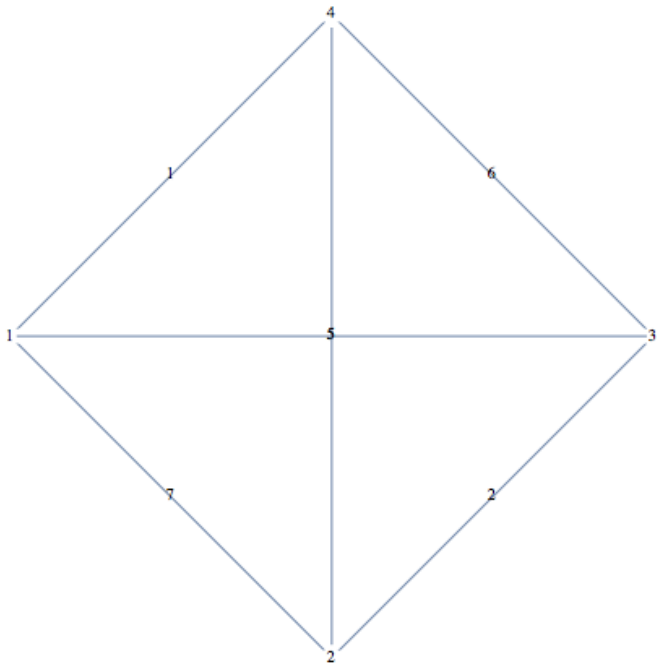
Traveling Purchaser Problem: A Greedy Approach

The Traveling Purchaser Problem, a generalization of the Traveling Salesperson Problem, is an NP hard optimization problem. To explain the Traveling Purchaser Problem(TPP), I will start with the Traveling Salesperson Problem(TSP). In TSP, the story goes, there is a salesperson that needs to travel to a set number of cities. The roads are represented as a connected and with the edge weights representing the lengths. The salesperson must find the shortest route that visits all the cities once and ends in the city of origin. In the example below, there are four cities at the nodes and the connections between them are the edges with the lengths printed on them. This is a simple enough problem that with a little bit of thinking one can find that the shortest path would be 1 → 4 → 2 → 3 → 1 with a distance of 13 units. It is easy to see how this would be much more difficult to do as the problem got larger. (Traveling Salesman, Wikipedia)



TPP is a similar problem, except each city has a market with a certain number of goods and the edge weights represent cost instead of distance. Each good can be priced differently in

each market. The Purchaser has a list of goods that must be purchased and must determine which markets to buy the goods to spend the minimal amount of money. Transportation costs have to be factored in but it is not necessary for the purchaser to visit every city as long as all the goods are bought. (Traveling Purchaser, Wikipedia)

The Traveling Salesperson Problem has been studied for a long time and has been shown to be NP hard. An NP problem is a decision problem where the answer can be verified (not solved) in polynomial time. Polynomial time refers to a big O notation, or a limit, of n^k calculations, where n the size and k is a constant. Polynomial time is usually thought of as fast. A problem that is NP hard is not necessarily NP, but it has been shown to be at least as hard as solving the hardest NP problems. This means that finding the solution is among the most difficult, or time consuming, to solve of all the NP problems. (NP- Hard, Wikipedia) For the TPP where there is exactly one unique good in each city, it is equal to the TSP problem and therefore TPP is NP hard as well. (Traveling Purchaser Problem, Wikipedia)

I looked at a subproblem of the TPP where every market has all of the required goods and each market has enough of each good to satisfy all of the purchasers requirements. This helps simplify the brute force algorithm. To find the optimal solution I looked at all the ways that each good could be selected. For the shown example, it would be all the ways that one could choose an item from each column, 27. This would be $Number\ of\ Cities^{Number\ of\ Goods}$ possibilities. It is unnecessary to look at situations where the purchase of one good is split over multiple markets since if two markets have different priced goods it is always cheaper to buy all the goods at the cheapest market. If a good is the same price in multiple markets, then there is no benefit from splitting it up. To determine the cost of transportation I used Dijkstra's algorithm. This algorithm finds the shortest path from one node to

	Item 1	Item 2	Item 3
City A	2	2	5
City B	4	3	1
City C	1	3	2

all other nodes. This algorithms has a complexity of O(E + VLog(V)) where E is the edges or roads and V is the vertices or the cities. (Dijkstra, Wikipedia) Since this has to be called for every vertex, it would have a complexity of O(V E + V^2 Log(V)). Then the complexity for finding the optimal solution for this subproblem would be O(V E + V^2 Log(V) + V^M) where is M is the number of goods at the market.

To find a solution to this problem for cases that are too large for the above method, I proposed a heuristic that uses a greedy algorithm to rate all the goods in each market and choose the one with the best rating. This means that all V^M goods are ranked and the purchaser goes to the market of the best ranked good and buys all of the required goods. The remaining V^{M-1} are then reevaluated from the purchaser's current position and the next item is selected. This continues until all of the items have been selected. This approach wouldn't necessarily find the best result, but if it could be shown to be close to the optimal solution for a large percentage of the times it is run, it could be very useful. There already exists several heuristics for TSP. (Dijkstra, Wikipedia)

To rate each good, it is important to identify the important factors that would indicate which were the best goods to buy. The factors that are easy to identify are cost of buying the good (times the required quantity of that good) and the cost of traveling there. Yet the ranking should also be aware of the quality of the destination. It would also be helpful to look at the number of connected edges since they suggest how central the city is. Another factor would be the average price of goods in the market, if this value was low then it would be likely that the purchaser would be able to buy more than one good at the market. If purchaser did have leave, it would be helpful if the average cost of transportation around the city was low. Looking at all these factors, score for each good would be;

$$a \cdot \#Edges + b \cdot \overline{MarketPrice} + c \cdot \overline{TravelCost} + d \cdot TravelCost + e \cdot (Qty \cdot Price)$$

The algorithm would buy the item that had the lowest score. Since the first three factors are all related to the quality of the location, these could all be grouped together so that the importance of quality can be as a coefficient. That would change the above expression to;

$a \cdot Quality + b \cdot TravelCost + c \cdot (Qty \cdot Price)$, where

$Quality = x \cdot \#Edges + y \cdot \overline{MarketPrice} + z \cdot \overline{TravelCost}$. Therefore for M unique goods, this algorithm would have to rank V cities. This algorithm has a complexity of O(V E + V^2 Log(V) + V M) = O(V(E+ M) + V^2 Log(V)), which is an algorithm of polynomial time.

I decided to use a genetic algorithm to optimize the real coefficients a,b,c,x,y,z. A genetic algorithm is an optimization method that is based on the concept of natural selection. In this case a population member would be the array (a, b, c, x, y, z). To execute a genetic algorithm, there is an initial population of randomly generated members. A ranking system is used to determine the most fit members. This is followed by a selection process that normally chooses the highest ranking members of the population to persist. To add variation to the new population some of the selected members are mixed to simulate breeding and random mutations can be added. Once the new population has been built, the whole process is repeated. Each cycle represents a generation and if all the conditions are right, then if the process is run for a large number of generations, the final population should be much more adept for the ranking system than the initial one.

In this case, the ranking system has been already been determined. For each generation a certain number of randomly generated graphs will be created, lets call this number matchSize. These graphs contain all the information of the TPP including market data. Each population member will try to determine the optimal purchasing path for all of the given problems. Their solutions for each problem are added together and this provides the score for each population member.

For the selection method I decided to use the Roulette wheel selection method. This means that the probability of a member being selected to move on to the next generation is proportionate to its score. This leaves a small chance that the weaker members will be selected, which can be good as some of their attributes may be useful when mixed with a stronger member. (Fitness Proportionate Selection, Wikipedia) To determine each members chance, I subtracted each score from the maximum score. This means that the member that found the lowest (best) priced solutions is the farthest from the maximum score and therefore it would have the largest percentage. The algorithm would then pick a number between 0 and the total of all the chances to see which one was selected. For example, if three members had scores of (1, 3, 5), they would be given a chance of (5 -1, 5 - 3, 5 - 5) = (4, 2, 0). Then if the random number was less than 4, the first member would be chosen, and if it was between 4 and 6, the second member would be chosen.

After two members are selected, there is a percent chance that they will breed. Let's call this breedingRate. If they breed, then the offspring will be carried on instead of the two parents. For the breeding method I used Uniform Crossover. This creates each offspring by looking at each characteristic, in this case each coefficient, and randomly choosing which one to use from each parent. (Crossover, Wikipedia) This seemed to make sense as there was no clear logic as to how to mix the coefficients. Ideally if one parent has good coefficients in one area and the other has it in another area, that eventually the right coefficients would be chosen from each parent. After breeding is determined, the algorithm checks to see if the selection will be mutated by using a probability given by the mutationRate. If a member is mutated, a random selection of the six coefficients have a random real number added to them.

In the first attempt to optimize the proposed heuristic, I used the following constants:

matchSize	10
populationSize	10
breedingRate	.5
mutationRate	.3
generations	50,000

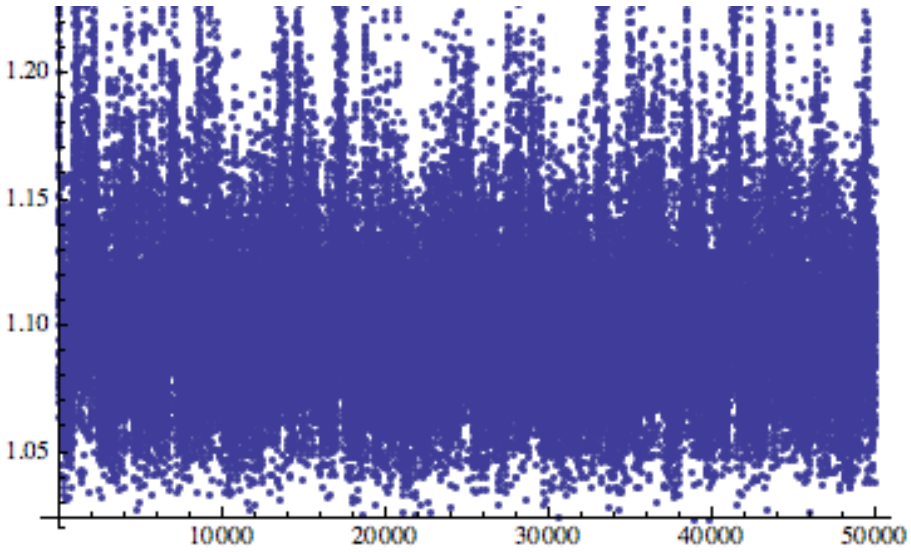
Further Details include:

Random Graph: BernoulliGraphDistribution

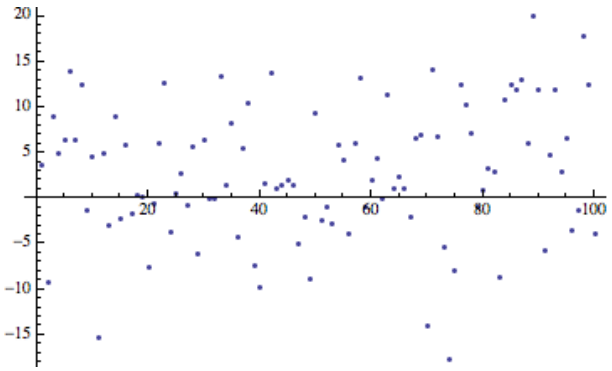
- 5 nodes,
- 5 market items
- Probability of an edge between two node: uniformly distributed between .2 and 1

The initial random population was generated using random real numbers between -10 and 10 and mutations added a -1 or a 1.

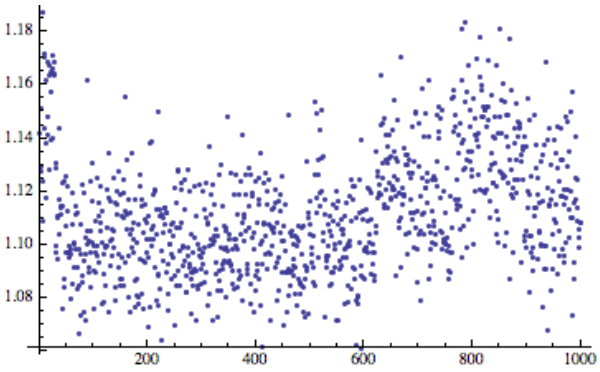
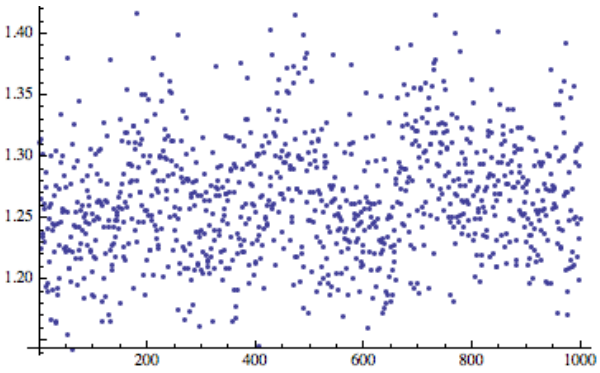
To track the progress of the populations over time, I had the highest ranked member from each generation compete against the first algorithm (the brute force approach) using the same system that ranks the population members. I then returned the heuristics results divided by the optimal solution.

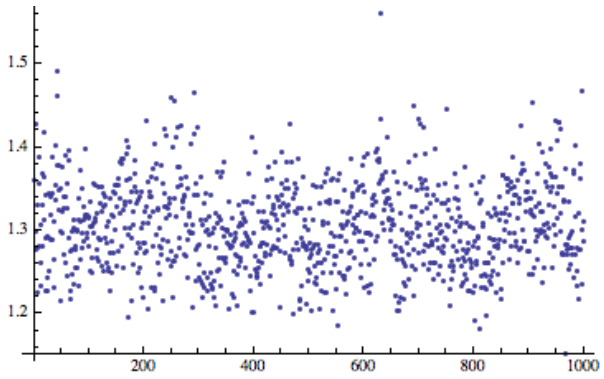
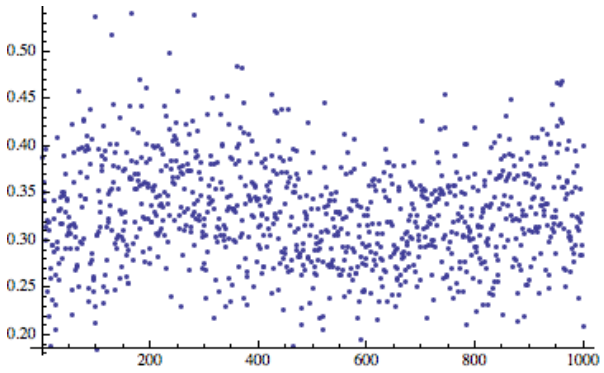
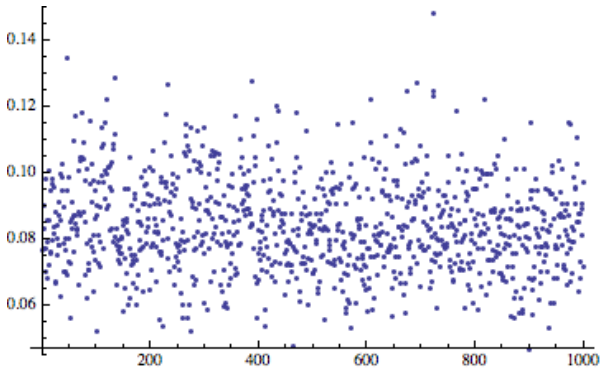


As is clearly evident, there does not appear to be any sort of convergence happening. The relationship between the optimal solution and the best heuristic for the data seems to be fairly random with central tendency toward the ratio near 11:10. To get a better idea of how well this was doing, I analyzed the final, returned set from the above calculations against the mean score from all the possible prices for 100 random graphs (just take the average score instead of the minimum score for the optimization algorithm). The y-axis represents the distance from the mean, with positive scores meaning doing better than average. The data seems to be fairly evenly spread around the mean, further implying that the performance of the heuristic was not much better than random.



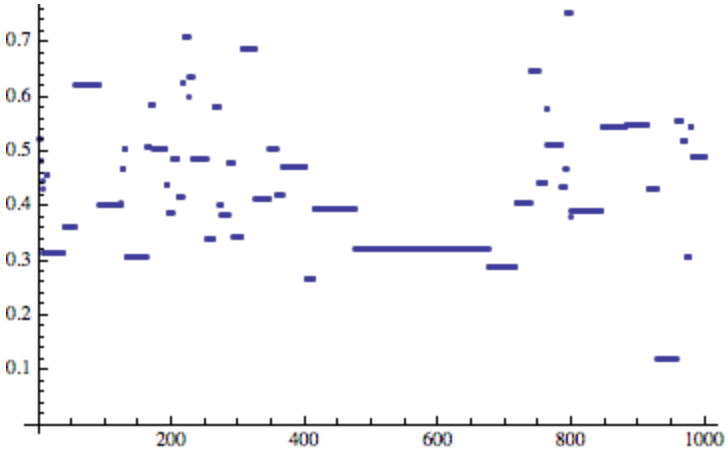
I then proceeded to to try a number of variations to see the effects.

In case the best solution was never making it into the new population without being breed or mutating, I changed the algorithm to automatically add the winning solution into the new population before starting selection. This would not keep it from being selected again and then mutated.	
Updated the formula: $a \cdot distance + b(quantity \cdot Price) - c \cdot \#Edges$ $d \cdot \overline{MarketPrice} + e \cdot \overline{TravelCost}$. This way the coefficients could be ratios.	

breedingRate = .7 mutationRate = .3	
New Formula: $\frac{a(b \cdot distance + c(quantity \cdot Price) + d \cdot}{MarketPrice + e \cdot TravelCost)/\#Edges}$ This new arrangement should allow the coefficients to be relevant despite the graph data. In the last formula, if the market costs and travel costs were not very high, then the number of edges would become less important despite its coefficient. (Y axis now shows the % error)	
New Formula: $\frac{a \cdot distance + b(quantity \cdot Price) + c \cdot}{MarketPrice + d \cdot TravelCost}$ The number of edges seemed to be hurting the results more than helping it.	

None of the changes that were made seemed to affect the scattered random nature of the results. The only noticeable chances occurred with changes to the arrangement and number of factors looked at. Looking at the last two tries, the formula without the factor for the number of edges has a noticeably better percent error than the prior run, despite the fact that the general shape is the same. This implies that the chosen factors have an affect on the outcome, but the coefficients have little effect.

To further investigate this thought, I tracked the changes in coefficients over the full run of the genetic algorithm. To display this, I graphed the mean of the winning coefficients for each generation. Wherever there is a straight line, that means that the same set of coefficients is winning consecutive generations.



By looking at the above graph for the generations between 500 and 600 you can see that the same set of coefficients was winning every time. Yet, if you look at the graph in the fourth row of the table, you see that the ratings for those generations are just as varied as anywhere else. This proves that the coefficients are not affecting how well the heuristic does in relation to the optimal solution. There might be some coefficients that are better than others, but the percent error is going to vary depending on the difficulty of the problem.

It would seem that a further investigation into other factors might be able to diminish the average percent error, but the cases are too varied to find optimized coefficients. There may have been improvements that could have been made to the genetic algorithm, but I believe the last graph shows that the results are going to be the same despite the coefficients and therefore the genetic algorithm is not worth the time. While it has been shown, to a small extent, that the design of the greedy algorithm can affect the average percent error of its result, I believe that the TPP might be to complicated for a greedy approach and require a more sophisticated heuristic.

Sources

"Crossover (genetic Algorithm)." *Wikipedia*. Wikimedia Foundation, 18 Apr. 2014. Web. 10 Apr. 2014.

"Dijkstra's Algorithm." *Wikipedia*. Wikimedia Foundation, 24 Apr. 2014. Web. 6 Mar. 2014.

"Fitness Proportionate Selection." *Wikipedia*. Wikimedia Foundation, 15 Mar. 2014. Web. 19 Apr. 2014.

"NP-hard." *Wikipedia*. Wikimedia Foundation, 23 Apr. 2014. Jan. 30 Apr. 2014.

"Traveling Purchaser Problem." *Wikipedia*. Wikimedia Foundation, 07 Mar. 2013. Web. 17 Jan. 2014.

"Travelling Salesman Problem." *Wikipedia*. Wikimedia Foundation, 16 Apr. 2014. Web. 17 Jan. 2014.