The Traveling Salesman Problem:
Adapting 2-Opt And 3-Opt Local Optimization to Branch & Bound Techniques

Hitokazu Matsushita
hitokazu@byu.edu

Ogden Mills
ogdenlaynemills@gmail.com

Nathan Lambson
nlambson@gmail.com

December 7, 2011
Professor Tony Martinez
BYU, Computer Science

Abstract

The Travelling Salesmen Problem has captured the attention and resources of both the academic and business world. In an effort to discover new, and effective strategies to solve TSP, our team adapted well known TSP strategies with optimization techniques to create a unique algorithm capable of solving this complex problem. The algorithm is based on the 2-Opt and 3-Opt local search optimization algorithms and used in conjunction with a modified branch and bound algorithm. The result is a unique algorithm which is capable of solving an ATSP (asymmetrical travelling salesman problem) of 300 cities in approximately 12 minutes. The algorithm gradually improved the solutions path length as compared to the greedy solution until it capped at approximately 26% for TSPs with 100+ cities.

1 Introduction

This report centers on the algorithm created by the authors which solves ATSP problems. Different aspects of the algorithm will be analyzed including core algorithm paradigms, creative adaptations, optimization techniques, the process which led to the algorithms structure, and the effectiveness of the algorithm as compared to a greedy, random, and Branch & Bound approach. Additional information will be given on the Branch & Bound algorithm along with applicable optimizations and how these affect the algorithm created by the authors – a variation of the 2-Opt and 3-Opt local search algorithms. In order to establish a basis for comparison, a discussion of the Greedy approach is necessary.

2 The Greedy Approach

2.1 Implementation of the Algorithm

The Greedy implementation starts from an arbitrary city and searches for the shortest available path to an unvisited city. After occupying this city the algorithm repeats the process until it has arrived at the original city.

Source code for this implementation is to be found in the files attached to this report – All algorithms discussed begin in the "ProblemAndSolver.cs" file.

2.2 Purpose as a Benchmark

This approach applies the simplest form of optimization to the naïve solution for TSP problems – Greediness. For this reason the Greedy approach will serve as a benchmark against other algorithms.

2.2 Big-O Complexity Analysis

The implementation of the Greedy algorithm uses the following non-trivial functions during its execution:

- City.costToGetTo - It is comprised of O(1) constant time operations and reduces to O(1)
- TSPSolution.costOfRoute - Traverses each edge in a given route, the size of a route is O(n) times some constant factor which reduces to O(n).
- ProblemAndSolver.PickCheapestPath - Executes constant time operations for every city in the current citySet, O(n).
- ProblemAndSolver.FindGreedyRoute - For every city it also calls PickCheapestPath, O(n^2)
- ProblemAndSolver.solveProblemGreedily - Essentially a wrapper around FindGreedyRoute which has O(n^2) runtime complexity.

The Big-O complexity of the Greedy Algorithm is O(n^2).

3 The Authors TSP Approach

3.1 Implementation of the Algorithm

Our TSP solution is a variation of the 2-opt and 3-opt local search algorithm using the best BSSF (best solution so far) from a 30 run of a Branch and Bound solution – with upper and lower bounds – as the initial BSSF for the algorithm.

Source code for this implementation is to be found in the files attached to this report – All algorithms discussed begin in the "ProblemAndSolver.cs" file.

3.2 Local Search Algorithms

Local search algorithms are designed to be more efficient by implementing a neighbor-style solution. A local search algorithm can be defined in terms of the operations, exchanges or moves, which can be applied to modify a given tour to become like another tour. For example, if you began with a feasible tour, the local search algorithm would perform a series of exchanges or moves as long as it detected a gradual improvement over each iteration. Once the algorithm terminates it has discovered a locally optimal tour.

Local search algorithms are based on the basic operation of a "move." a move deletes two edges thereby dividing a tour into two paths. Later, the algorithm will reconnect these paths in the opposite configuration. This move was first proposed by Flood[1956] and later adapted to the 2-opt local search algorithm by Croes[1958]. The main idea behind it is to take a route that crosses over itself and reorder it so that it does not.

3.3 2-Opt and 3-Opt Local Optimization

2-opt, 3-opt, and k-opt search algorithms are among the simplest and best known local search algorithms which still yield significant benefits from the local search paradigm, though there are certain weaknesses which should be considered. Papadimitriou and Steiglitz, [Papadimitriou & Vazirani, 1984], showed that for 2-opt, 3-opt, and k-opt algorithms, there are certain TSP's which have only one optimal tour but exponentially many locally optimal tours, all of which are longer than the optimal tour by an exponential factor. Worst-case results for these types of algorithms can be obtained if the TSP to be solved is not generated randomly but contrived to exploit the weaknesses in 2-opt, 3-opt, and k-opt search algorithms.

3.4 Branch & Bound

An important aspect of the algorithm is its need for a good initial path. Even though the original use of the greedy algorithm to generate the initial path was functional, it nevertheless hindered the algorithm on the whole and made analysis of the 2-Opt and 3-Opt implantation impractical. Something had to be done to further optimize the algorithm.

Early on efforts were made to adapt the greedy algorithm to create a tighter initial bound for the 2-Op and 3-Opt algorithm, but the results were nominal. Seeing the importance of the initial state used by the local search algorithm, the focus for optimization shifted to efficiently creating an even tighter initial bound for the 2-Opt and 3-Opot algorithm.

Efforts to optimize the initial K-Opt algorithm path led to the use of branch and bound. Branch and bound, being an optimal algorithm, is capable of solving the same TSP problems as a local search algorithm but consumes available resources very quickly. For this reason branch and bound was restricted to a maximum 30-second execution period, where the BSSF would then be used as the initial path for the local search algorithm.

Given such a small time constraint for branch and bound to run, the algorithm was modified to "dig" for a solution thereby greatly increasing the odds of finding an improved BSSF within the time limit. In practice, for smaller values of n, the branch and bound algorithm would often discover the optimal solution.

3.5 Branch & Bound Implementation

The branch and bound algorithm used to determine the starting path for the 2-Opt and 3-Opt algorithm uses both upper and lower bounds in conjunction with digging, or depth-first searches, in order to more reliably return a useful BSSF within its 30-second execution time limit.

Two queues were created, one for the include states and the other for the exclude states. This modified branch and bound would first empty the include queue and then proceed to do the same with the exclude queue, forcing it to dig as deep as possible. This helped greatly in consistently returning better solutions – even when the algorithm timed out. This improved initial value for the local search algorithm proved key in optimizing the algorithm as a whole.

3.4 Why These Algorithms Were Selected

One of the most reasonable and robust approaches to address the TSP problem is local search. The most basic implementation of local search for TSP is called 2-opt. In 2-opt, two arbitrary edges in the existing route are chosen and examine whether the cost of those edges are reduced by exchanging those edges. The same process is iteratively conducted for any pair of edges and terminated when there is no more improvement. This is a simple but powerful method to quickly search through and identify a better route than the current one in a simple nested loop. However, this approach is prone to be caught by a local optimum because the next option to take in the exchange process relies solely on the particular two edges, which is similar to greedy search in terms of the narrow horizon. This indicates that the possibility of reaching local optima is very high when the total number of cities are quite large due to many local optima in such a case. Furthermore, 2-opt can be quite ineffective in our problem setting because the paths between two cities are asymmetric and there are chances that the costs of edges between cities can be infinity.

To mitigate the chances of being caught by some local optima with 2-opt, we further investigated incorporating 3-opt local search, in which three consecutive cities are inserted in the other positions of the route and are then analyzed for any potential cost reduction. This method is simply a variation of 2-opt; therefore it is not guaranteed to avoid local optima in the process of searching. However, the chances to find the global optimum is increased by using 2-opt and 3-opt concurrently because the patterns of route change proposed by these two approaches are fundamentally different. To combine these two local search methods, we implemented the algorithms in a sequential order and iteratively examined all the possible paths proposed by them.

Another problem caused by local search is minimizing the cost of internal edges enclosed between the exchanged edges. In the regular 2-opt or 3-opt, this can be dealt with simply by reversing the order of such internal nodes to maintain the adjacent relations between cities in the original route. However, this is not true in our case because of the asymmetric nature of the edges in this problem. To ensure that we always obtained a better solution than the current solution, after edge exchanges we restricted the path exchange to when both edge exchange and reordering of the internal cities in the route resulted in a better route. However, we assumed that this condition could be too tight thereby reducing the possibilities of obtaining better solutions. To increase the chances to find better solutions, we implemented greedy search locally (i.e., cities between the exchanged edges). Although this approach sacrificed time efficiency,

we were able to find more reasonable solutions which seemed to be optimal in many cases up to 150 cities.

3.5 Pros and Cons

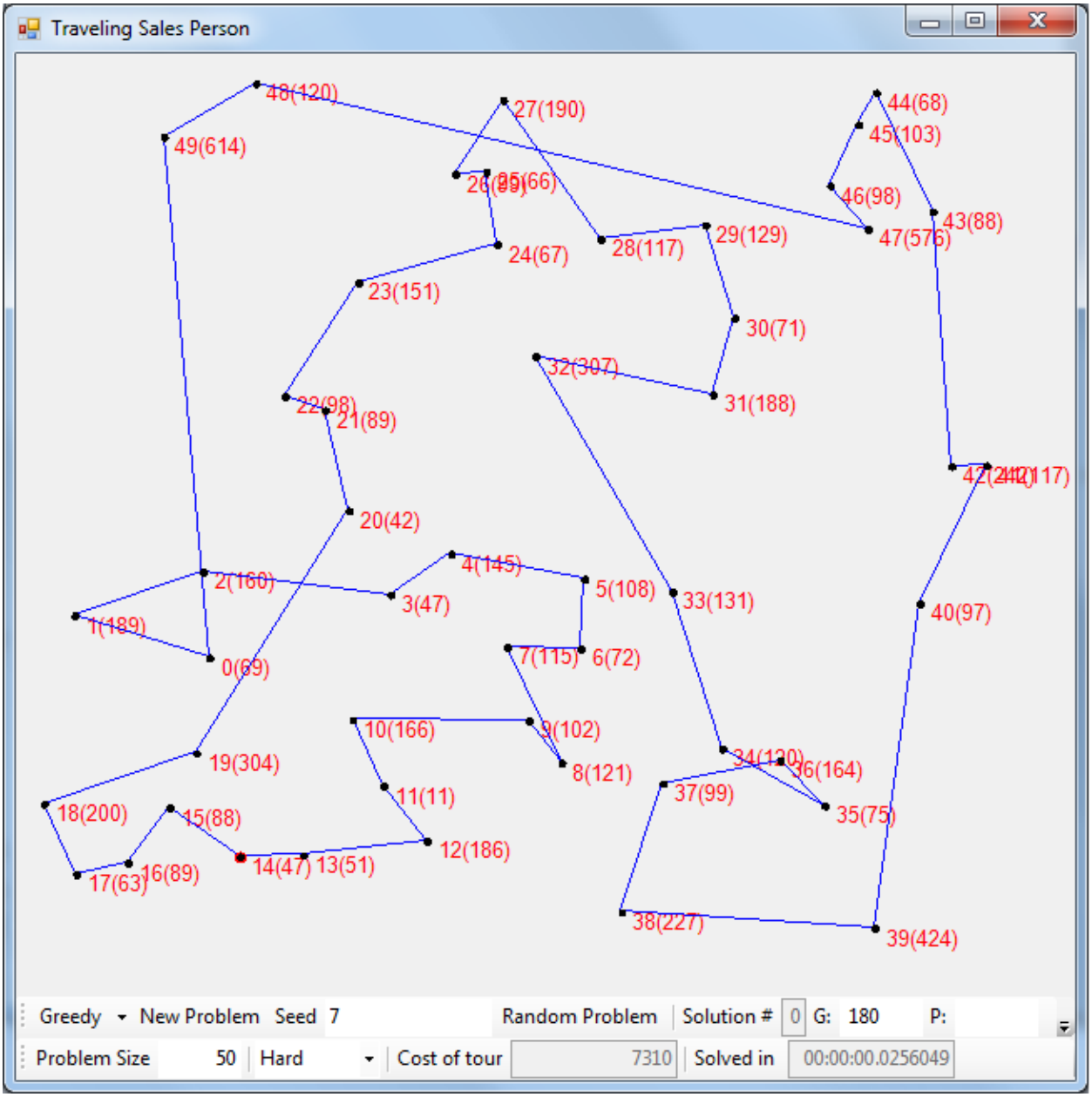
Pros

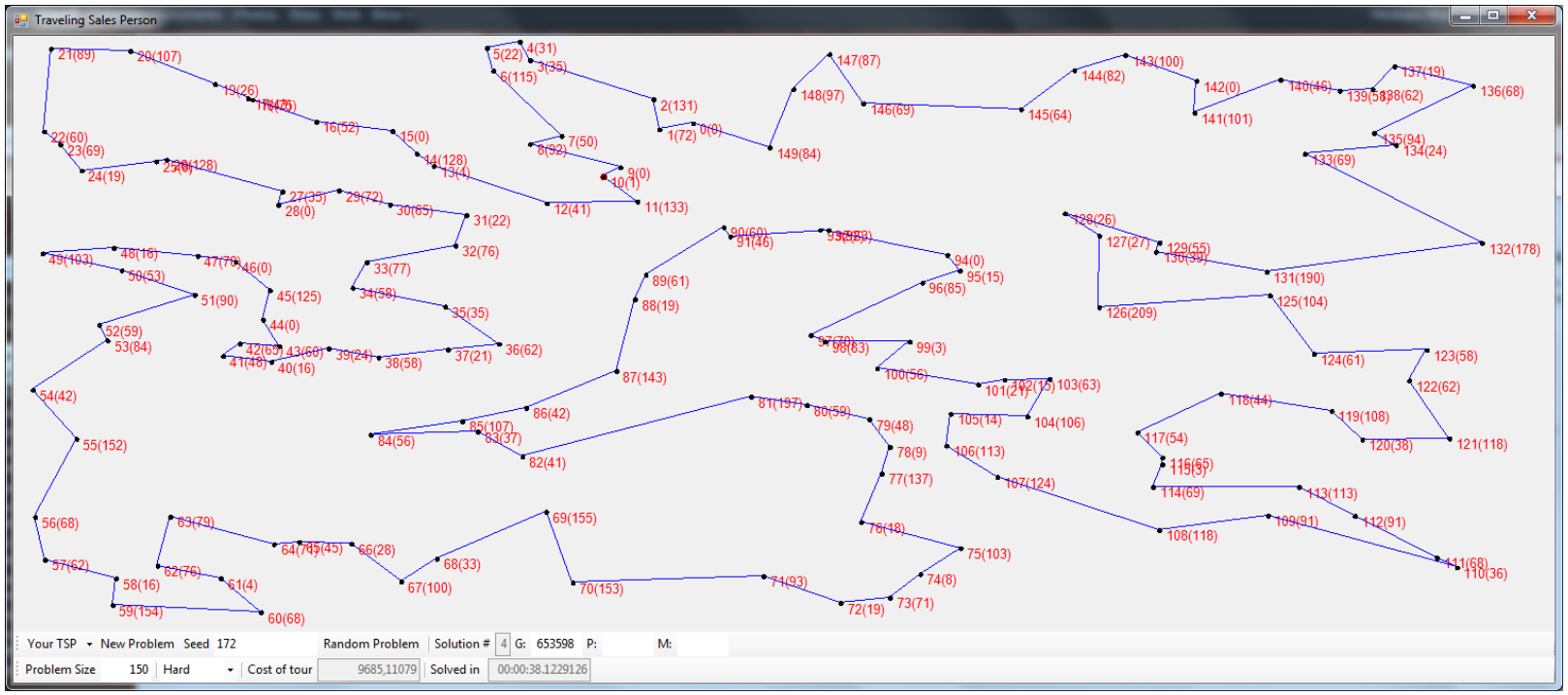
- Because our algorithm is a local search algorithm it is also an "anytime algorithm," meaning that the algorithm can be terminated at any time and still return a "best solution so far."
- Our TSP algorithm is relatively simple to implement as compared to many other more complicated TSP solution styles.
- Despite having certain configurations of TSP which will cause poor performance from local search algorithms, real-world worst-case scenarios are much better than the theoretical worst-case scenarios.

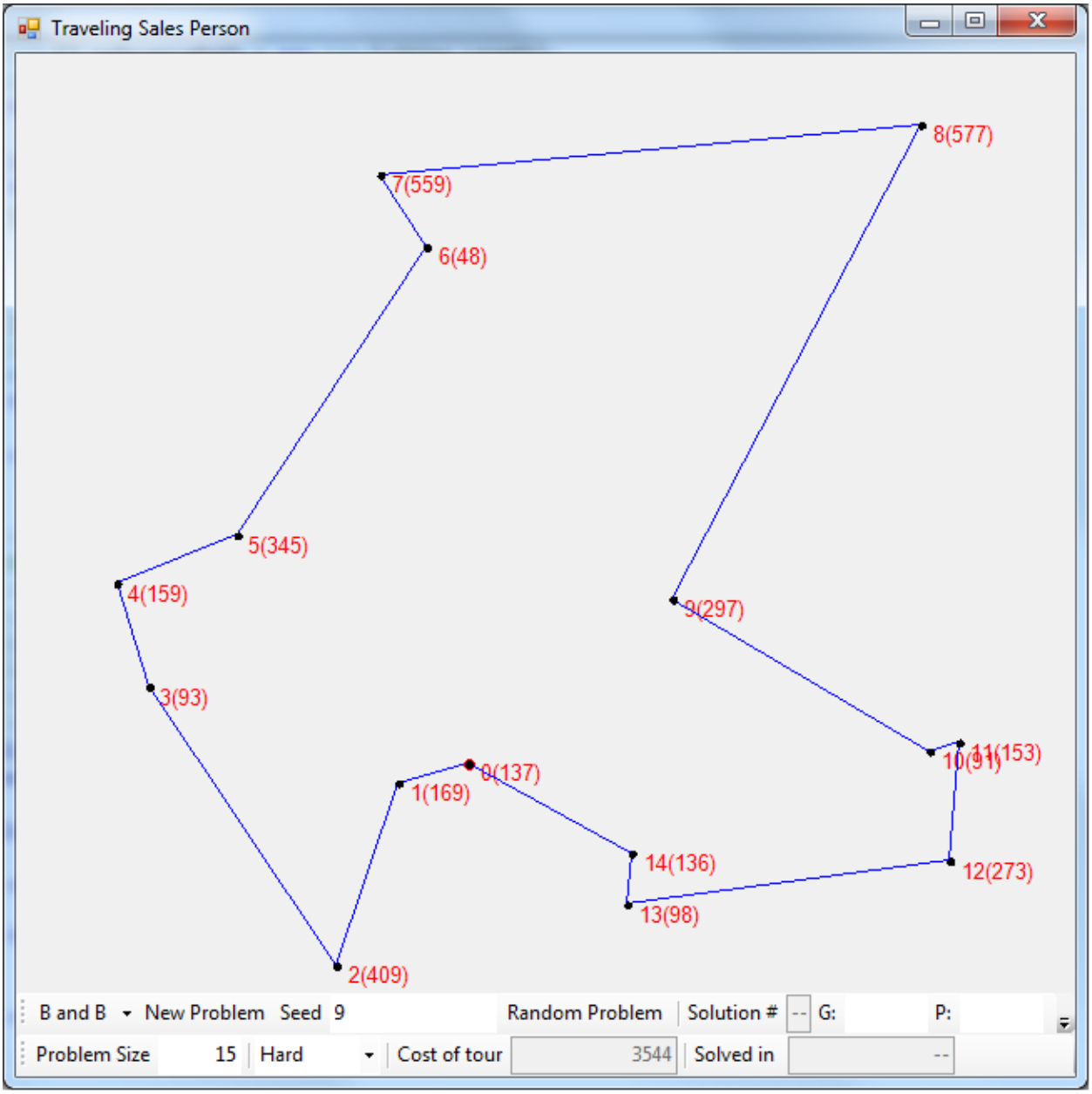
Cons

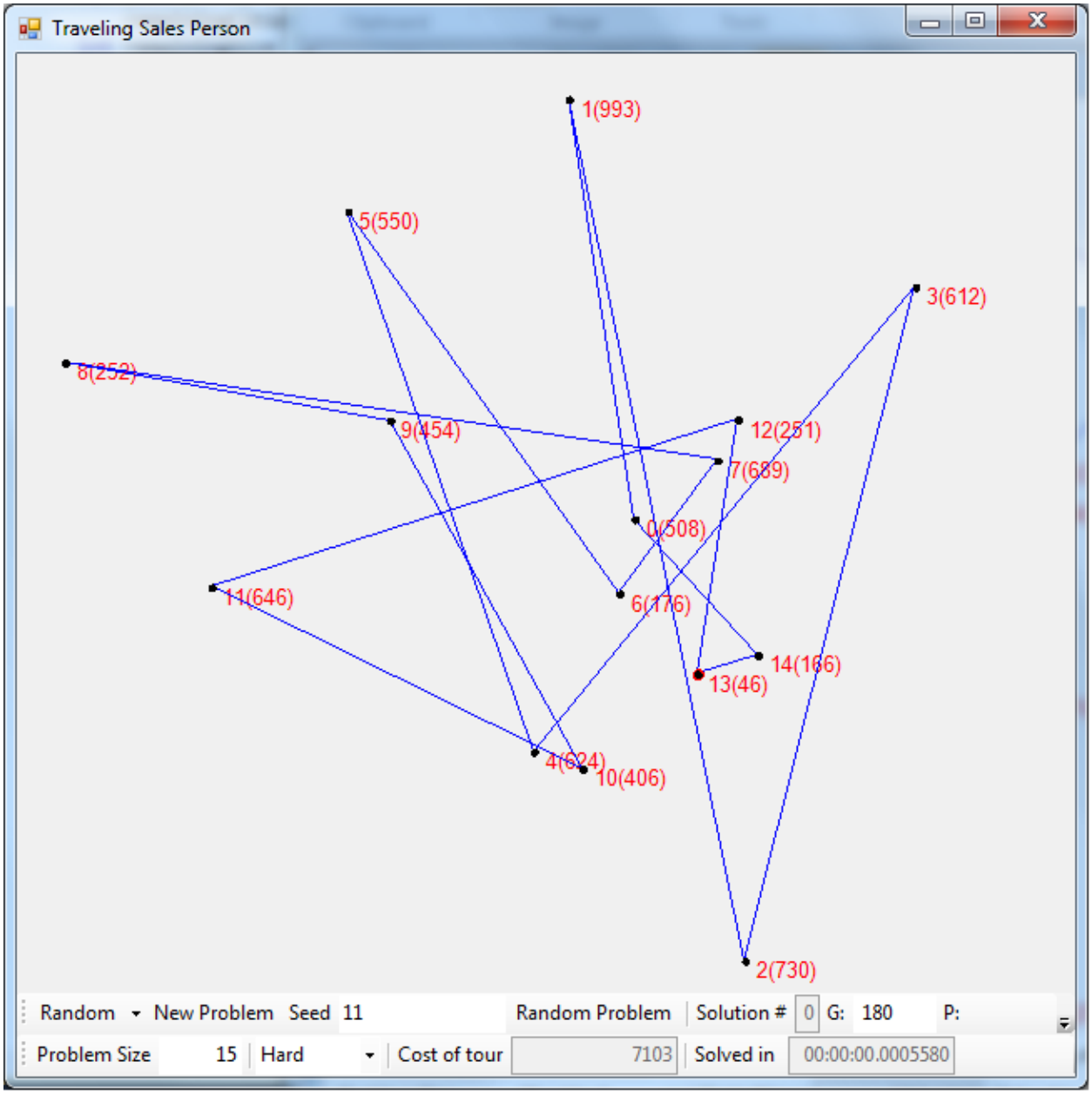
- There are known configurations of TSPs which will cause worst-case scenarios.
- There are other, more complicated, variations of the 2-opt and 3-opt local search algorithms which yield better results
- Local search algorithms have a tendency to search "excessively" as compared to other styles and rely on tight bounding functions to mitigate these effects.

4 Typical Screenshots of Each Algorithm




Traveling Sales Person
Your TSP
New Problem
Seed 172
Random Problem
Solution # 4
G: 653598
P:
M:
Problem Size 150
Hard
Cost of tour 9685,11079
Solved in 00:00:38.1229126


Traveling Sales Person
8(577)
7(559)
6(48)
5(345)
4(159)
9(297)
3(93)
11(153)
10(91)
0(137)
1(169)
14(136)
12(273)
13(98)
2(409)
B and B
New Problem
Seed 9
Random Problem
Solution #
--
G:
P:
Problem Size
15
Hard
Cost of tour
3544
Solved in
--


Traveling Sales Person
1(993)
5(550)
3(612)
8(252)
9(454)
12(251)
7(689)
0(508)
11(646)
6(176)
14(166)
13(46)
4(624)
10(406)
2(730)
Random
New Problem
Seed
11
Random Problem
Solution #
0
G:
180
P:
Problem Size
15
Hard
Cost of tour
7103
Solved in
00:00:00.0005580

5 Empirical Results Table of TSP Algorithms

Random

# of Cities	Speed	Path	Imp
15	0.000	7,776.22	0.000
30	0.000	16,145.875	0.000
60	0.000	31,198.000	0.000
120	0.001	64,581.250	0.000
240	0.002	123,683.333	0.000

Greedy

# of Cities	Speed	Path	Imp
15	0.001	4,173.33	0.463
30	0.001	5,466.333	0.661
60	0.004	8,126.778	0.740
120	0.012	11,862.667	0.816
240	0.064	17,685.000	0.857

B & B

# of Cities	Speed	Path	Imp
15	0.015	3,364.56	0.194
30	17.310	4,573.889	0.163
60	TB	TB	TB
120	TB	TB	TB
240	TB	TB	TB

Groups TSP

# of Cities	Speed	Path	Imp
15	0.021	3,364.56	0.194
30	17.572	4,568.444	0.164
60	77.801	6,230.556	0.233
120	55.380	8,671.333	0.269
240	515.413	12,925.111	0.269

5.1 Discussion and Analysis of Results

For smaller cities (below 50) our algorithm performs as well as branch and bound because branch and bound usually discovers the correct solution making 2-Opt and 3-Opt unnecessary. As the city size increases our algorithm constantly improves as compared to the greedy implementation. It's improvement as compared to greedy is capped at approximately 26% for TSPs with 100+ cities. It is able to take what branch and bound found in 30 seconds and improve on it until no improvements can be found via the 2-Opt and 3-Opt local search algorithm. Our algorithm excels at finding solutions quickly, even for large numbers of cities, though there is no practical way of knowing if the result discovered is the actual optimal solution or how close it might be. Since our algorithm solved up to 300 cities in 12 minutes, this algorithm is ideal if you need a quick and relatively good path in a short amount of time.

References

Flood[1956] M. M. FLOOD, ''The traveling-salesman problem,'' Operations Res. 4 (1956), 61-75

Croes[1958] G. A. CROES, ''A method for solving traveling salesman problems,'' Operations Res. 6 (1958), 791-812

[Papadimitriou & Vazirani, 1984] C. H. PAPADIMITRIOU AND U. V. VAZIRANI, ''On two geometric problems related to the travelling salesman problem,'' J. Algorithms 5 (1984), 231-246