Optimization Analysis

Kenny Darrell

Meta-Heuristic Approaches to the Traveling Salesman Problem

The Traveling Salesman Problem has a rich history dating back almost two hundred years with no sign of fading interest. This is due to its oxymoronic nature of being very simple to understand even for a child, yet being in the NP-Hard class of problems it can cripple supercomputers. The problem consists of having to travel through every node in a collection only once resulting in the shortest distance. There are various rules that can be applied or omitted as well. A symmetric problem is where the distance from node A to B is the same as B to A, which is asymmetric otherwise. There is also the complete tour where you must return to the start opposed to just visiting each, which can yield different solutions, as well as other criteria. Analysis will be performed on various size problems ranging from small to large city instances of symmetric complete tours. Comparing solution methods and outcomes of different approaches should allow for some insight in the difficulties of the problem.

On first thought one solution method would be to compare every possible tour and take the minimum as the optimal solution. On further investigation a 10 node tour has 3628800 possible tours while 20 nodes has roughly 2.4329\*1018 and 100 nodes has roughly 9.3326\*10157 tours. This approach clearly falls apart very fast because the number of tours grows at a factorial (x!) rate while in extreme cases an instance could have 100,000 cities. Various exact methods have been created to optimally solve this problem based on branch and bound and cutting plane algorithms but are still unable to find solutions in realistic times. Heuristic approaches are capable of finding “good” solutions in “short” time periods as they do not check all tours but do not guarantee the best solution. Meta-Heuristic approaches are heuristic approaches that are able to jump out of local optimum situations, causing them to at times be worse than earlier solutions but being more robust. In this examination the quality of these solutions will be examined as well as the run time. Four different approaches will be under consideration; Ant Colony/Swarm Intelligence Optimization, Tabu Search, Simulated Annealing, and Genetic Algorithms. A collection of Traveling Salesman Problem (TSP) instances of varying size will be solved with each algorithm. The instances have been taken from the TSPLIB1 and can be seen in the appendix. For each instance the runtime, tour length, and tour will be recorded for each algorithm. These are compared to the known optimal solutions which have been proven to be exact. From here an algorithm can be gauged on how good the solution is and how long it took to produce.

The Ant Colony Optimization (ACO) method is a type of Swarm Intelligence that uses a group of ants that randomly travel through the nodes, with no repeats. After the collection of ants has completed their tour the lengths are compared to find the shortest. This tour is then given a weighting, pheromones are deposited on this trail. The process has iteration, but now the uniform random selection from a given node to all other nodes has changed to have a larger chance of choosing the segment from the previous best solution. After this round some of the pheromone from the previous round will evaporate, the increased weight is reduced, and more pheromone is laid onto the new best tour. The amount of pheromone to lie and how quickly it evaporates affects the convergence rate of the solution as well as its ability to jump out of a local optimal tour.

The Genetic Algorithm (GA) is a type of Evolutionary Algorithm which creates a population of tours based on a greedy approach. This means from the current node select the closest non-repeat node. The initial part of the tour will be very good but the later portion can have very large segments. A group of parent solutions are then selected which are the shorter than the rest of the population. These two parents will be combined to make two child tours. In order for this to create a full tour the cutting and merging has to be done in a manner that results in two full distinct tours. This can be done by taking the shorter segments from A and B and doing a random substitution to complete the tour or via a method of miscarriage that discards infeasible child tours until a child is a complete tour. These children tours replace two of the longer tours in the population. This process of selecting parents and merging them to create child tours is then iterated until some criteria has been met. To stop the process from converging to an identical population a mutation is implemented. The mutation is set so that it will occur at random iterations. The mutation can be set up in a number of ways such as taking half of one parent and then a greedy approach for the rest of the tour, an entire new greedy tour, or replace a group of segments with a shorter combination of the same segments. The control parameters here are population size, number of parents to select and create children from, and the percentage of times that reproduction will be mutated.

The Tabu Search is a meta-heuristic approach that operates on a single tour as opposed to a group of tours. The method starts out with an initial feasible tour, possibly greedy or randomly produced. The process will then look at groups of four or five (larger numbers become difficult for computation) continuous node segments and implement sub-tour reversals on them. The sub-tour reversal for a segment of four consecutive nodes (ABCD) with segments (AB, BC, CD) could reverse BC to give consecutive nodes of (ACBD) with segments (AC, CB, BD). After looking over the all sets of consecutive nodes a reversal is selected that will give the best reduction in tour length. If all reversals lead to an increased tour length the smallest increase will be selected, allowing the procedure to jump out of a local optimal solution. This reversal is added to a Tabu list which will not be a legal reversal for a future iteration. This Tabu list can be finite so after a certain number of iterations a reversal will be removed from the list. A reversal can be selected if it is on the list if will lead to a solution which is smaller than any other seen tour.

The Simulated Annealing meta-heuristic that relies on a probabilistic approach to finding the best tour. A feasible is required to start the algorithm. All tours in the instance are thought of as a state of some system. Each state has an associated Temperature, the tour length, and the minimum Temperature is sought by changing the state of the system. A move (four node reversal from above) from one sate to another is made at step. A segment is selected for the move at random. The Temperature of the new tour is subtracted from the new tour. If this difference is positive the move is made. If the difference is negative, the tour is worse; a draw from a uniform distribution is made for the selection. The value ex, where x = (Zn-Zc)/T, is the cutoff point for acceptance of the move were Z is a temperature and n and c denote new and current states. The T is a schedule that decreases over time, so in earlier iterations the process can jump into much worse tours. This schedule will eventually go to zero and be similar to Tabu in that it can only make better moves.

Each above method was implemented into an algorithm to be called as a solution approach. They are implemented in the Matlab framework. Each approach originated with a TSP application from the Matlab User Community2. The application was stripped of all code except the algorithm itself, and then wrapper code was added. The wrapper will allow for a functional calling of only the solution approach, while stripping any plotting, initialization, or graphical use interface code. From here the approach is not biased when calculating solution time. The simulated annealing code was never able to get off the ground. The application seemed to work, plots were created as the process iterated and commanded for user input for termination if a current solution was acceptable. The algorithm was never able to stand on its own though, even after termination criteria was added and all user directives were removed. Long periods of time would result in almost a noise as a solution. When further investigation was done on the application it was very slow as well. Output plots were showing temperature changes over time, moves made, and how the solution was advancing but not showing good results for a tour. This method was then scrapped.

The solution method will involve loading a TSP instance into Matlab, calculating its distance matrix, calling the implemented function to solve the instance, and saving all return data. This is all done with an initialization script. This will also call a running time method at the same instant as the function call and provide cpu time stopping at the moment execution is returned to the script. When the execution is returned the time, tour and tour length is saved into a structure indexed by instance size and algorithmic approach. All data is then cleared to give the next approach the same starting conditions. After all approaches have been run for a given instance plots are created along with the known optimal solution. Another script will iterate through feeding the initialization tool all of the TSP instances.

The results are shown for the 16 (Ulysses16), 29 (bays29), 51 (eli51), 70 (st70), 96 (gr96), and 225 (tsp225) city instances in Figures 1-6 respectively. For each figure the optimal tour is shown, as well as the Ant Colony, Genetic Algorithm, and Tabu Search solutions. For smaller instances the heuristic tours look to be fairly close to the optimal tour, but as the size increases they seem to drop off. The Ant Colony and Tabu Search methods drop off much slower than the Genetic Algorithm which seems to deteriorate to almost noise, a random solution, by the 225 node instance. These plots are fairly similar to all output plots. The plots are a very good visual off what is happening but to say anything concrete the data must be checked. In Figure 7, the difference from the Meta-Heuristic approach to the known optimal solution can be seen. It is very clear from this that the Genetic Algorithm performs much worse than the others. It also seems that the Ant Colony approach is consistently in second place. This would lead one to believe that the a solution coming from the Tabu Search would be better than the other methods if new cases were to be performed and one method needed to be chosen. One point of interest though is that the Tabu Search has a tour length that is smaller than the optimal solution. How could it outperform a proven best, at most it could tie the method with the same or an alternate optimal equivalent solution. The difference is almost a 10% reduction. In looking at the plot there are too many data points to see clearly but it looks as though the Tabu Search could be in certain areas but it crosses over itself, such an occurrence has been proven to be incapable of being in the optimal solution. The data has either been corrupted in Matlab or an error in the data file.

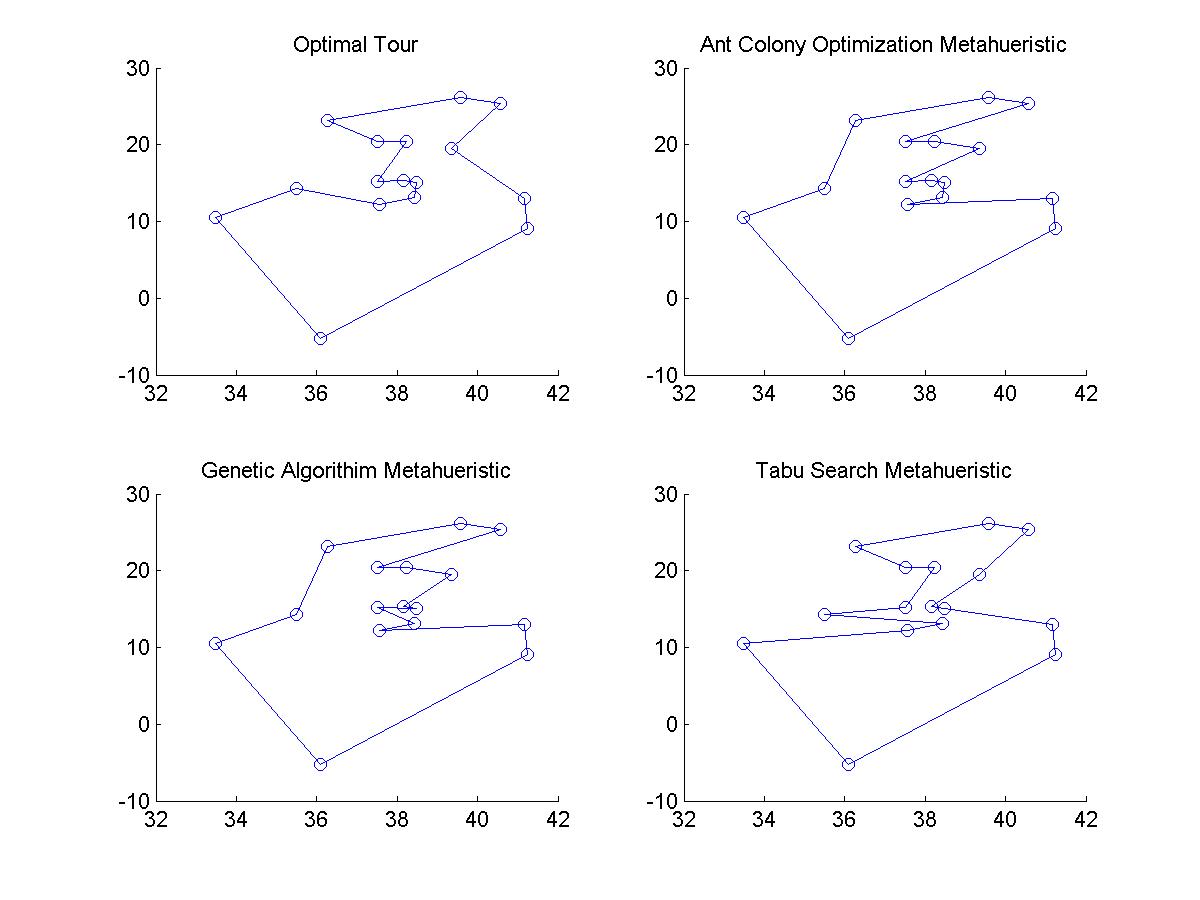


Figure 1, Varied Approach Optimal tours of the Ulysses 16 Node TSP Instance

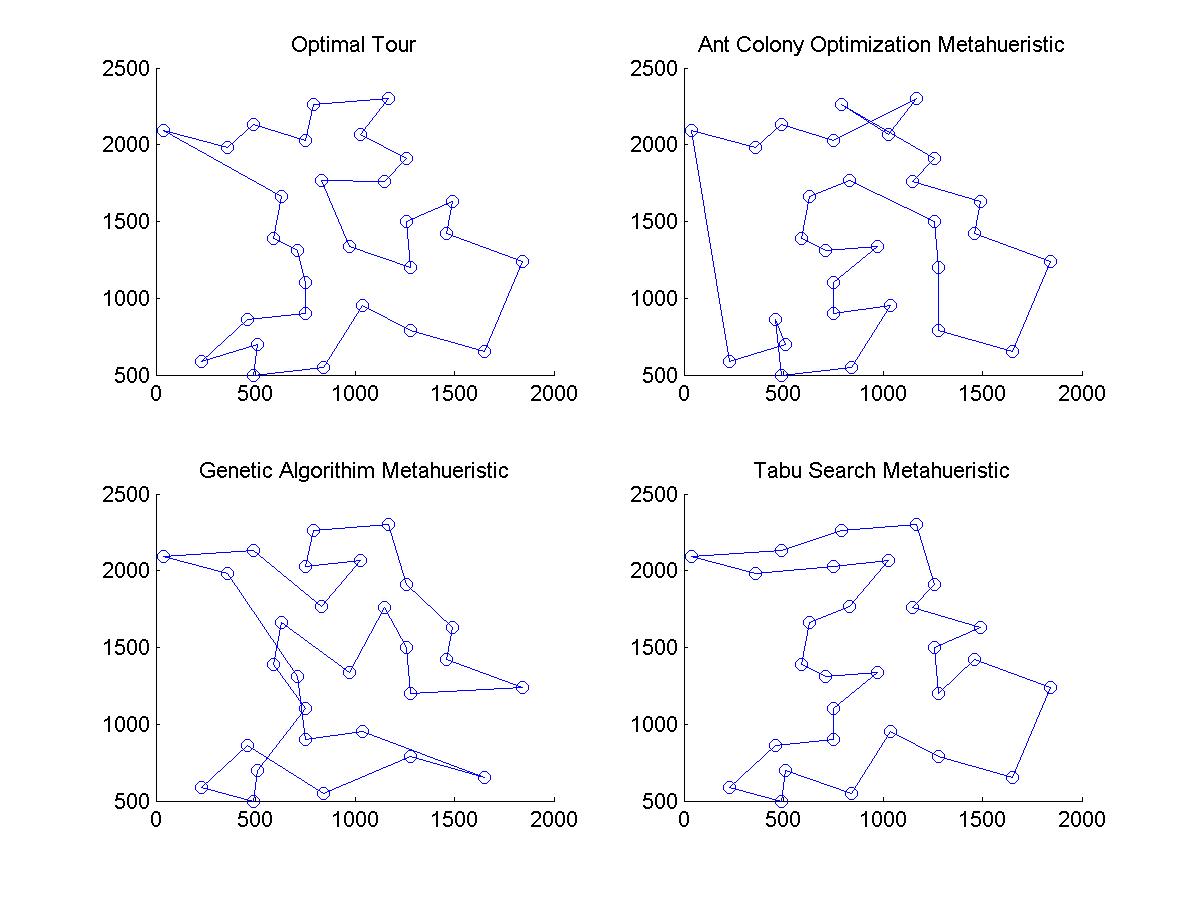


Figure 2, Varied Approach Optimal tours of the Bays 29 Node TSP Instance

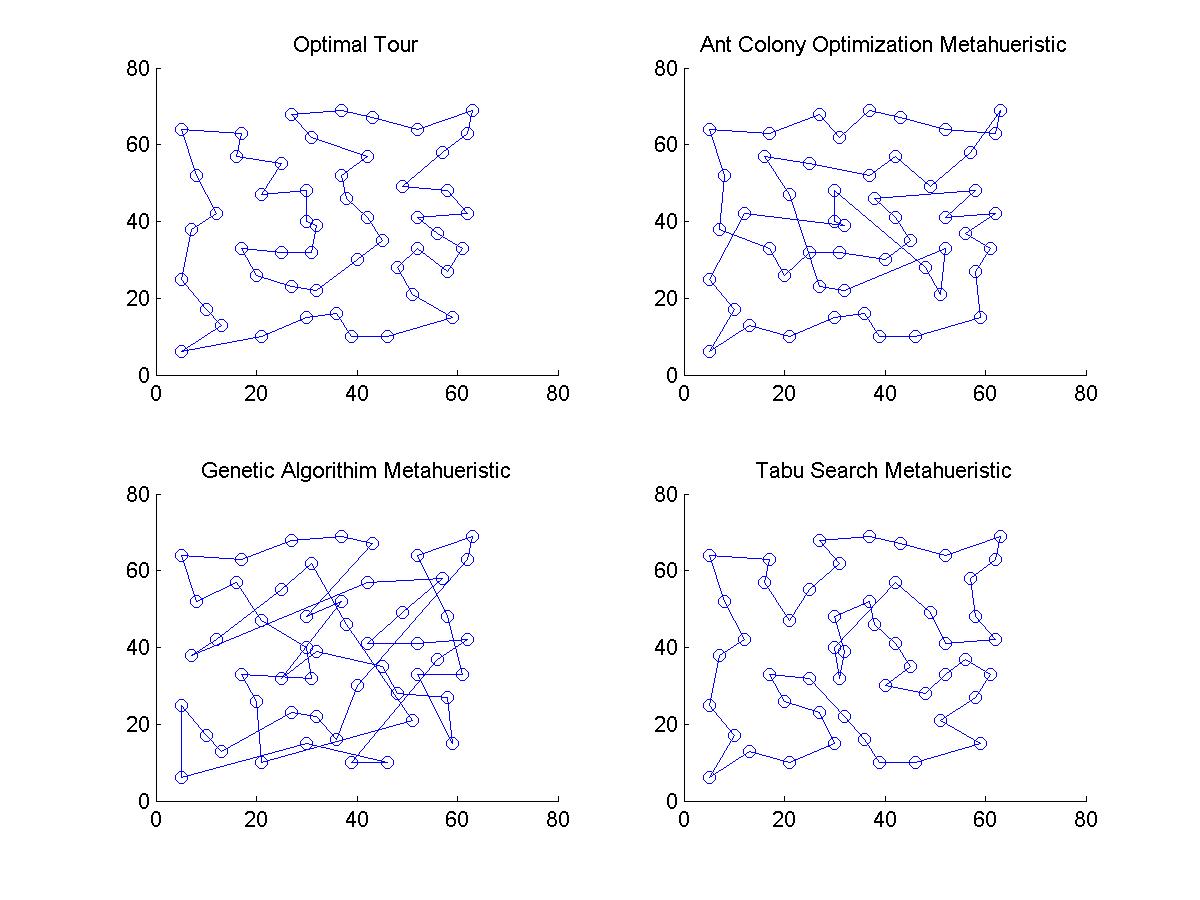


Figure 3, Varied Approach Optimal tours of the Eli 51 Node TSP Instance

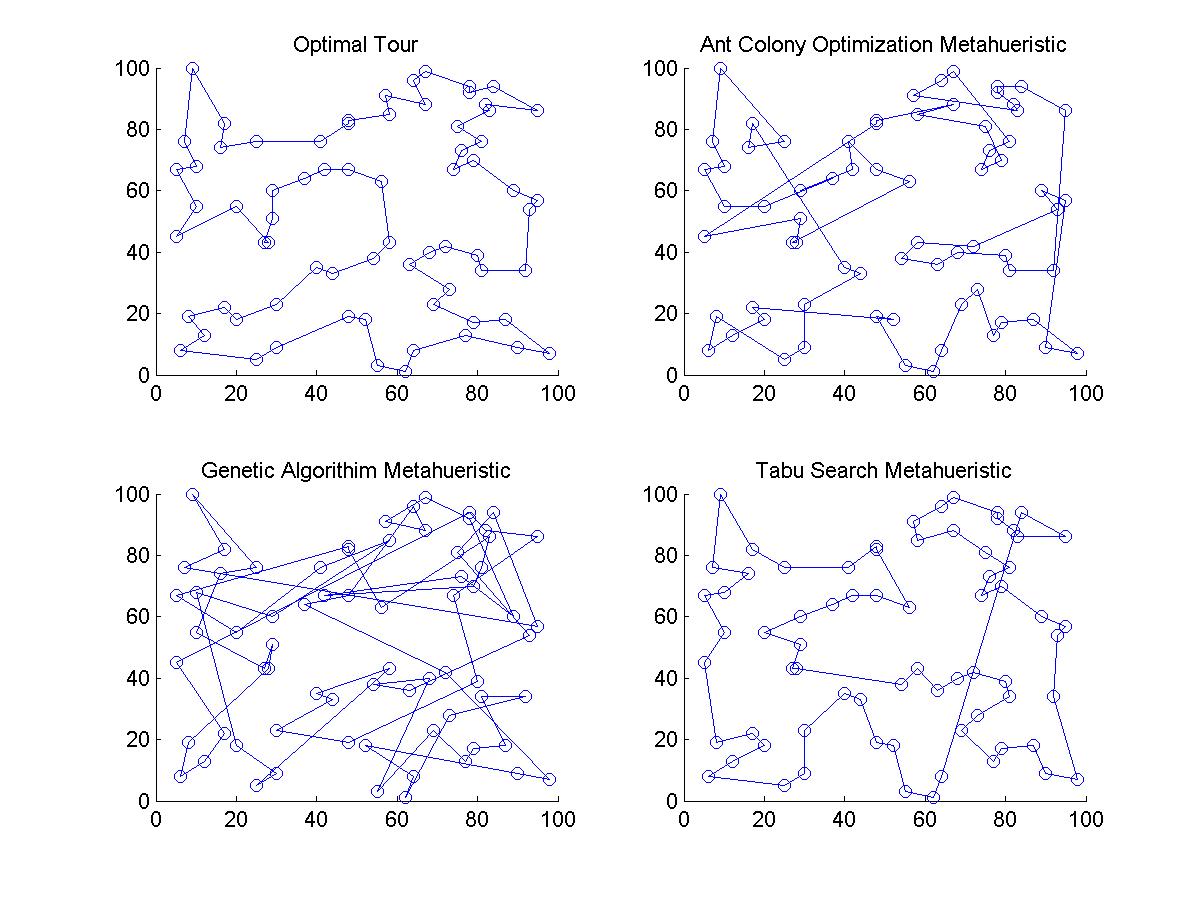


Figure 4, Varied Approach Optimal tours of the St 70 Node TSP Instance

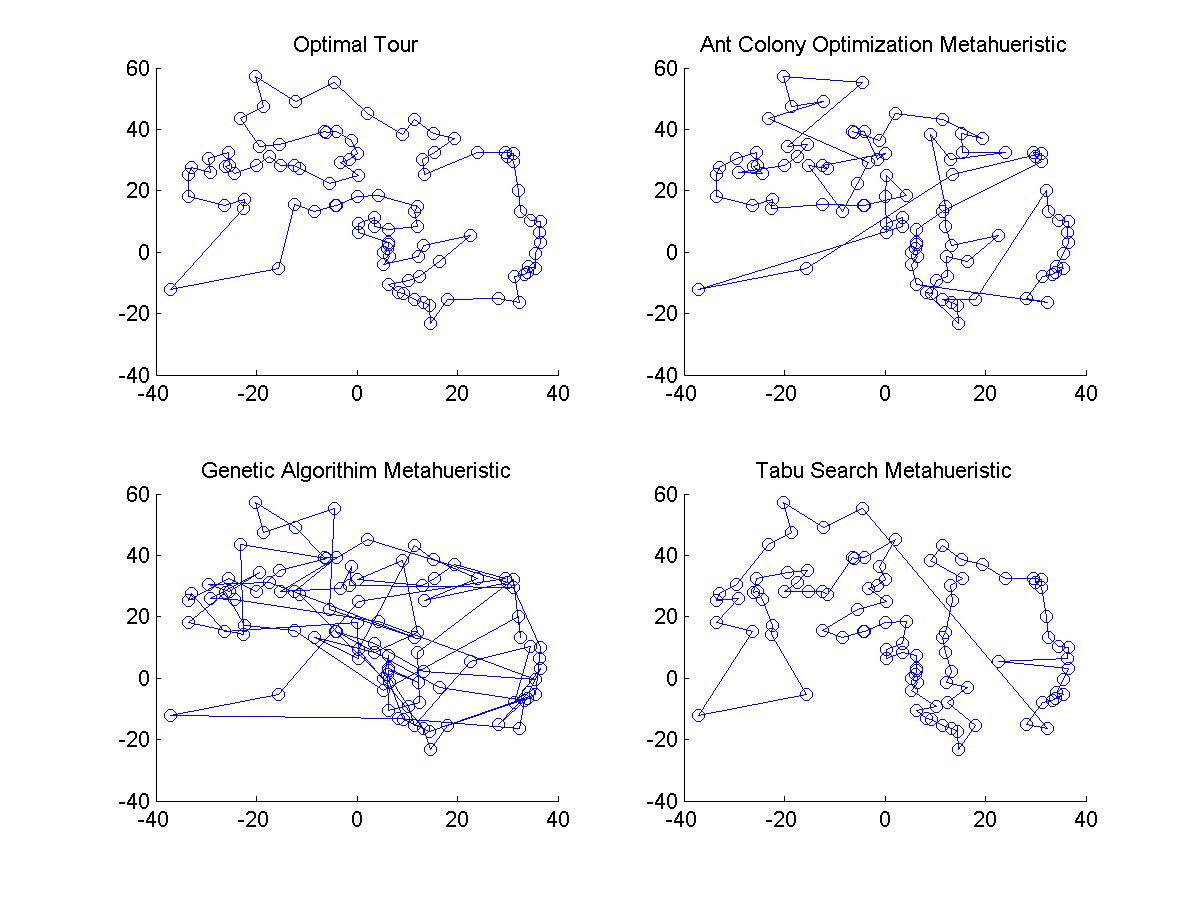


Figure 5, Varied Approach Optimal tours of the Gr 96 Node TSP Instance

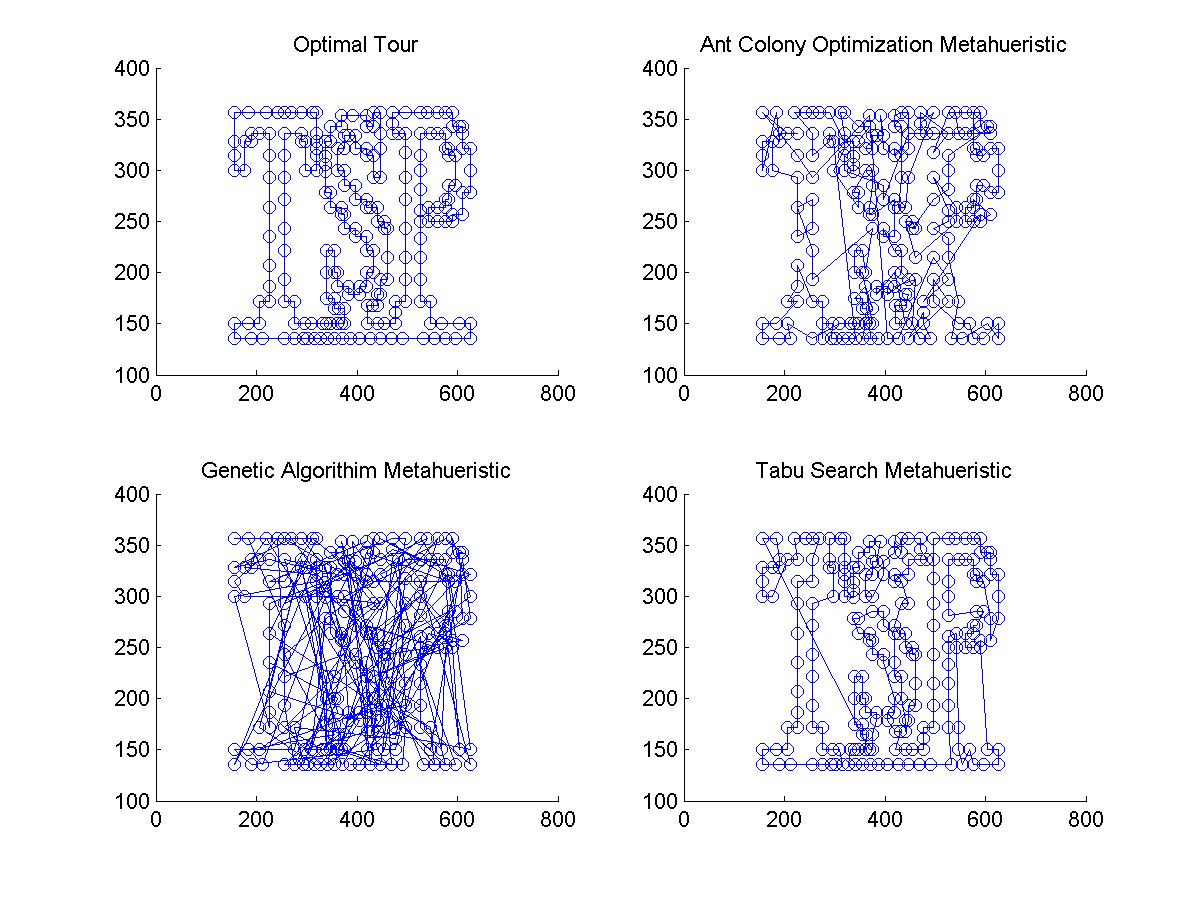


Figure 6, Varied Approach Optimal tours of the TSP 225 Node TSP Instance

Figure7, Heuristic Tour Length Difference from Optimal Solution

Further attention has to be placed on the runtime as well before the Tabu Search can be regarded as a superior method. In Figure 8-10 the runtime in seconds are plotted against the size of the TSP instance. Form these figures it is seen that the methods follow a polynomial computational complexity. This means that as the input is increased that the time to compute the solution grows at powers of the input. The Ant Colony Method seems to be the worst approach. When looking at this growth rate you can see that it takes very long amounts of time as the number of nodes gets large. This is why the Gr 666 instance does not occur. This instance was the initial cutoff of where no larger instances would be run. After the roughly 35,000 seconds, almost ten hours, it took to run the Pa 561 I decided it was not worth it for one more data point. I needed my laptop for other things and these solutions tend to use about 98% of the cpu, and it had an expected runtime of 51,000 seconds or 14 hours. The Genetic Algorithm and the Tabu Search both seemed to take under ten seconds for all instances considered, though the GA approach seems to be growing much faster. This would mean that has the size of the TSP increased past this the runtime will explode. The Tabu Search seems to be almost linear in the region under consideration.

Figure 8, Time in for Solution Calculation of Ant Colony Heuristic

Figure 9, Time in for Solution Calculation of Genetic Algorithm Heuristic

Figure 10, Time in for Solution Calculation of Tabu Search Heuristic

After the comparing the quality of the solutions and there runtimes I think that the Tabu Search would be the appropriate choice for solving future problems given the choice of these three. Other implementations could change this maybe by incorporating better search methods, or sorting algorithms, or even trimming some possible fat out (excess dead code that is still ran). I have a better appreciation for the difficulties involved in solving the TSP as well. I originally was unsure why someone could not solve it exactly ort why so much time was involved. In reality you could require the solution to a 100,000 node instance, which using the Tabu Search would take about 51 years. This answer will not be the best either, only a good one being better that a large portion of other solutions. I think this leaves a lot of future questions for me as well that could expand on this learning. Other approaches could be compared as well as looking at the structure of two equal node instances that have a different structure. I would also be curious to see how the exact optimal solutions are found, especially for larger cases. It is certain though that if instances of imaginable size say a million or even a billion cannot be computed in the age of the universe with current methods the problem will likely be around for another two hundred years.

\*\* All source code is available upon request, will also include a directory structure as well that is needed for proper execution.

**Appendix**

**TSP Instance Description**

ulysses16 - Odyssey of Ulysses

ulysses22 - Odyssey of Ulysses

bays29 - 29 cities in Bavaria, street distances

att48 - 48 capitals of the US

eil51 - 51-city problem

berlin52 - 52 locations in Berlin

st70 - 70-city problem

pr76 - 76-city problem

gr96 - Africa-Subproblem of 666-city TSP

eil101 - 101-city problem

lin105 - 105-city problem (Subproblem of lin318)

gr120 - 120 cities in Germany

ch150 - 150 city Problem

gr202 - Europe-Subproblem of 666-city TSP

tsp225 - A TSP problem

a280 - drilling problem

pa561 - 561-city problem

gr666 - 666 cities around the world

1. TSPLIB, <http://www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/>
2. Matlab User Community, [http://www.mathworks.com](http://www.mathworks.com/)
3. Genetic Algorithm Source Code, Joseph Kirk
4. Tabu Search Source Code, Yonathan Nativ