Project 6: TSP Using Hungarian Algorithm

Chengbo Xing, Bradley Tyler, Egan Schafer

CS 4412
Professor Paul Bodily
4/30/2020

Abstract

The Traveling Salesman problem (TSP) is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Many algorithms have been proposed for this problem with varying benefits in time and space complexity and optimal paths [1]. We'll show the benefits of the Hungarian Method to the Traveling Salesman Problem in this project. 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 [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 default random, 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 Salesman Problem (TSP) is a well studied problem in computer science. The problem is: given N number of cities, with corresponding distances between the cities, what is the shortest path that will visit each N city once and return to the originating city [1]. In combinatorial optimization it is considered an NP-hard problem because the algorithm itself cannot be solved in polynomial time. It also falls under the category of NP-complete because it maps to every other NP-complete problem, and it can be verified in polynomial time [6].

There are four different algorithms being compared that attempt to estimate a best solution for the TSP. The greedy algorithm is used as a baseline for speed, and the branch and bound algorithm as a baseline for correctness (as it is guaranteed to eventually find a correct answer). The default random solution provides a comparison to prove that our solutions are better than the simplest answer of guessing and checking, and the Hungarian algorithm will be our own algorithm to solve the TSP. The greedy algorithm will be analyzed briefly and the Hungarian algorithm will be analyzed in detail. The results produced by all four algorithms will be compared in the end.

Algorithm Explanation

1. Greedy algorithm

The greedy algorithm is a simple heuristic solving the TSP. It is not guaranteed to get a good solution or even find a solution at all. The algorithm starts at some city and from that city takes the lowest cost edge to a city not covered so far. This then repeats until either we get back to the starting city, or there are no edges leading to uncovered cities. In order to increase the odds of finding a solution, and finding a good solution, we run the algorithm starting at every city. Let n be the number of cities. We visit each city once, and at each city check order n possible edges to find the minimum. Thus it is O(n^2).

2. Hungarian algorithm

2.1 Background

The Hungarian method is a combinatorial optimization algorithm that solves the assignment problem in polynomial time and which anticipated later primal-dual methods. It was developed and published in 1955 by Harold Kuhn, who gave the name

"Hungarian method" because the algorithm was largely based on the earlier works of two Hungarian mathematicians: Dénes Kőnig and Jenő Egerváry [2].

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 [2]. 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.

There are actually many algorithms that can be used to solve the TSP, and each of them have their own advantages and disadvantages. Therefore, among many options, we chose the Hungarian algorithm that we are not familiar with. We are looking to grow our knowledge in something we know nothing about. We think this is an adventure, it provokes our challenges. We enjoy this process very much.

2.2 How the Hungarian algorithm works? [4, 5]

The algorithm begins by creating a matrix that we are all familiar with, a reduced cost matrix. This method iterates through the rows and columns, subtracting the minimum value from each row and column until each row and column has at least one value of zero. Next, we are required to 'draw a line' through the rows and columns of the matrix (row/column scanning). In our implementation, we changed the values in each row or column into the infinity value. After obtaining the reduced matrix, we search each row for exactly one zero, mark the city, and draw a vertical line (mark the values as infinity) through the column that the marked city is in. Next we search the columns for exactly one zero, mark the city and draw a horizontal line (mark the values as infinity) through the row that the marked city is in. Next, we take the minimum of the remaining 'sub matrix', or remaining unmarked cells, and subtract that value from each remaining value. Finally, we take that same minimum value leftover from the 'sub-matrix' or unmarked cells, and add it to the intersecting points of the marked matrix. Repeat this process until the number of marked cities equals the number of total cities. After multiple iterations, we have a matrix with selected cells that equal the total number of cities. Through analysis of those cells, we can obtain a path which may be the optimal path for the traveling salesman problem. We check whether or not it is the optimal path by comparing it to other solutions so far.

2.3 Complexity analysis [2]

The optimal complexity of the Hungarian method regarding the traveling salesman problem is O(n^3). In our implementation, the bulk of the algorithm is handled in the row_col_scan() method which handles the 'marking' of the selectected rows and columns. This selection requires a scanning of each row and each column, multiple times within a while loop until the conditions are met; this requires a total time of O(n^4). An additional caveat to our implementation is the function used to find the optimal tour. We used a somewhat naive approach in order to discover the optimal tour. The approach has a worst case time complexity of O(n!) under the circumstances that the heap is filled to capacity. However, under regular circumstances, the time complexity of using the heap will be O(n).

Analysis of results

The table below shows experimental results for the four algorithms. The algorithms

are compared based on computation time (measured in seconds), path length (cost of the best tour), and the percentage of improvement over the results of the greedy algorithm (in the case of the greedy algorithm the table shows its improvement over the random algorithm). The table below shows runtimes for seven different problem sizes with five trials each.

	Random		Greedy		Branch and Bound			Hungarian Algorithm		
# Cities	Time (sec)	Path Length	Path Length	% of Random	Time (sec)	Path Length	% of Greedy	Time (sec)	Path Length	% of Greedy
10	0.0000	13167	8999	68.34	0.1941	7770	86.34	0.0108	8869	98.56
15	0.0086	21682	11380	52.48	5.2553	9664	84.92	0.9440	10911	95.88
20	0.0036	25814	12523	48.51	245.17	11129	88.87	19.009	11621	92.79
30	0.0372	39295	17260	43.92	TB	TB	---	TB	TB	---
60	27.989	75533	25193	33.35	TB	TB	---	TB	TB	---
100	TB	TB	35170	---	TB	TB	---	TB	TB	---
200	TB	TB	55448	---	TB	TB	---	TB	TB	---

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, but it was shown to not produce the optimal solution and in fact will effectively never produce the optimal solution on large problems. This is where the advantage of the Hungarian Method shone forth, it is faster than the Branch and Bound and produces a more optimal path than greedy. 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 the worst case. Some larger trials ran fast with the Hungarian but as the number of cities increased this happened less and less.

Future Work

The function we have which finds the optimal tour is generally naive. If we were to spend more time and ingenuity we could potentially solve the problem in a more elegant way. For example, rather than using an endless heap that results in n factorial time, we could implement another data structure. Regardless of the method, improving upon the section of our algorithm that can potentially take n factorial time would vastly improve our program.

References

[1] Travelling salesman problem. (n.d.). Retrieved April 27, 2020, from https://en.wikipedia.org/wiki/Travelling_salesman_problem

[2] Hungarian algorithm. (n.d.). Retrieved April 27, 2020, from https://en.wikipedia.org/wiki/Hungarian_algorithm

[3] Traveling Salesman problem using Hungarian method example https://cbom.atozmath.com/example/CBOM/Assignment.aspx?he=e&q=tsh

[4] Travelling Salesman problem in Operations Research using Hungarian Method: by kauserwise https://www.youtube.com/watch?v=k3I2eThAErc&feature=youtu.be

[5] Using Matrices and Hungarian Method to Solve the Traveling Salesman Problem. (Bryan Couto) https://digitalcommons.salemstate.edu/cgi/viewcontent.cgi?article=1169&context=honors_theses

[6] NP-completeness. (n.d.). Retrieved April 30, 2020, from https://en.wikipedia.org/wiki/NP-completeness

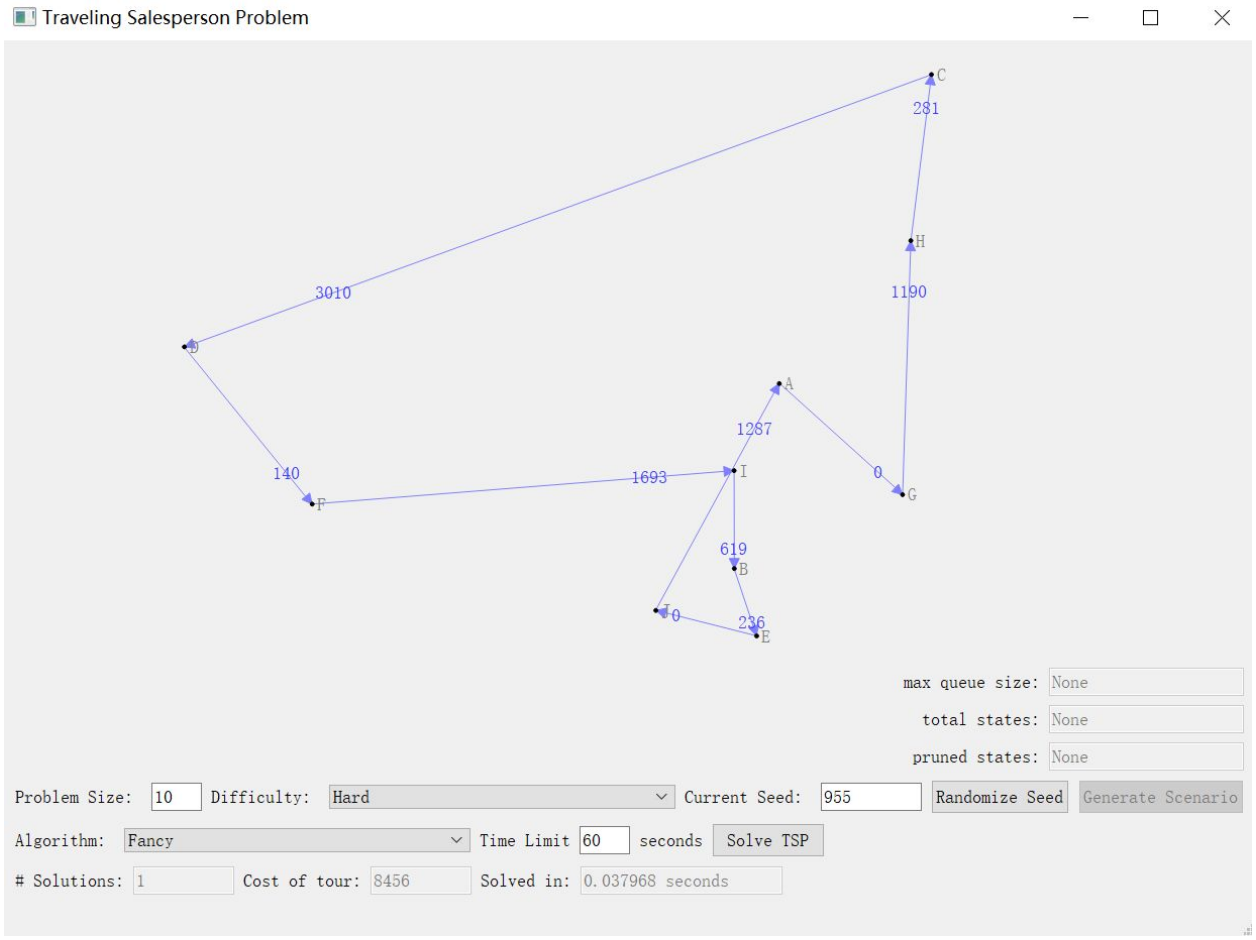
[7] The Hungarian Algorithm for the Assignment Problem. (n.d.). Retrieved April 30, 2020, from http://www.mathcs.emory.edu/~cheung/Courses/323/Syllabus/Assignment/algorithm.html

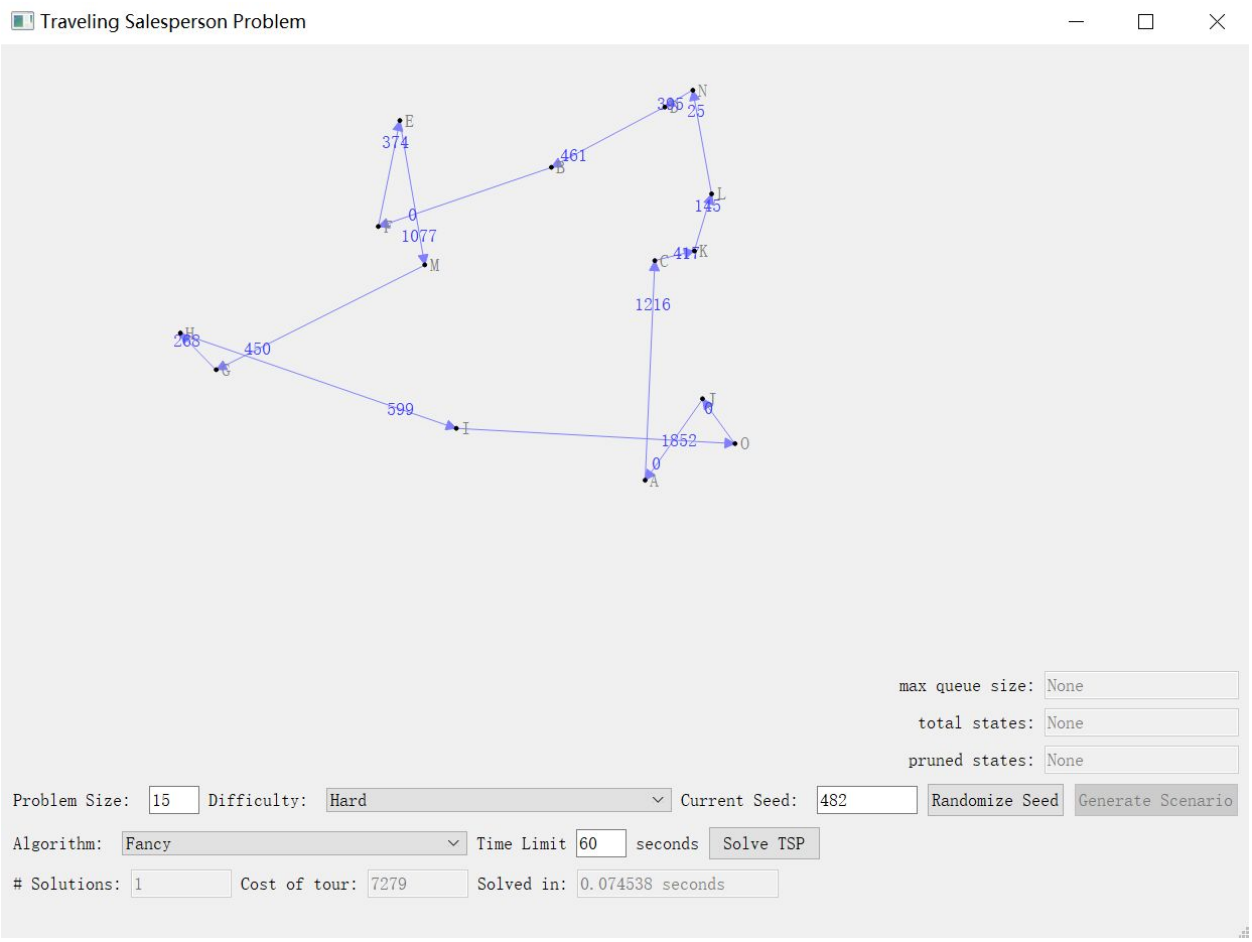
GitHub Link

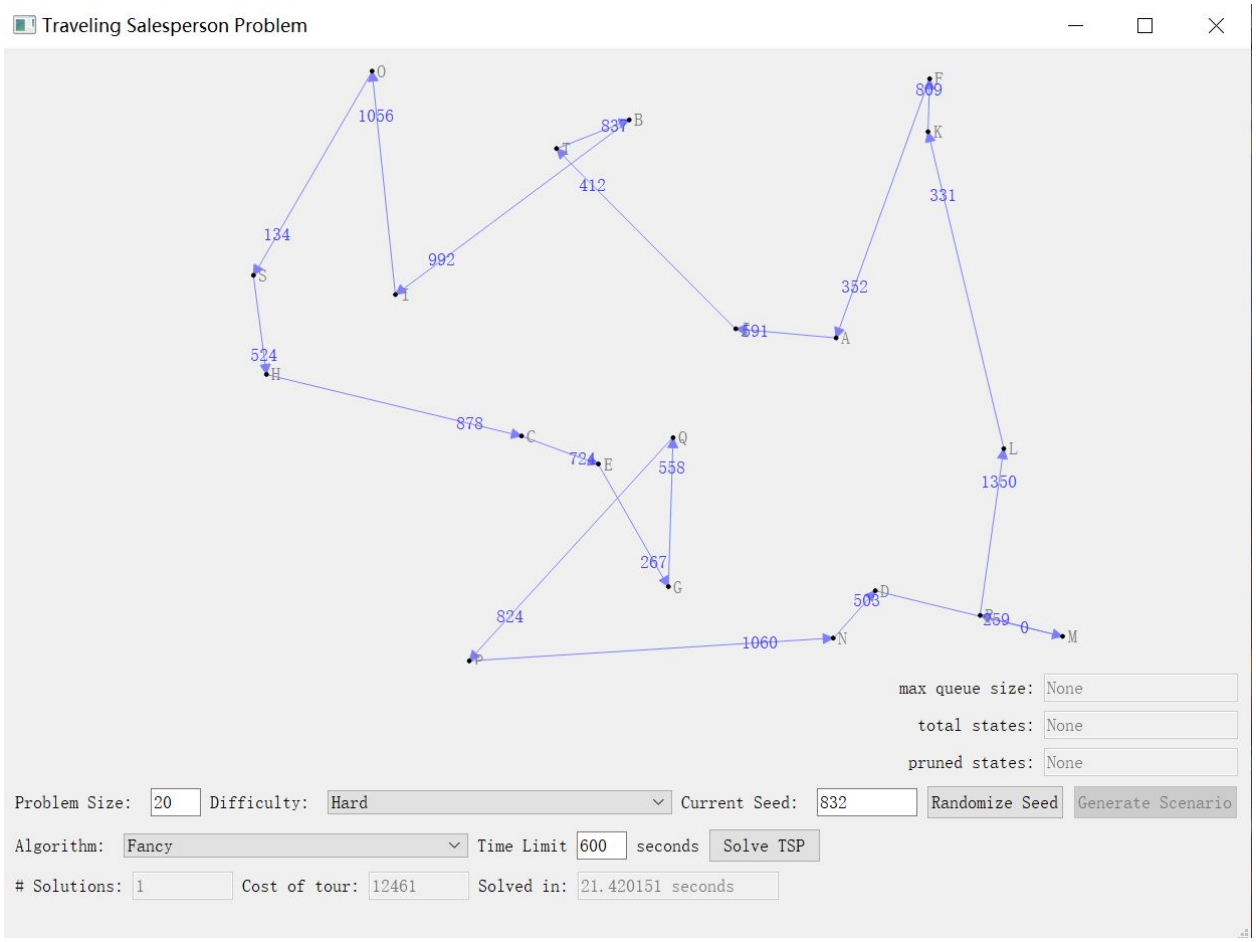
https://github.com/tylebrad/CS-4412-Project-6

Screenshots

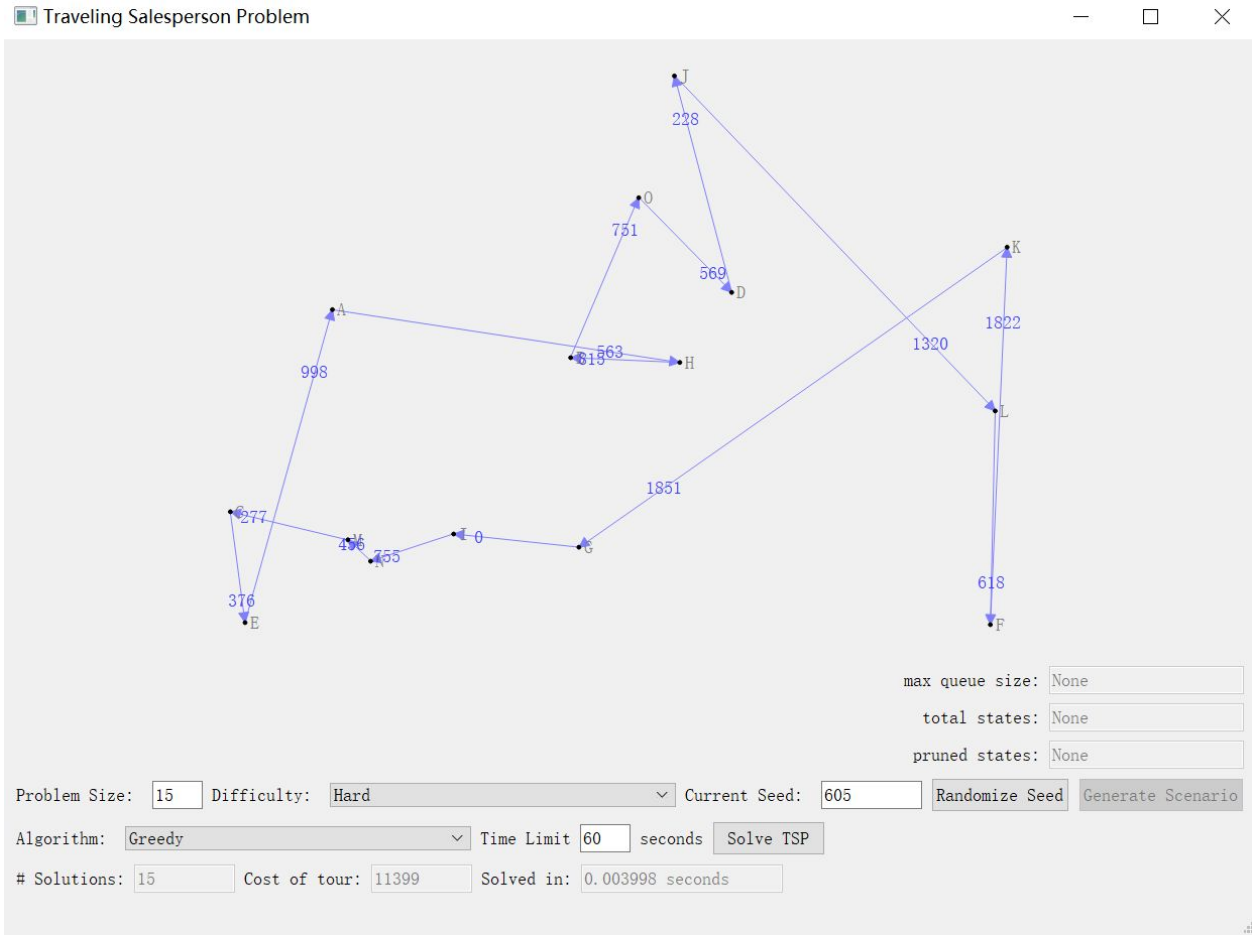
Hungarian algorithm

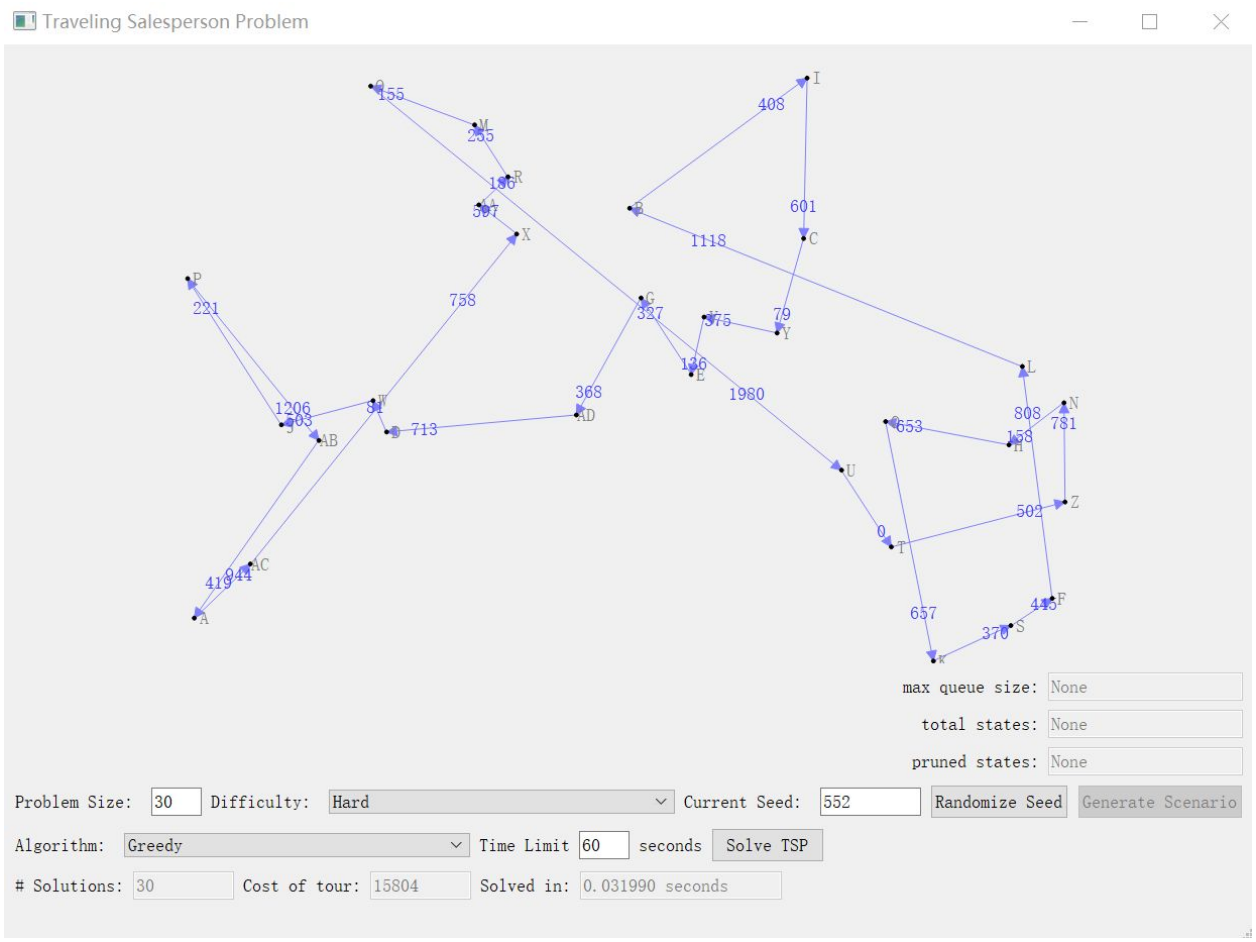


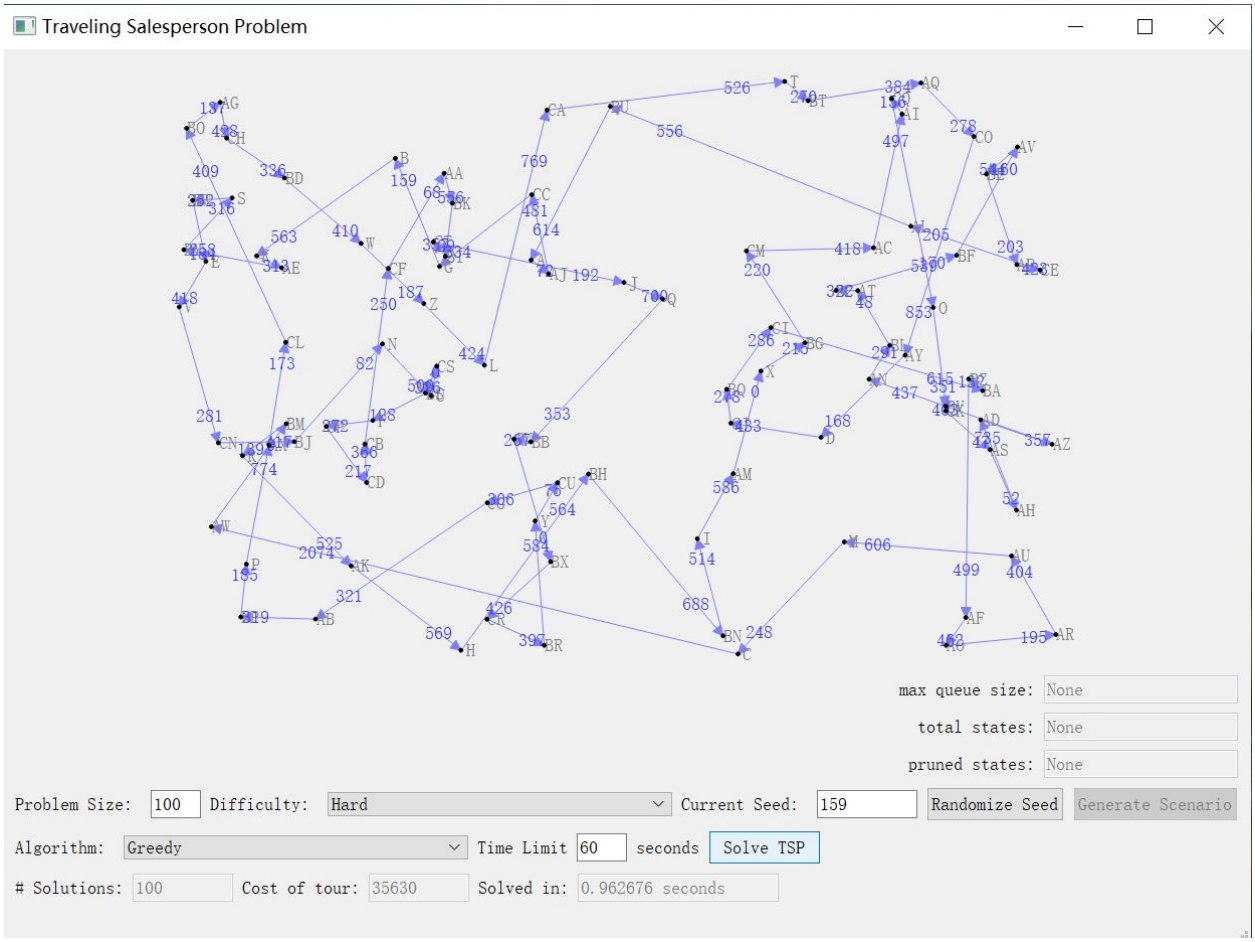

Traveling Salesperson Problem
N
E
374
461
B
L
145
0
F
1077
M
C
K
1216
H
285
450
S
599
I
J
0
1852
0
A
max queue size: None
total states: None
pruned states: None
Problem Size: 15
Difficulty: Hard
Current Seed: 482
Randomize Seed
Generate Scenario
Algorithm: Fancy
Time Limit 60 seconds
Solve TSP
Solutions: 1
Cost of tour: 7279
Solved in: 0.074538 seconds


Traveling Salesperson Problem
O
1056
B
837
T
412
F
809
K
331
134
992
S
I
352
591
A
524
H
878
C
724
E
Q
558
267
G
L
1350
824
D
503
259
0
M
P
1060
N
max queue size: None
total states: None
pruned states: None
Problem Size: 20
Difficulty: Hard
Current Seed: 832
Randomize Seed
Generate Scenario
Algorithm: Fancy
Time Limit 600 seconds
Solve TSP
Solutions: 1
Cost of tour: 12461
Solved in: 21.420151 seconds

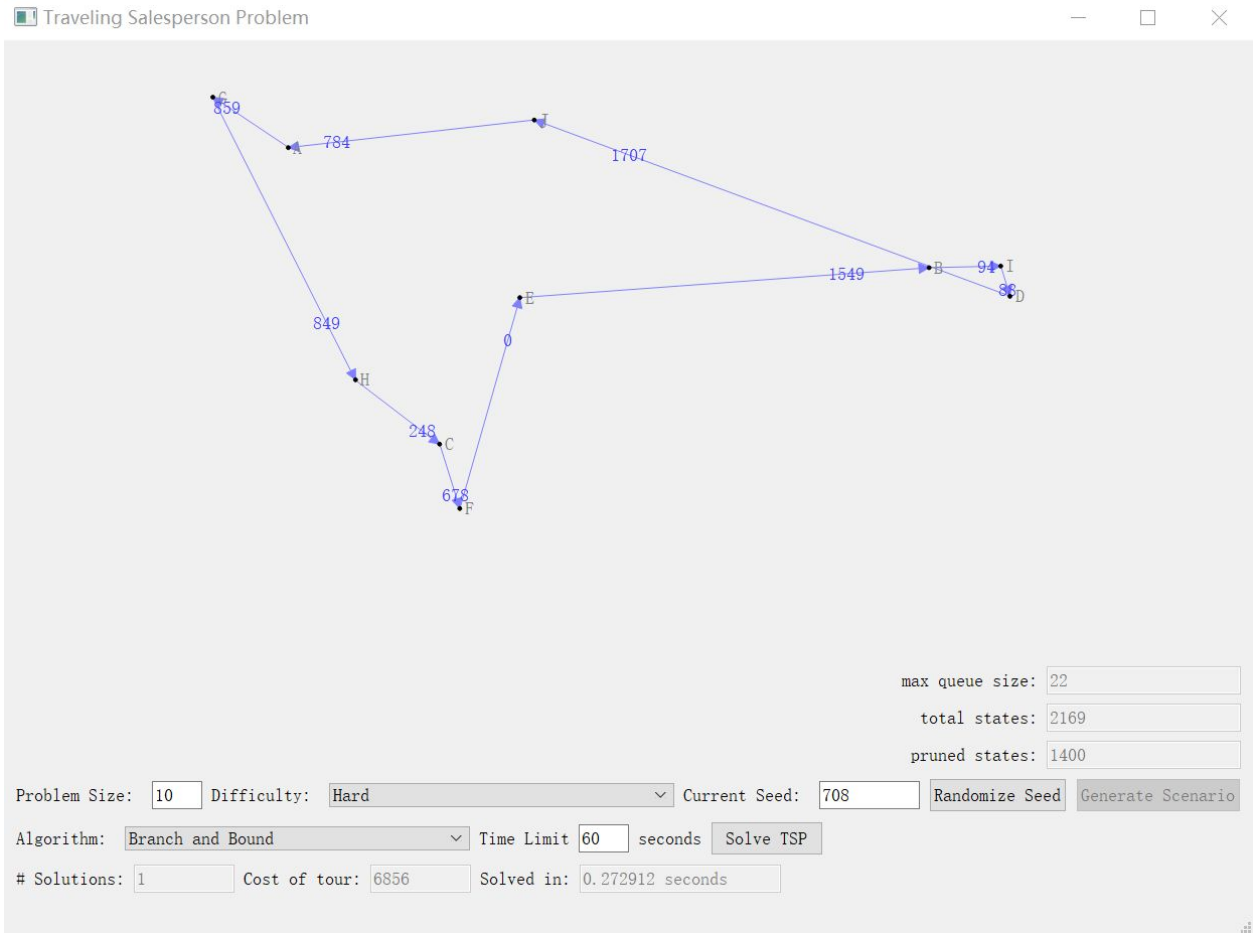
Greedy algorithm



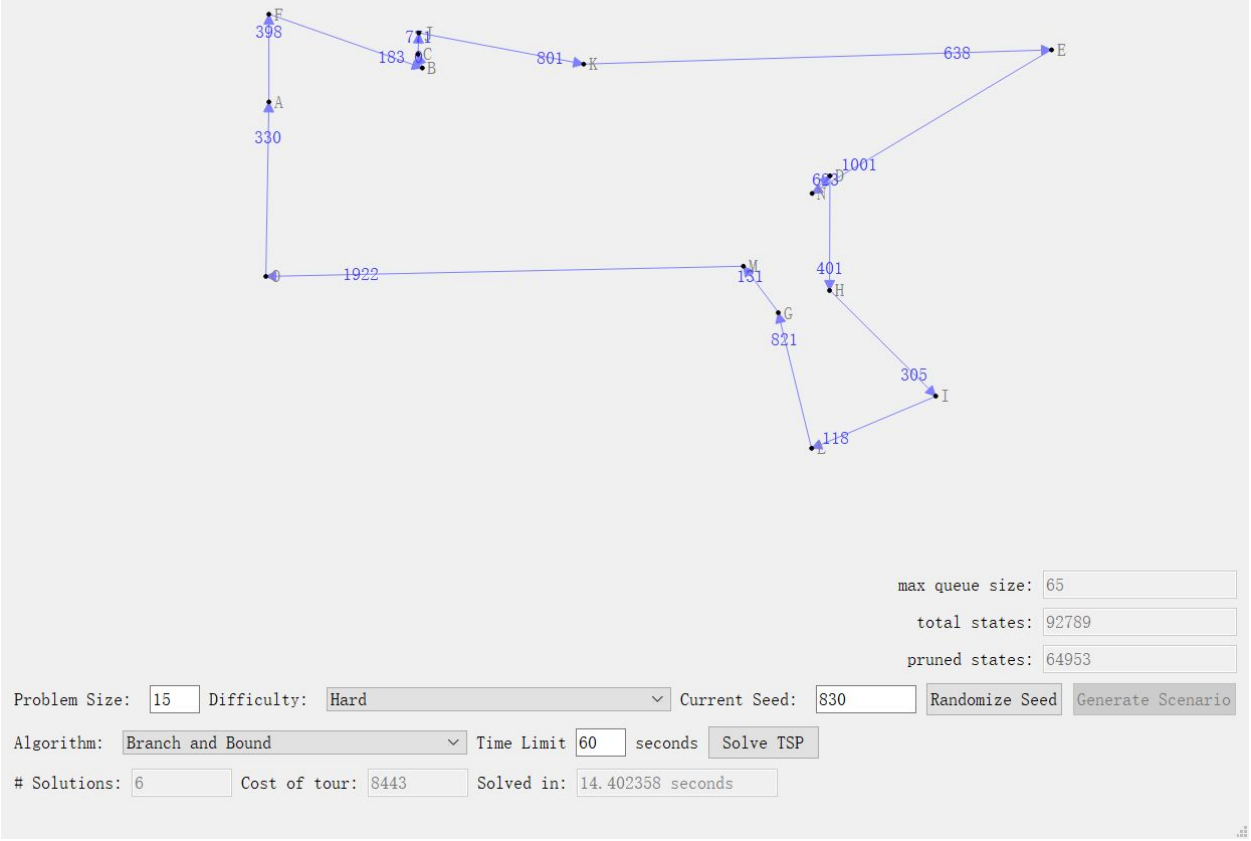

Traveling Salesperson Problem
155
235
408
601
1118
758
327
375
79
136
1980
368
1206
713
653
808
781
502
657
370
445
944
max queue size: None
total states: None
pruned states: None
Problem Size: 30
Difficulty: Hard
Current Seed: 552
Randomize Seed
Generate Scenario
Algorithm: Greedy
Time Limit 60 seconds
Solve TSP
Solutions: 30
Cost of tour: 15804
Solved in: 0.031990 seconds


Traveling Salesperson Problem
max queue size: None
total states: None
pruned states: None
Problem Size: 100
Difficulty: Hard
Current Seed: 159
Randomize Seed
Generate Scenario
Algorithm: Greedy
Time Limit 60 seconds
Solve TSP
Solutions: 100
Cost of tour: 35630
Solved in: 0.962676 seconds

Branch and bound algorithm



Traveling Salesperson Problem



max queue size: 65

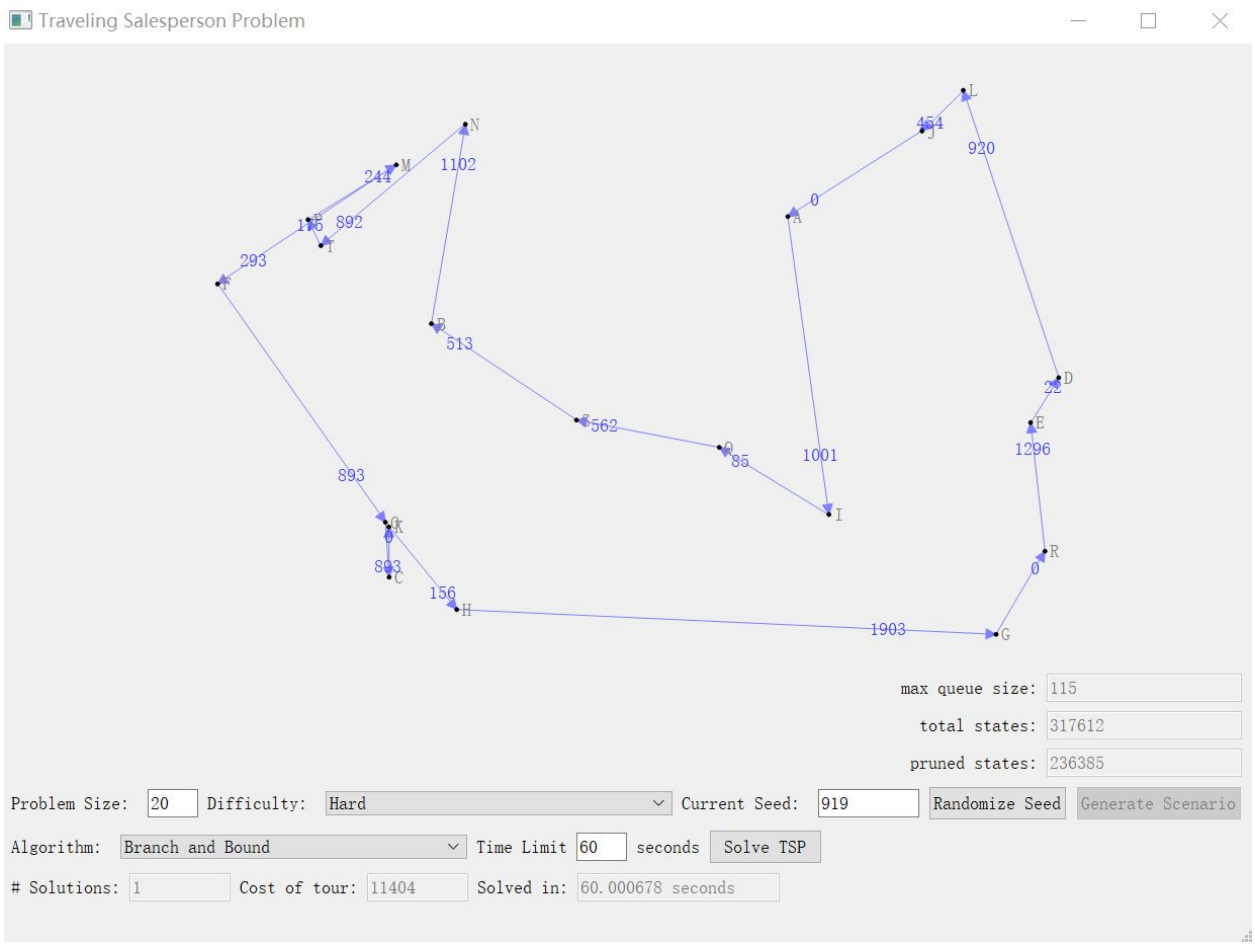
total states: 92789

pruned states: 64953

Problem Size: 15 Difficulty: Hard Current Seed: 830 Randomize Seed Generate Scenario

Algorithm: Branch and Bound Time Limit 60 seconds Solve TSP

Solutions: 6 Cost of tour: 8443 Solved in: 14.402358 seconds


Traveling Salesperson Problem
N
M
1102
244
892
293
513
562
85
1001
893
156
1903
0
454
920
22
1296
max queue size: 115
total states: 317612
pruned states: 236385
Problem Size: 20
Difficulty: Hard
Current Seed: 919
Randomize Seed
Generate Scenario
Algorithm: Branch and Bound
Time Limit 60 seconds
Solve TSP
Solutions: 1
Cost of tour: 11404
Solved in: 60.000678 seconds