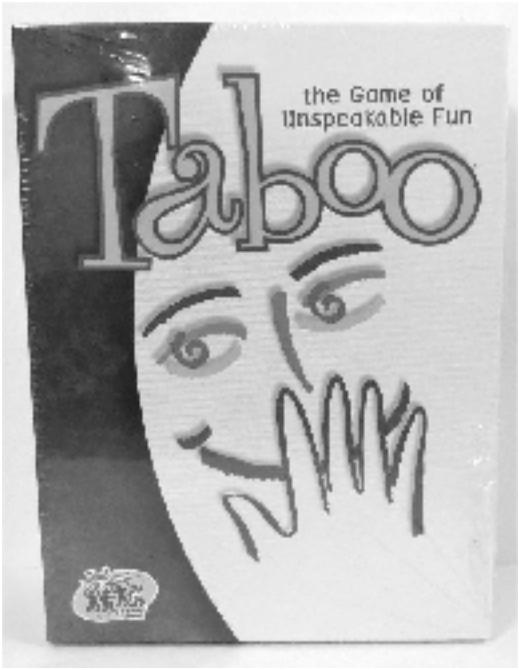
Travelling Salesman Problem: Tabu Search



(Anonymized)

April 2017

Abstract

The Tabu Search algorithm is a heuristic method to find optimal solutions to the Travelling Salesman Problem (TSP). It is a local search approach that requires an initial solution to start. Through implementing two different approaches (Greedy and GRASP) we plotted algorithm efficiency for various sized TSP problems to try and find an optimal solution.

Introduction

This paper reports the results after our valiant wrestle with an NP-Complete problem. We have implemented a Greedy algorithm, and three variations of the local search algorithm called Tabu Search. Our goal was to implement these variations to the point that they are consistently returning better results than our Greedy algorithm. We would like to express our thanks to Dr Farrell and the TAs for their assistance in the development of this project.

Greedy Algorithm

Description

The greedy algorithm was used to calculate the initial solution. To construct this solution, starting at the first city, a while loop checks whether or not all the cities have been added to the current route. The next city added to the path at each iteration is the city with the smallest cost from the most recently added city, effectively adding the shortest available edge each time. If a city is visited with no other paths to travel, the working solution is trashed and the algorithm starts again with a different city as the starting city. The greedy solution was also used to create a valid starting point for the initial solution to pass to the tabu search portion of the code.

Time Complexity

To put a time cap on the greedy algorithm is a stretch, but it resembles O(n^2) with the looping structure where each iteration the inner loop shrinks by one city. The greedy solution is challenging because of the nature of the Traveling Salesman Problem. Because TSP is NP-Complete, a solution can be verified in polytime, but the optimal solution is often unreachable or unverifiable in polytime depending on the size of the problem.

Tabu Search

General Definition

Tabu search chooses an initial state and sets it as the best solution. Then it creates an empty candidate list of each possible move, where each move is a switch of two cities in the best solution. Each of the candidates in a given neighbour which does not contain a tabu element are added to this empty candidate list. It finds the best candidate on this list and if its cost is better than the current best, it replaces the best solution. After a specified number of iterations any given tabu move will expire and will become a valid move.

Initial Solution Generation

When using a local search based algorithm like the Tabu Search, the initial solution is crucial in finding a good solution. We chose to use both a Greedy approach and a General Randomized Adaptive Search Process (GRASP) approach to generate our initial solution. (Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey, Sumanta Basu). A GRASP approach is basically the greedy algorithm modificated to be more random: instead of choosing the best solution among the neighboring vertices, it would randomly choose one among a certain percentage. The results improved but were dependent on the defined time constraint.

Tabu-search pseudo code

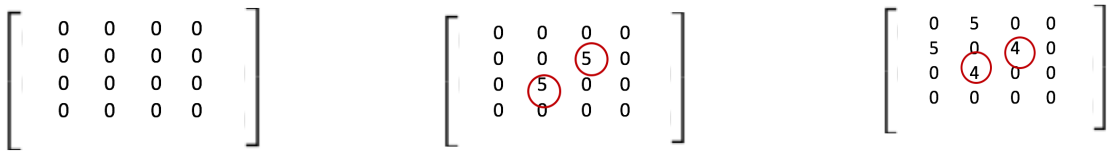
```
Get initial solution ( Greedy or GRASP ) and set it as the best solution.
Initialize a tabu list to keep track of what moves are eligible or
unavailable (‘tabu’) at each iteration.

While the user-defined constraint is not met:
      Iterate through the initial solution & find the best non-tabu switch
            that results in a cheaper route cost.
      Update & decrement the tabu list.
      If the move results in a better solution:
            Update the BSSF.
```

Tabu List

The tabu list is used to prevent cycles in the tour. In this context, a cycle represents the action of repeatedly visiting a certain set of solutions. The tabu list is also used to prevent the algorithm from being stuck at local optima. The tabu list is a matrix that represents all switches between cities. The list is initialized to zeros, and after a move is performed, the cell representing that

switch and its inverse is set to a limit (also known as tabu tenure). Each iteration of the algorithm will end with all cells in the tabu list being decremented by 1. Once the cell is at 0, the edge represented is eligible to be switched again.



In the above sample tabu list, it is initialized with zeros. After a switch of 2 and 3 are made, cells 2-3 and 3-2 are set to 5 (the tenure). In the next iteration, 1 and 2 are switched and set to 5, and the remaining are decremented.

Pro and cons

Pro

- Can move away from local optima towards a better solution.
- Better solution than the greedy algorithm.
- Fast, capable of producing solutions larger than those produced by branch and bound.

Cons

- The general success of the algorithm is dependent on the constraints set by the user (time, iterations, etc.)
- Depending on the above conditions, the optimal solution will rarely be found.
- GRASP solutions are initially random; while this ultimately helps move towards the optimum, some individual solutions have a cost that is far too high to even consider.

Time Complexity

Pinning down time complexity for an NP-complete problem is tricky. While solutions to NP-complete problems can be *verified* in polynomial time, there is no clear metric on how that solution is reached in the first place. That being said, the core components of the tabu search algorithm can be measured.

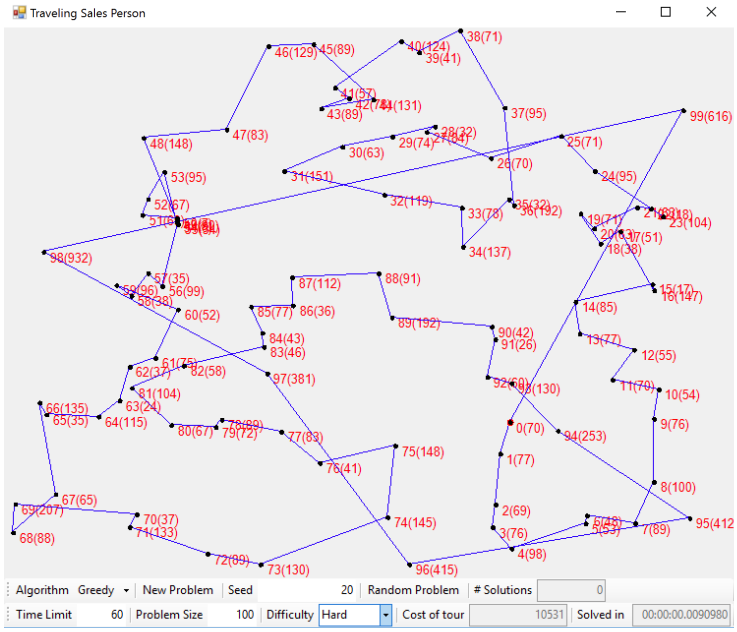
The key function in the tabu search that is repeatedly called is the getBestNeighbour function, the time complexity of which is O(n^3), where n is the number of cities in the current problem. This function iterates through the solution and attempts to improve the BSSF by performing every possible switch of two nodes (n^2). The 3rd n comes from calculating the total cost of the route after switching two cities by calling getTourCost(). After determining the best switch to make, getBestNeighbour() calls both the decrementTabu() and the tabuMove() functions to update the current tabu list before returning the updated path, which are both O(1)

functions. The cost of the path is then compared to the cost of the BSSF, and if the immediate cost is better, the BSSF is updated.

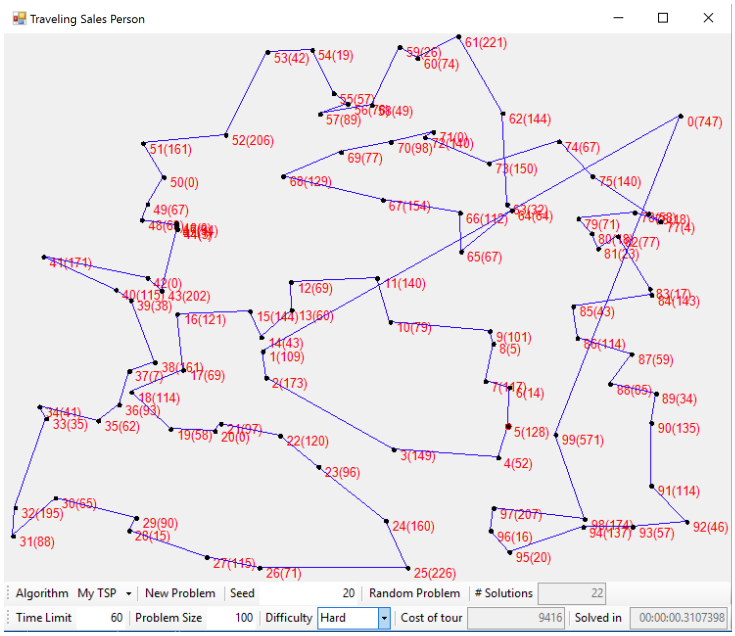
The trouble in predicting the theoretical time complexity arrives when trying to plug in this function to optimize the initial solution. To combat this, we performed three different variations of tests to chart out the performance of each different algorithm. First, we used the greedy algorithm as the initial solution and constrained the algorithm to 100 iterations before timing out. This approach was quick, but often returned costs insignificantly less than the initial solution. The time complexity of the greedy algorithm is outlined above. For the second and third groupings of data, we used the GRASP process to obtain our initial solution, which has the same time complexity as the greedy algorithm.

Screenshots

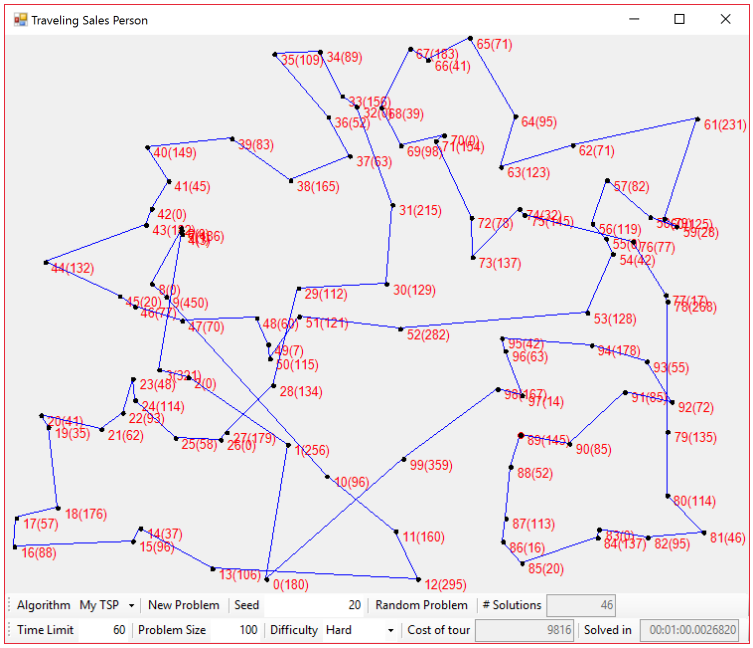
Greedy



Tabu with greedy initial solution



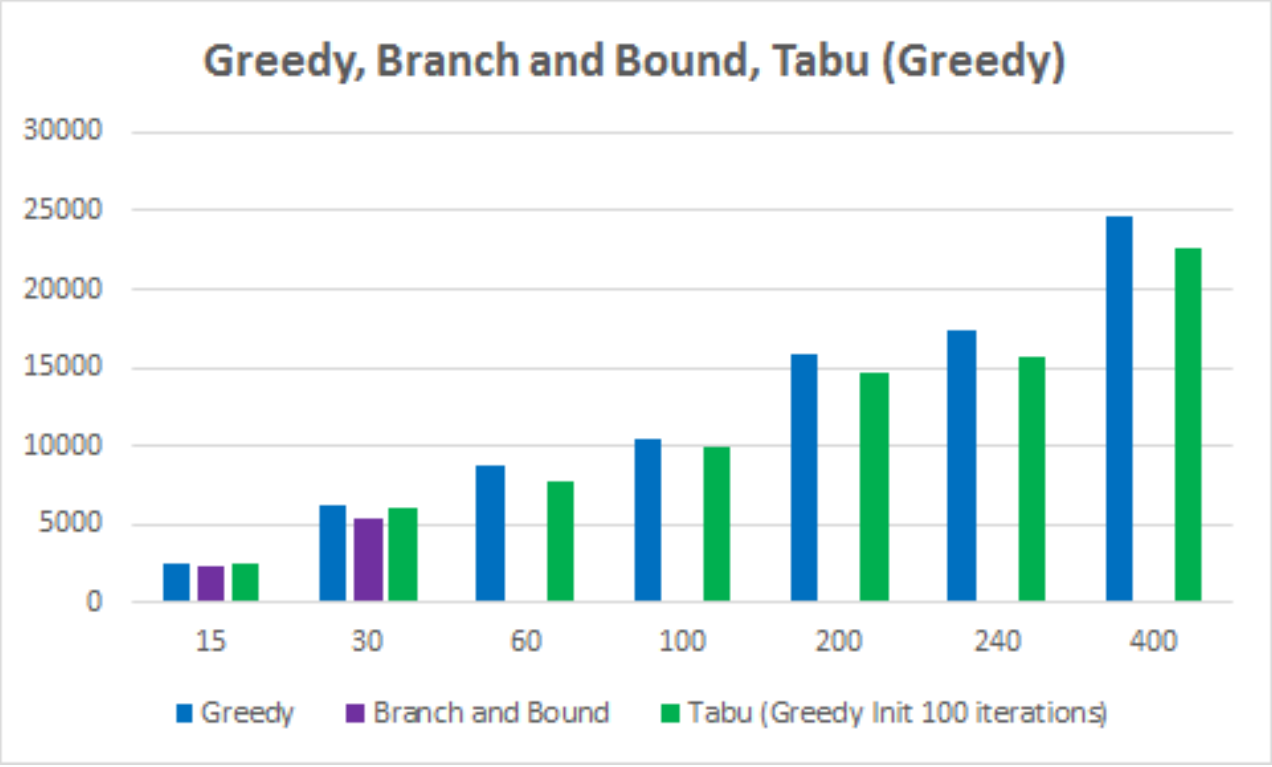
Tabu with GRASP initial solution

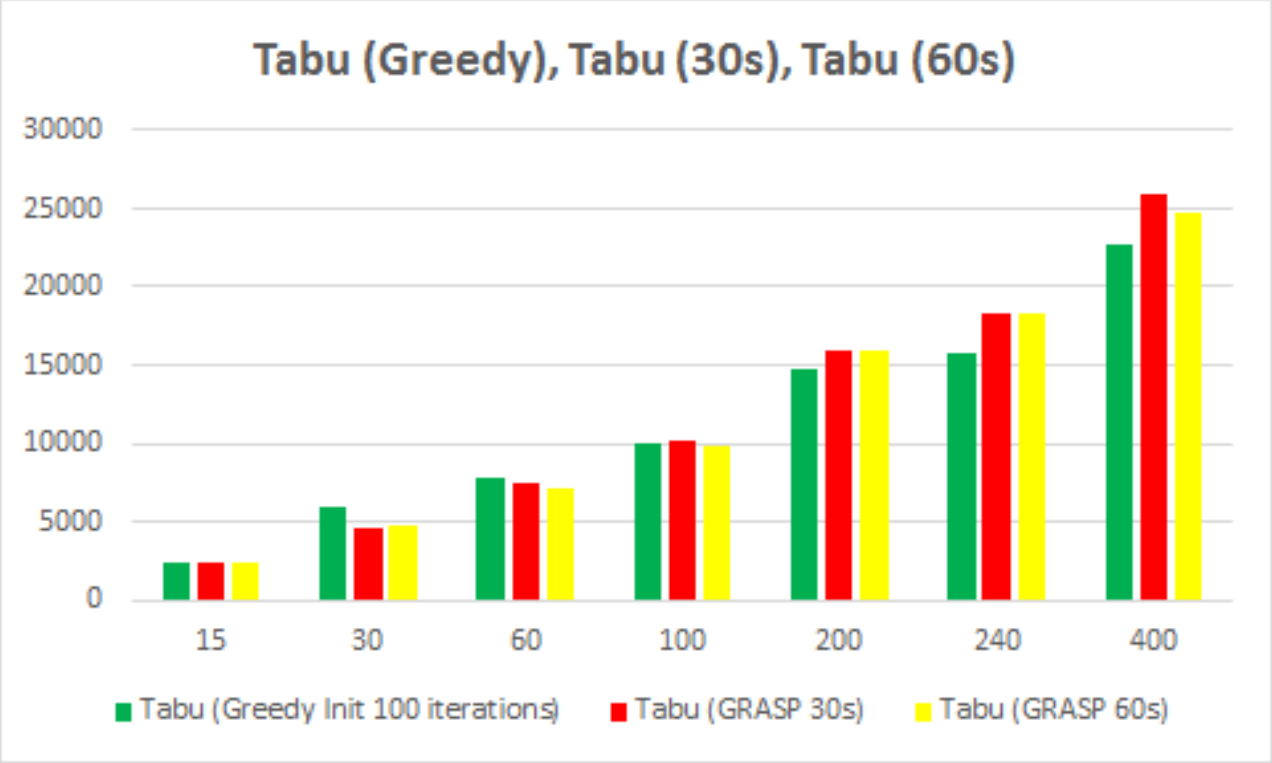


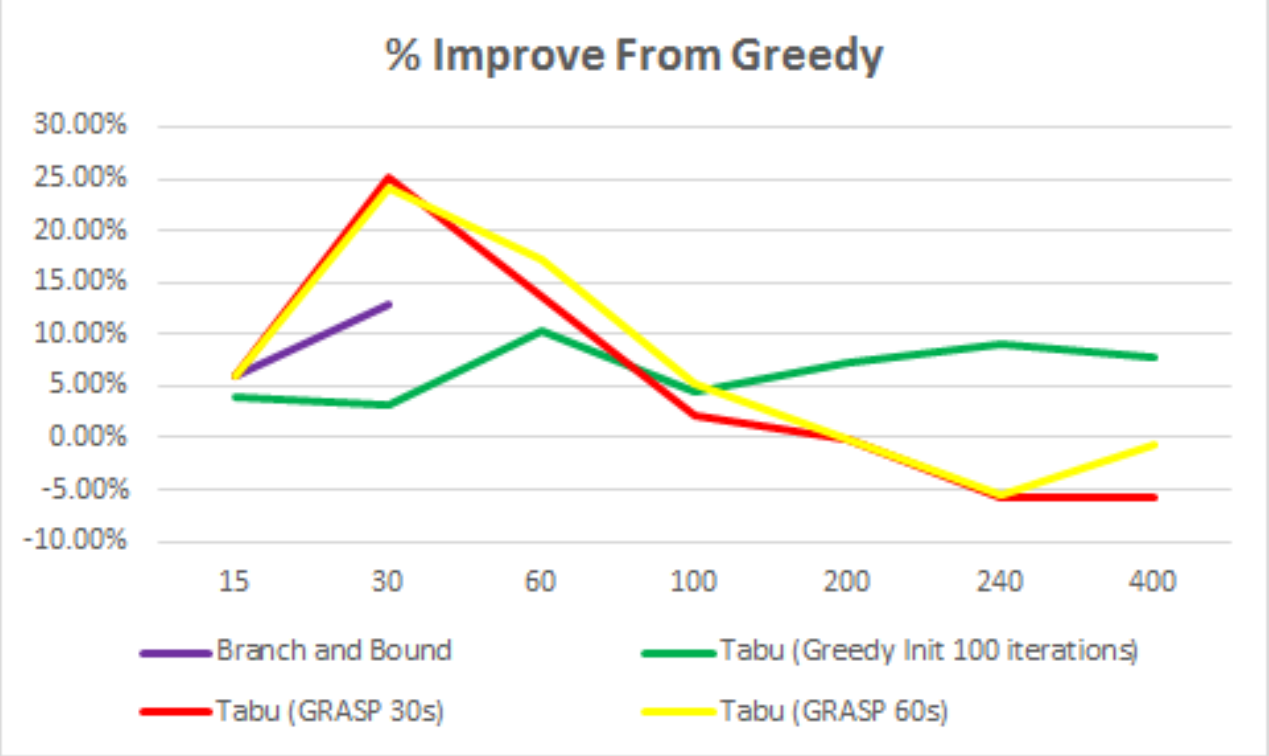
Data

	Random	Greedy			Branch and Bound		
# Cities	Path Length	Time (sec)	Path Length	% Improve	Time (sec)	Path Length	% Improve
15	5741	0.0030569	2528	55.97%	0.39558835	2375	6.05%
30	12428	0.0011587	6164	50.40%	600	5378	12.75%
60	31241	0.006	8678	72.22%	TB	TB	TB
100	54113	0.0051455	10393	80.79%	TB	TB	TB
200	106799	0.0547846	15851	85.16%	TB	TB	TB
240	125520	0.0647279	17302	86.22%	TB	TB	TB
400	204494	0.2028053	24575	87.98%	TB	TB	TB

	Tabu (Greedy Init 100 iterations)			Tabu (Fudged 30s)		Tabu (Fudged 60s)	
# Cities	Time (sec)	Path Length	% Improve	Path Length	% Improve	Path Length	% Improve
15	0.0015486	2429	3.92%	2375	6.05%	2375	6.05%
30	0.0266206	5974	3.08%	4619	25.06%	4683	24.03%
60	0.0609056	7775	10.41%	7483	13.77%	7191.2	17.13%
100	0.3109223	9922	4.53%	10170	2.15%	9839	5.33%
200	1.4515651	14715	7.17%	15878	-0.17%	15890	-0.25%
240	2.3626726	15748	8.98%	18306	-5.80%	18268	-5.58%
400	11.4777217	22689	7.67%	25983	-5.73%	24726	-0.61%




Tabu (Greedy), Tabu (30s), Tabu (60s)
30000
25000
20000
15000
10000
5000
0
15
30
60
100
200
240
400
Tabu (Greedy Init 100 iterations)
Tabu (GRASP 30s)
Tabu (GRASP 60s)


% Improve From Greedy
30.00%
25.00%
20.00%
15.00%
10.00%
5.00%
0.00%
-5.00%
-10.00%
15
30
60
100
200
240
400
Branch and Bound
Tabu (Greedy Init 100 iterations)
Tabu (GRASP 30s)
Tabu (GRASP 60s)

Analysis

Greedy, Branch and Bound, Tabu

Although the tabu solution does not necessarily find the optimal, it can run more cities than the branch and bound. Branch and bound will find the optimal solution, however, the solution takes a long time to find and needs a lot of resources. It quickly becomes unreasonable. The greedy algorithm is significantly better than the default and even for a large amount of cities it is fast. The greed solution is not typically optimal. The tabu search is significantly faster than the branch and bound but not as fast as the greedy. The tabu search takes a bit more time than the greedy, but still runs quickly to find a better solution than the greedy.

Tabu: Best Initial Solution

The tabu search calculated using an initial greedy solution (“greedy tabu”) originally is worse than the tabu search calculated using an initial GRASP solution (“GRASP tabu”). However, as the number of cities increases, the greedy tabu eventually becomes better. In order for the GRASP tabu to compete with the greedy tabu, the time that it is allowed to run needs to be increased. Given enough time the GRASP tabu will always perform as well as or better than the tabu search with an initial greedy solution. Some of the variance in the GRASP tabu 30s and GRASP tabu 60s is due to the randomization of the initial solution. The longer the algorithm is allowed to run the more likely it will generate a more efficient initial solution.