Traveling Salesman Problem: Variation on Greedy and 2-opt Algorithms

Ellery Davila
Kevin DeVocht
Gavin Eccles
Joseph Hovik

1 Explanation of The TSP problem.
Computer Scientists have been fascinated with the Travelling Salesman Problem for decades. Since it's NP-Complete, the TSP problem has an intrinsic property of being easily verified but very hard to compute efficiently. With these two properties, computer scientists imaginations can run wild with different ways to implement solutions for the TSP, trying to get the fastest, most efficient method. Here is our shot at solving the TSP.

2 Explanation of our greedy algorithm

a. What is it?

The greedy approach we implemented to solve the TSP is similar to other implementations. The greedy algorithm starts at an arbitrary city and progresses to the next city using the cheapest edge as the deciding factor on which city to visit. This continues until a solution is found. By its very nature, the greedy algorithm takes what's best at the moment, this produces a solution very quickly with only basic optimization by choosing the shortest node at each point. Because of this, Greedy serves as a good benchmark for other algorithms. Which is why we will be comparing it with both the Branch and Bound algorithm and our own 2-opt algorithm.

b. Complexity: The Greedy algorithm has an overall Big-O time complexity of O(n^2) . The reason for this is because it must check every cell in the cost matrix to find the shortest path. The width of the cost matrix is n. The length of the cost matrix is n. This means that the cost of checking every cell in the cost matrix is $n \times n$ or n^2, causing the O(n^2)
time complexity.

3 Explanation of our own algorithm

a. For our implementation, we used the 2-opt algorithm. We decided to use it because it is a simple example of local search algorithms, and we wanted to explore local search algorithms more in-depth. Our 2-opt algorithm works as follows. Run our greedy algorithm for the number of cities in the Cities graph up to 20 times. Pass each solution into the 2-opt method. In the two opt method, remove 2 edges and then try and add two edges back into the solution and see if the new solution has a lower cost. Now loop through every combination of removing two edges. 2-opt then returns the best of the 20 solutions. One of the problems we ran into was the fact that we were using a directed graph. Every explanation of what 2-opt was, started on the assumption that the graph was undirected, including Johnson and McGeoch[1995]. In fact, we never found a single explanation that even explicitly said one way or another if the graph needed to be directed or not. But as we read through the explanation of how the algorithm works, it became quite clear that all of the authors were talking about an undirected graph. As we started working through how we would implement 2-opt, it became apparent very quickly that a directed graph would require some additional checking. We found that it could be possible for the newly optimized solution using 2-opt, to actually not even be a solution at all.

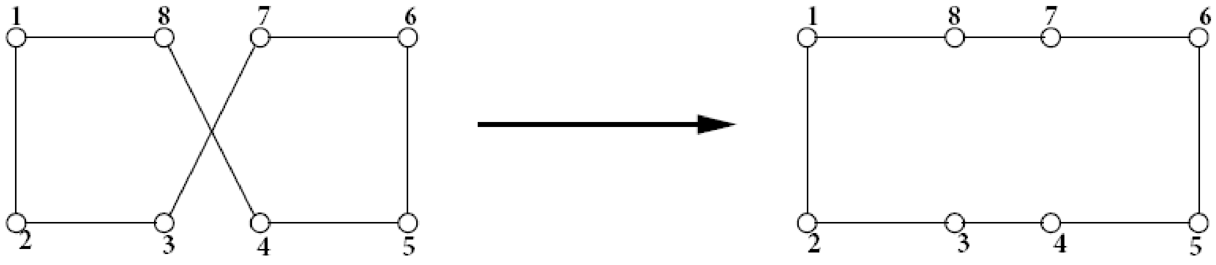
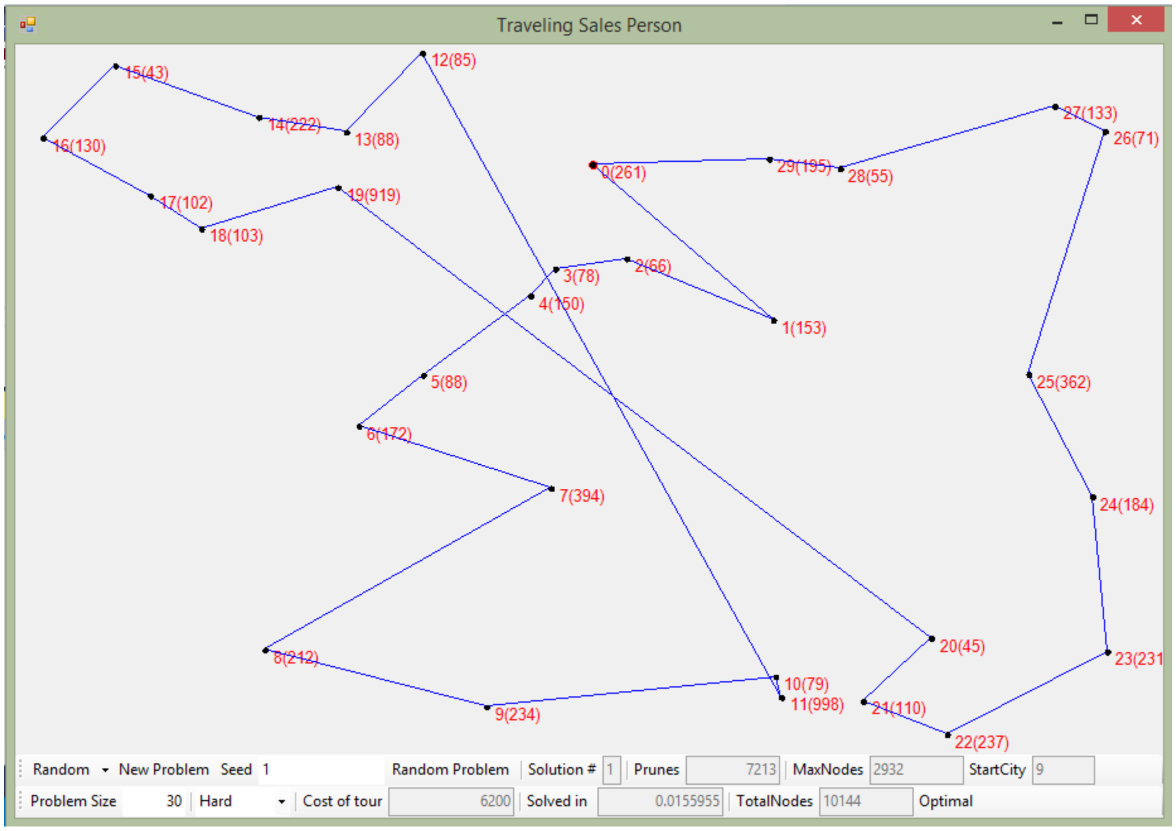


Fig.1

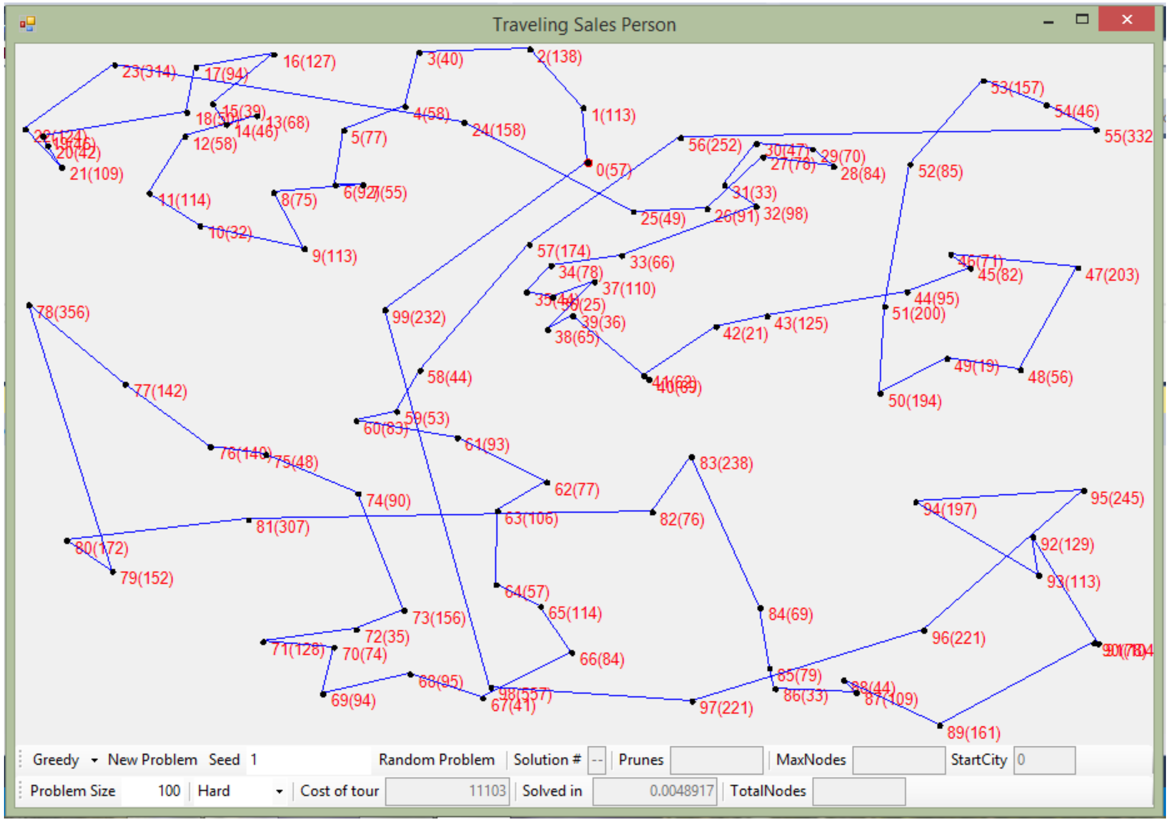
Lets quickly walk through the problem with using a directed graph. In figure 1, you can see the example from Dasgupta, Papadimitriou and Vazirani[2006] on page 298. We have added node numbers to help explain the problem. Lets say that the graph on the left is a solution that was passed to our 2-opt method. According to 2-opt, the solution on the right would be the newly optimized solution. A possible list representation of the initial solution could be 1,2,3,7,6,5,4,8. the new solution after running through 2-opt would then be represented as 1,2,3,4,5,6,7,8. And herein lies the problem. Using a directed graph and especially using the hard mode from the project which removes edges, it is possible that there is no path from 3 to 4 or 6 to 7 for example. Which means that we had to add some checking to the 2-opt algorithm to make sure that the solution was even valid. Looking at all of the algorithms that we have implemented, 2-opt seems to be the happy medium. While it is not as fast as the greedy algorithm, it does produce a solution with a reduced cost. On the other hand, Branch and Bound generally produces a solution with a lower cost, but it takes significantly longer than either greedy or 2-opt. So 2-opt is not the fastest or the best solution but is better in one of the two categories, regardless of which other algorithm we compared it to.

b. Complexity: Looking at our implementation of 2-opt, you see that the overall Big-O time complexity stays as O(n^2) . The real question though, is whether or not 2-opt can produce a solution with a lower cost than the greedy algorithm in a reasonable amount of time. Looking at Fig. 2 you see that it in fact does reduce the cost from the greedy algorithm and in a reasonable amount of time. This means that it must be running in O(n^2) , if our algorithm was worse, than the time it took would have been growing faster than it was.
c. Typical Images of Each Algorithm's Solution.



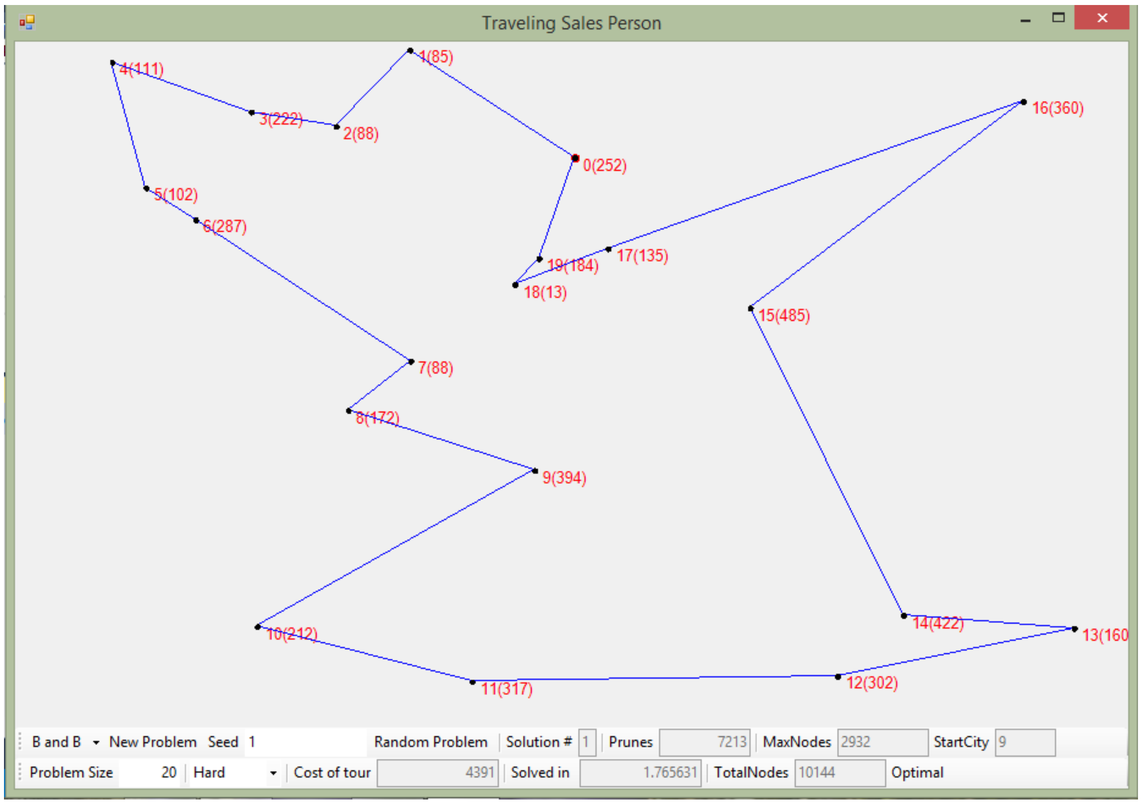
Random New Problem Seed 1 Random Problem Solution # 1 Prunes 7213 MaxNodes 2932 StartCity 9

Problem Size 30 Hard Cost of tour 6200 Solved in 0.0155955 TotalNodes 10144 Optimal



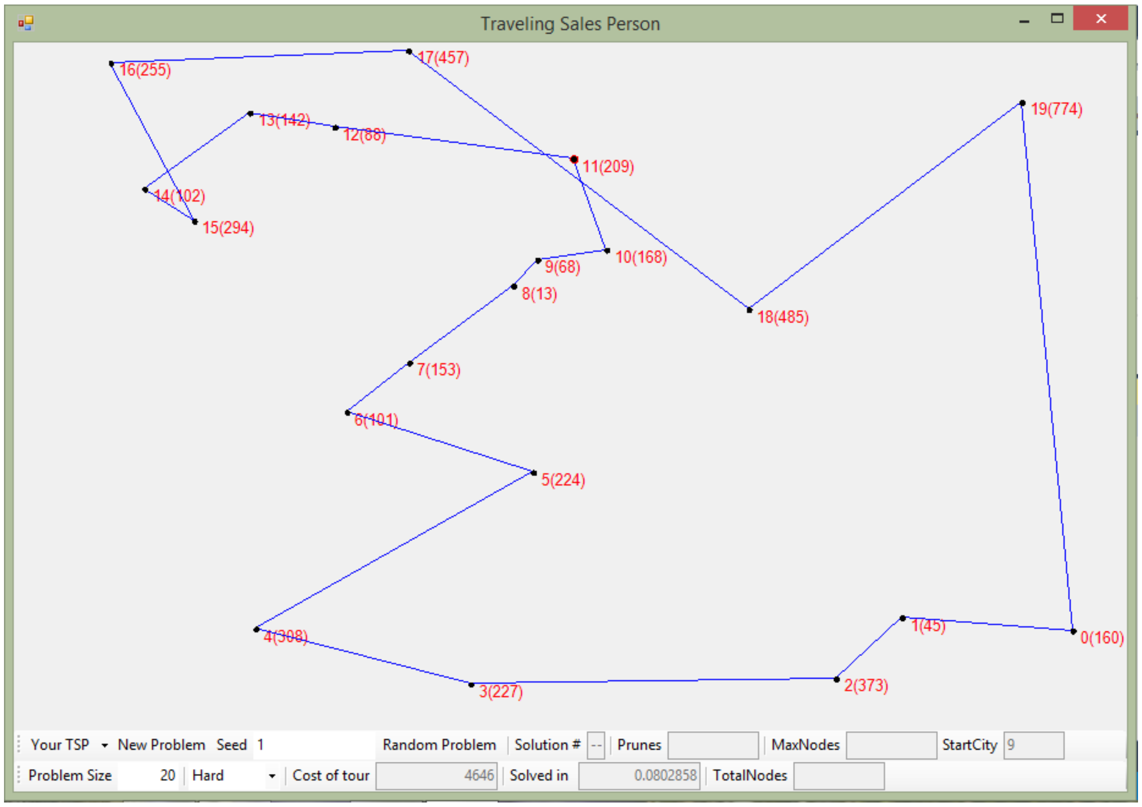
Greedy New Problem Seed 1 Random Problem Solution # -- Prunes MaxNodes StartCity 0

Problem Size 100 Hard Cost of tour 11103 Solved in 0.0048917 TotalNodes



B and B ▾ New Problem Seed 1 Random Problem Solution # 1 Prunes 7213 MaxNodes 2932 StartCity 9

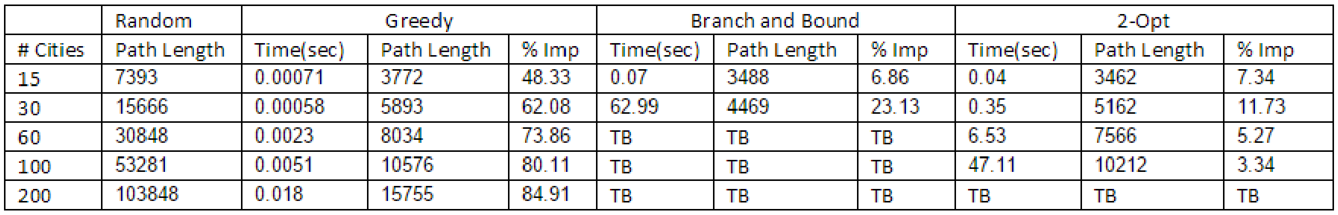
Problem Size 20 Hard ▾ Cost of tour 4391 Solved in 1.765631 TotalNodes 10144 Optimal



Your TSP ▾ New Problem Seed 1 Random Problem Solution # -- Prunes MaxNodes StartCity 9

Problem Size 20 Hard ▾ Cost of tour 4646 Solved in 0.0802858 TotalNodes

4. Analysis of Algorithms:



	Random	Greedy			Branch and Bound			2-Opt		
# Cities	Path Length	Time(sec)	Path Length	% Imp	Time(sec)	Path Length	% Imp	Time(sec)	Path Length	% Imp
15	7393	0.00071	3772	48.33	0.07	3488	6.86	0.04	3462	7.34
30	15666	0.00058	5893	62.08	62.99	4469	23.13	0.35	5162	11.73
60	30848	0.0023	8034	73.86	TB	TB	TB	6.53	7566	5.27
100	53281	0.0051	10576	80.11	TB	TB	TB	47.11	10212	3.34
200	103848	0.018	15755	84.91	TB	TB	TB	TB	TB	TB

Fig. 2

As you can see in Fig. 2 our 2-opt solution does not reduce the cost as much as B & B but if you look at the times, it is significantly faster for only slight gains in cost. This allows 2-opt to run in a reasonable time on larger lists of cities. B&B hits a wall around 30 cities while 2-opt can almost get 200 cities in 10 minutes(we 2-opt run for as long as it needed on 200 cities and it only took 13 minutes). So in conclusion we found that for the tighter bound that branch and bound offers, the time it takes isn't feasible since it will take 10 minutes or more just to do 40 cities. On the other hand, greedy alone is fast but not terribly tight as far as bounds. Our algorithm meets a good middle ground, where our cost is not as tight as B & B, and our time isn't as fast as greedy, but performs well, supplying a good cost at a respectable time.

If we had more time we would have liked to study k-opt to see if there is an optimal k. We would have also liked to do more research into the effects of the seed solution on k-opt. We also came up with another algorithm that we would have liked to have tested out further. It is a depth first branch and bound. We weighted the states lower down in the tree to help us reach a solution faster. We created it towards the end and did not have enough time to fully research it but the preliminary results look very promising. Individual cases are sometimes worse than greedy, but the average cost is an improvement. for example with 30 cities, this algorithm has a 15% improvement over greedy. Another promising aspect of this solution was that it was able to handle more cities than our 2-opt algorithm could. This allowed us to improve on greedy at a larger number of cities than we had previously been able to do.

Johnson and McGeoch[1995] David S. Johnson
Lyle A. McGeoch, ''The Traveling Salesman Problem:
A Case Study in Local Optimization", (1995), 3. 2-OPT, 3-OPT, AND THEIR VARIANTS

Dasgupta, Papadimitriou and Vazirani[2006] S.Dasgupta,C.H.Papadimitriou, and
U.V.Vazirani, "Algorithms", (2006) 297-298