to use for smaller problems that need to have more accurate solutions than a greedy algorithm but run faster than the Branch and Bound method. For larger problems the part of the algorithm that finds the set of rows and columns that contain all zeros would need to be improved. The current implementation is recursive and takes too much time and space to be feasible for problems much bigger than Branch and Bound can handle. One possibility to fix this would be to use threads instead of recursion. Threads would decrease the amount of work done to find a viable tour by splitting that work up and thereby decreasing overall runtime.

**References**


[1] WikiHow. (n.d.). Retrieved November 16, 2018, from https://www.wikihow.com/Use-the-Hungarian-Algorithm

[2] Hungarian algorithm. (n.d.). Retrieved November 17, 2018, from https://en.wikipedia.org/wiki/Hungarian_algorithm

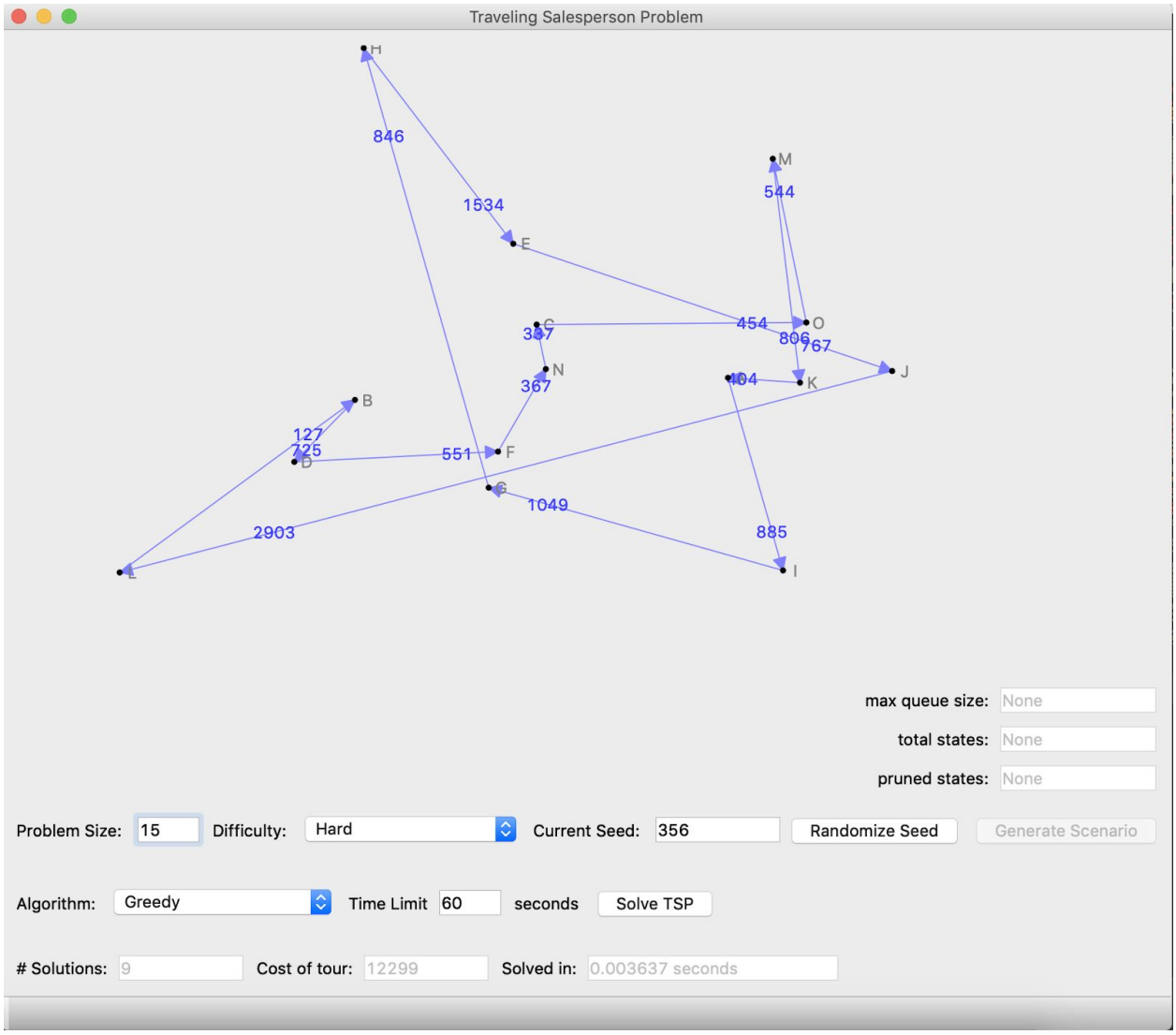
[3] The Hungarian Algorithm for the Assignment Problem. (n.d.). Retrieved November 26, 2018, from http://www.mathcs.emory.edu/~cheung/Courses/323/Syllabus/Assignment/algorithm.html

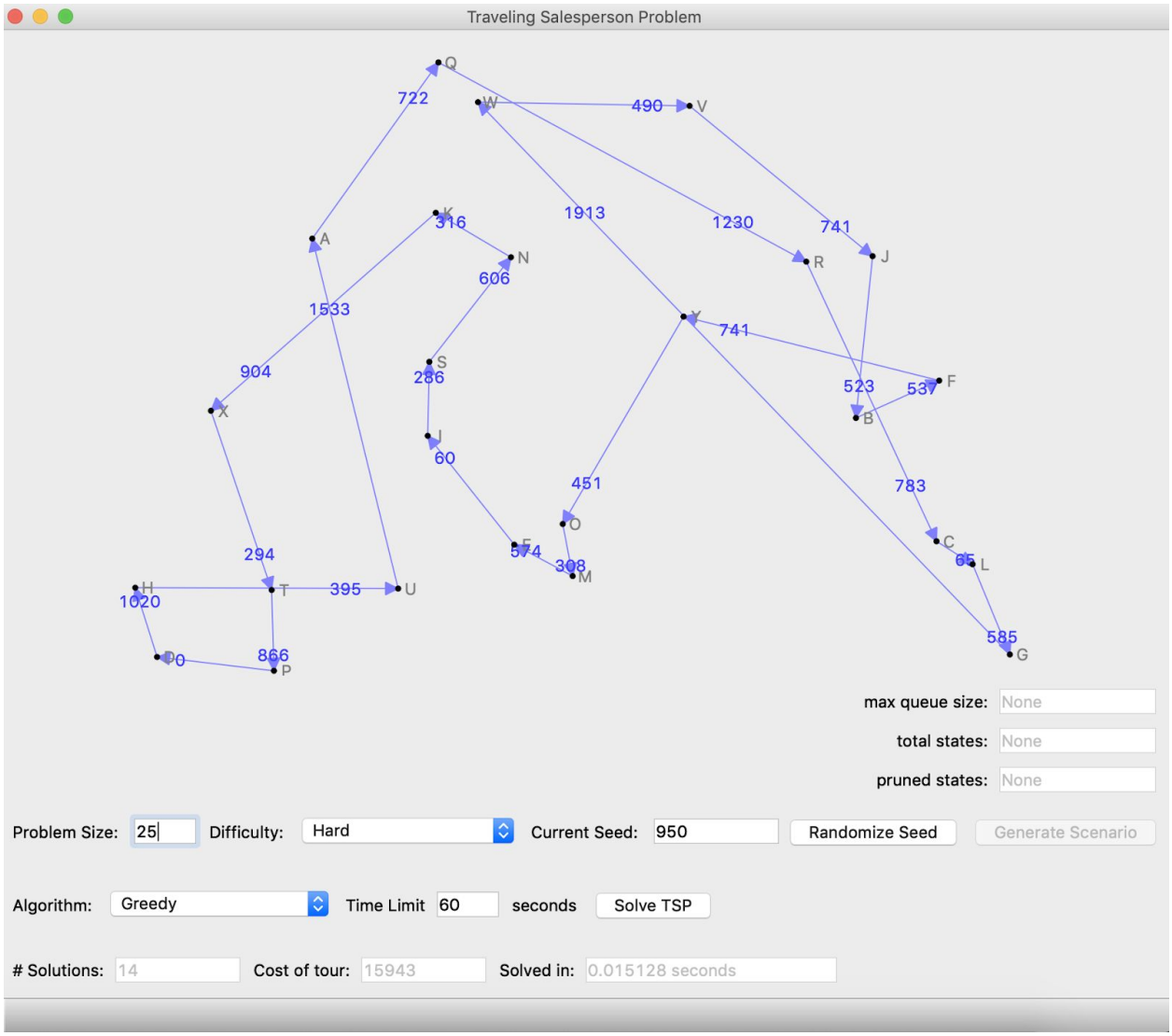
[4] Kuhn, H. W. (2010). The Hungarian method for the assignment problem. In 50 Years of Integer Programming 1958-2008 (pp. 29-47). Springer, Berlin, Heidelberg.

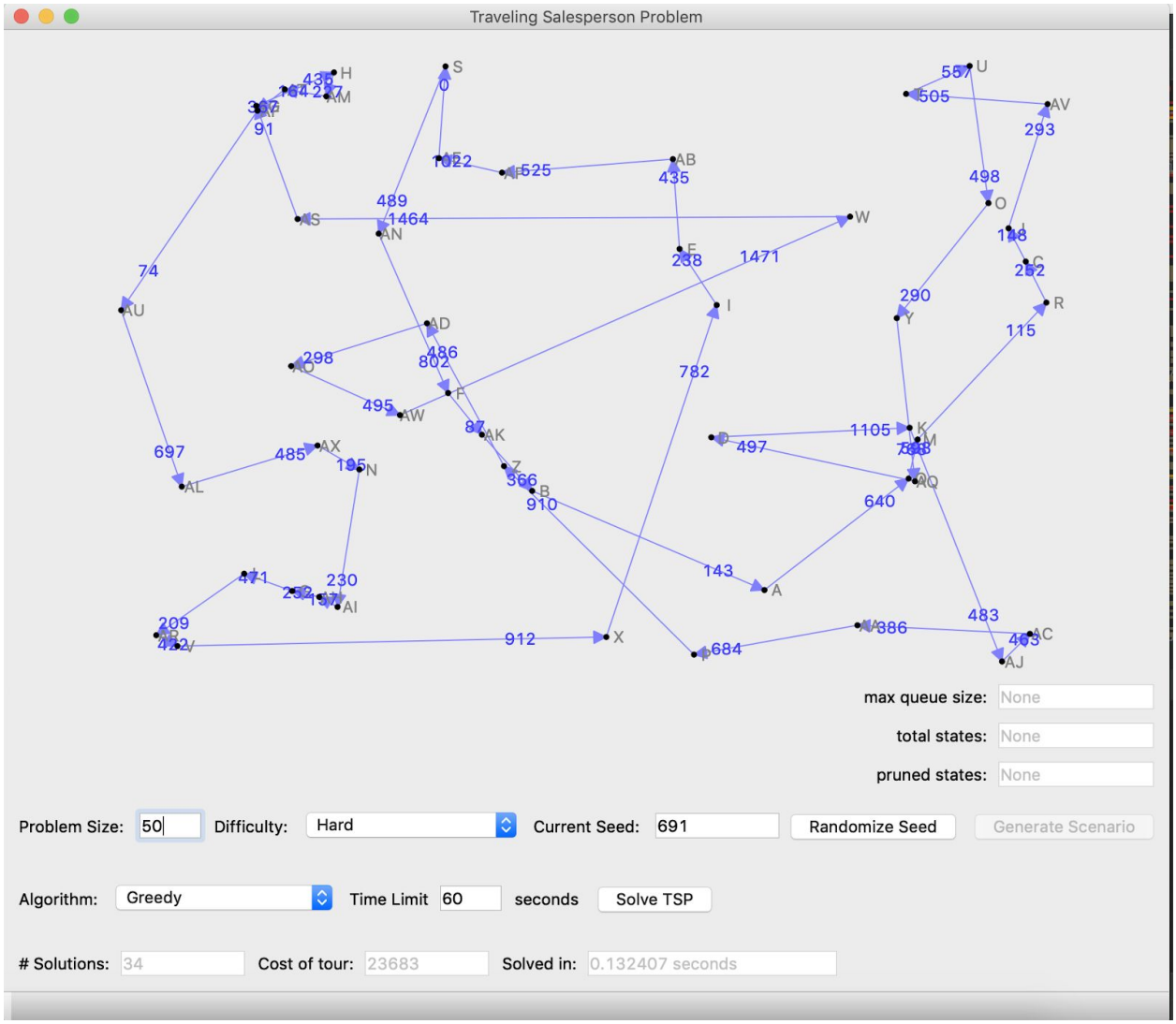
[5] Hungarian algorithm. (n.d.). Retrieved November 17, 2018, from https://en.wikipedia.org/wiki/Hungarian_algorithm

## Appendix

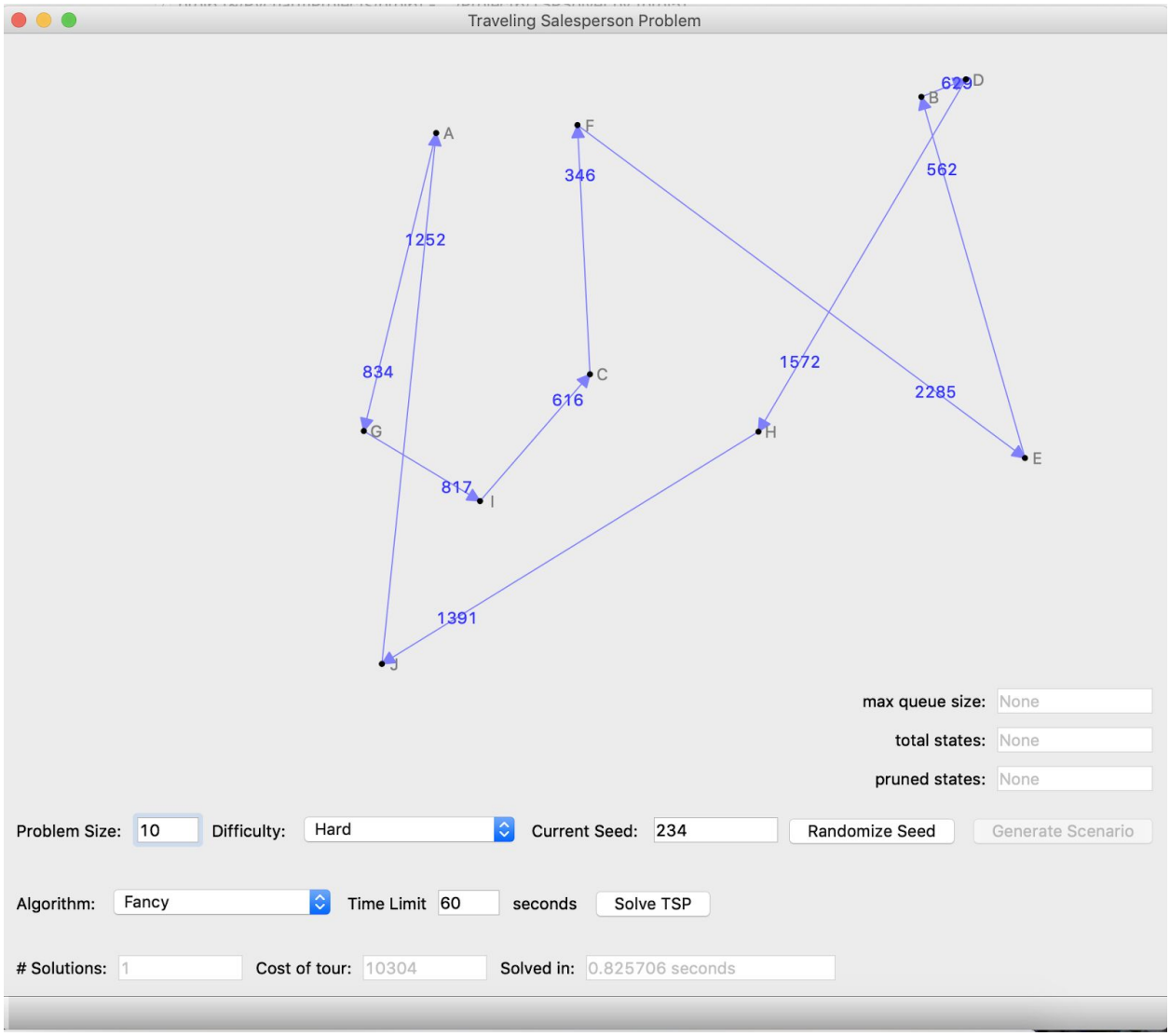
*Greedy Screenshots*

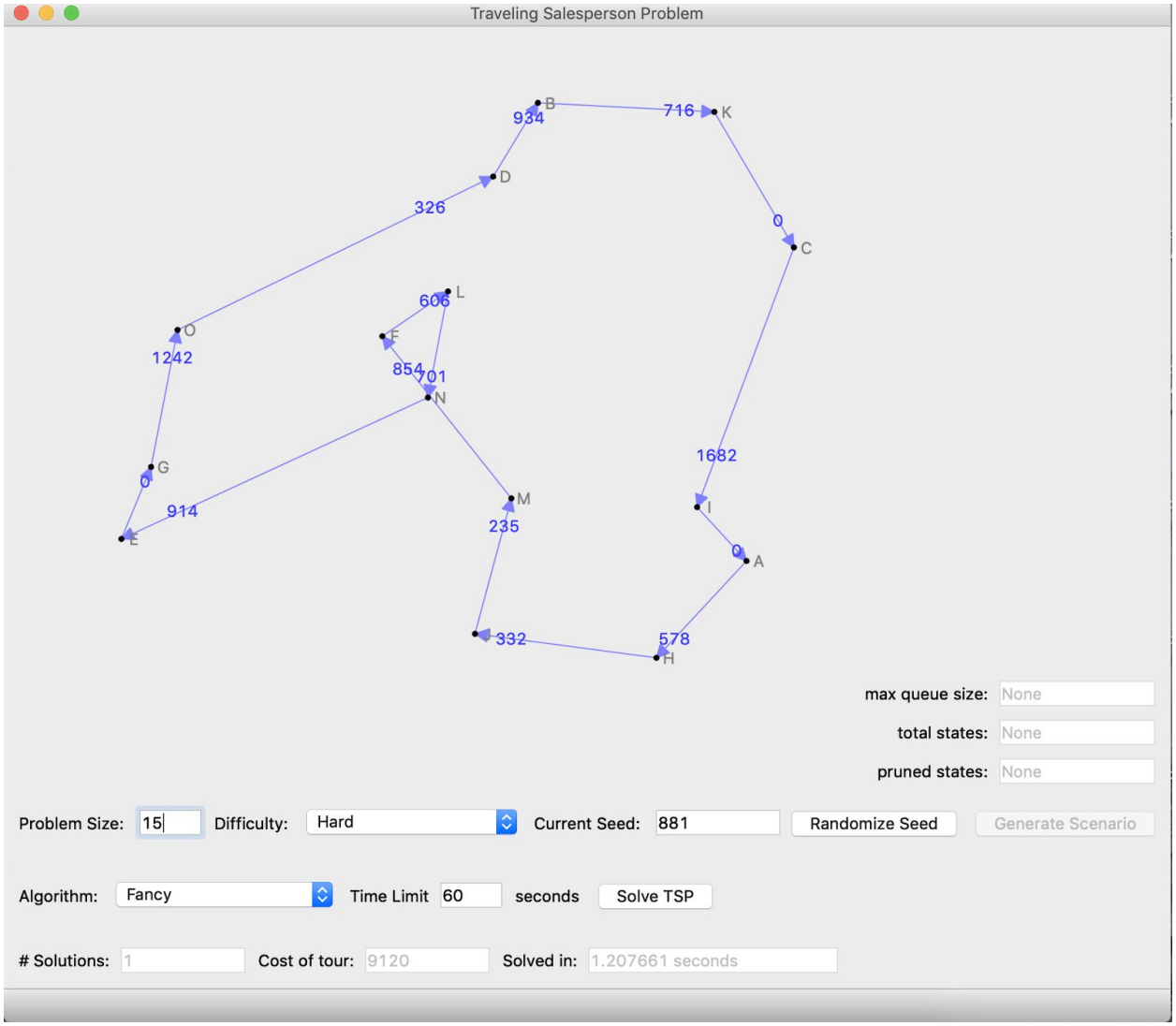


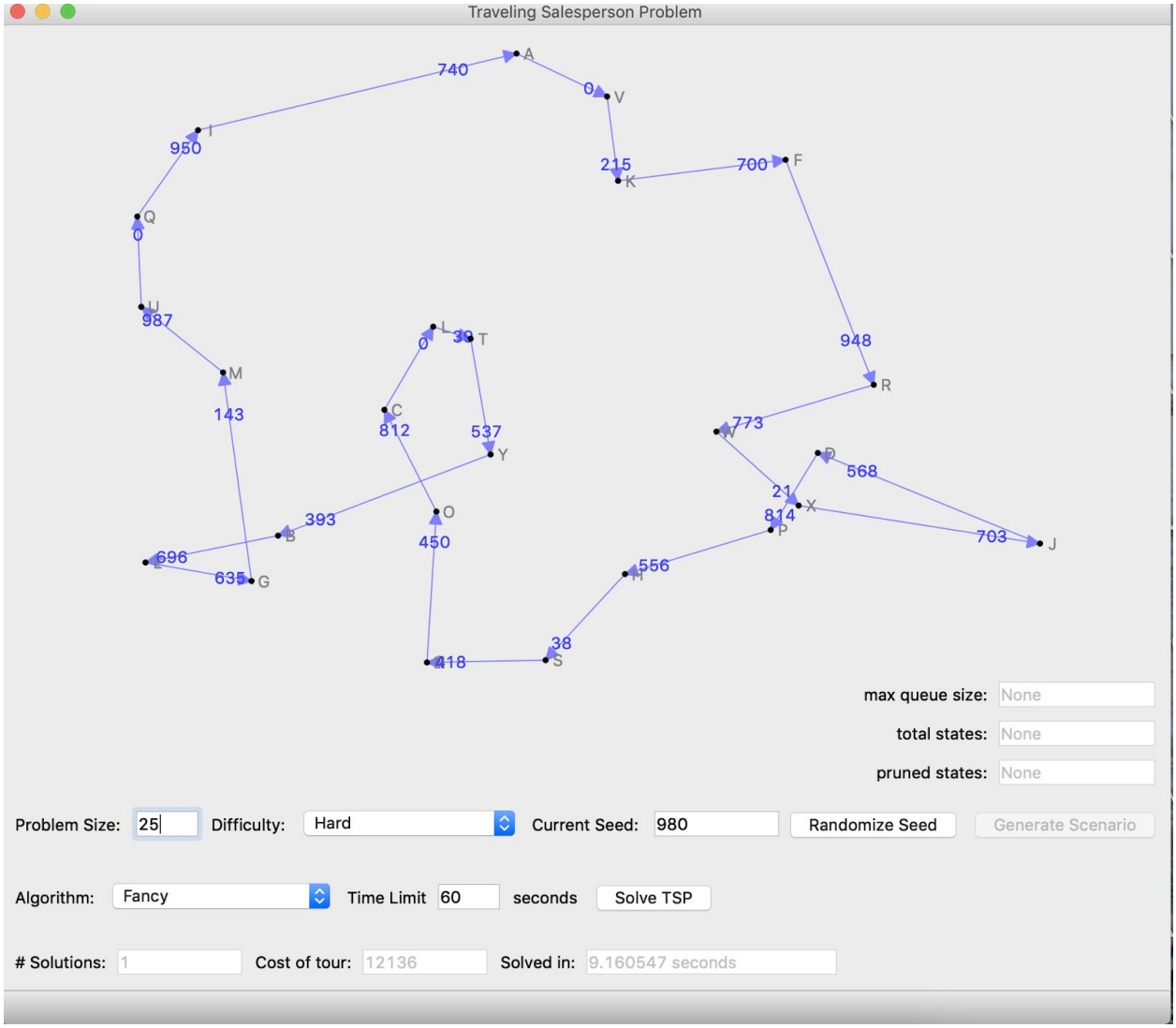

Traveling Salesperson Problem
Q
W
V
A
K
N
R
J
Y
S
F
X
B
J
O
C
L
E
M
H
T
U
P
G
722
490
316
1913
1230
741
606
1533
741
904
286
523
537
60
451
783
294
574
308
65
1020
395
866
585
max queue size:
None
total states:
None
pruned states:
None
Problem Size:
25
Difficulty:
Hard
Current Seed:
950
Randomize Seed
Generate Scenario
Algorithm:
Greedy
Time Limit
60
seconds
Solve TSP
# Solutions:
14
Cost of tour:
15943
Solved in:
0.015128 seconds


Traveling Salesperson Problem
max queue size: None
total states: None
pruned states: None
Problem Size: 50
Difficulty: Hard
Current Seed: 691
Randomize Seed
Generate Scenario
Algorithm: Greedy
Time Limit 60 seconds
Solve TSP
# Solutions: 34
Cost of tour: 23683
Solved in: 0.132407 seconds

*Hungarian Screenshots*




Traveling Salesperson Problem
B
934
716
K
D
326
0
C
L
606
O
1242
F
854
701
N
G
0
914
E
M
235
1682
I
0
A
332
578
H
max queue size:
None
total states:
None
pruned states:
None
Problem Size:
15
Difficulty:
Hard
Current Seed:
881
Randomize Seed
Generate Scenario
Algorithm:
Fancy
Time Limit
60
seconds
Solve TSP
# Solutions:
1
Cost of tour:
9120
Solved in:
1.207661 seconds


Traveling Salesperson Problem
A
740
0
V
I
950
215
K
700
F
Q
0
U
987
L
0
30
T
948
M
143
C
812
537
Y
R
773
W
D
568
21
814
X
P
703
J
O
450
393
B
696
L
635
G
556
H
38
S
418
max queue size:
None
total states:
None
pruned states:
None
Problem Size:
25
Difficulty:
Hard
Current Seed:
980
Randomize Seed
Generate Scenario
Algorithm:
Fancy
Time Limit
60
seconds
Solve TSP
# Solutions:
1
Cost of tour:
12136
Solved in:
9.160547 seconds