ROB537: HW2

Ammar Kothari

1 Introduction

Optimization through search is an important part of modern engineering. With the advent of modern computers, people are able to generate and evaluate many solutions. Search has been used widely in fields like engineering and finance, but has also found uses in politics [1]. Search is an integral part of finding solutions to challenging problems today.

In this assignment, I apply three search algorithms to solve a Travelling Salesman Problem (TSP) with 15, 25, or 100 cities that are dispersed somewhat evenly through the map. A fourth scenario with 25 cities is structured in a way to make the solution easily identifiable. Figure 1, 2, 3, 4 show the layout of the cities in each scenario. The three algorithms investigated are Simulated Annealing, Genetic Algorithms, and Monte Carlo Tree Search.

2 Problem Description

TSP is meant to replicate the challenge of a traveller trying to visit many places without revisiting any locations. Accordingly, the solution must contain every city only once, except for the start location. The path must start and end at the same location. The goal is to minimize the total cost of the path. In this case, the cost is the distance travelled.

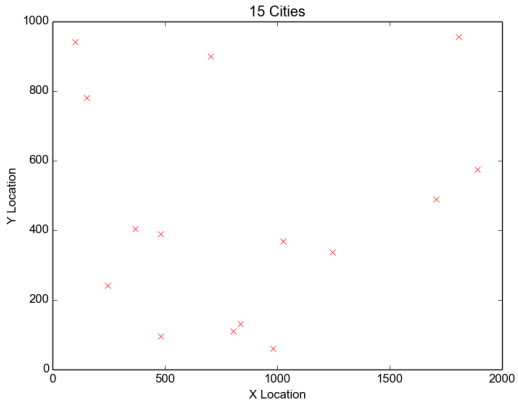

Figure 1: 15 cities plot

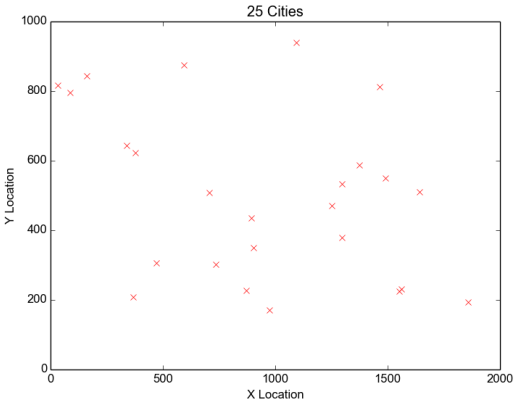

Figure 2: 25 cities plot

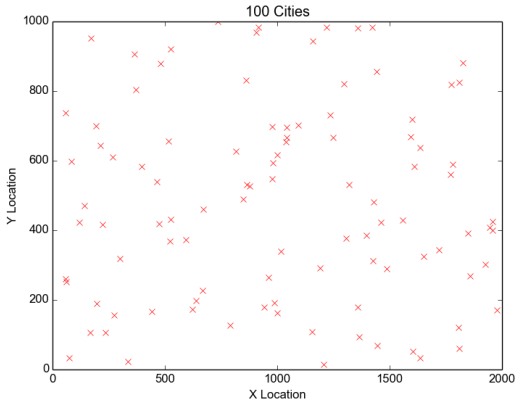

Figure 3: 100 cities plot

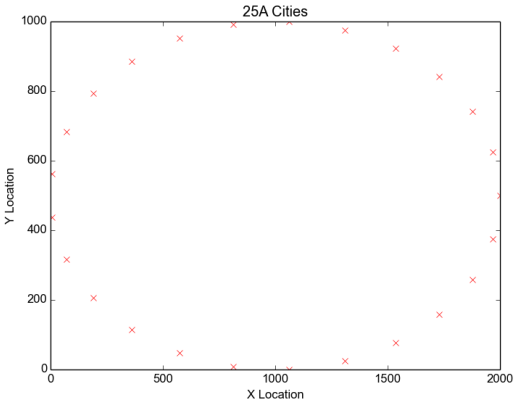

Figure 4: 25 cities plot

3 Algorithm Explanation

3.1 Simulated Annealing

```
s = InitialGuess()
For i < TOTALITERATIONS:
    sNew = NeighborSol(s)
    s = TempSelect(s, sNew, Temp)
    DecreaseTemp(Temp)
FinalSolution = s
```

InitialGuess – Intial solution generated randomly
TOTALITERATIONS – Number of solutions to try
NeighborSol – Generate a succesor state to s by switching the order of two adjacent cities in s
TempSelect – Selects a solution between the two presented based on the temperature. The best solution is probabilistically choosen based on the temperature. The higher the temperature the more likely the next solution is choosen at random.
DecreaseTemp – Decrases the temperature such that it ends at 0 (always choosing the better solution) when the maximum number of iterations is reached.

3.2 Evolutionary Algorithm

```
POP = InitialGuess(POP_TOT)
For i < TOTALITERATIONS:
    POP_SEL = ChooseParents(POP)
    CHILD = PerturbPopulation(POP_SEL)
    POP = SelectPop(CHILD, POP)
Soltuion = Min_Value(Pop)
```

InitialGuess – Initial population of solutions generated randomly
TOTALITERATIONS – Number of generations
ChooseParents – Epsilon-greedy choose best solutions in population based on some amount of noise
PerturbPopulation – Finds neighbor solutions based on choosen parents. The amount of variation to original solution is based on an amount of noise. Variation is implemented as the number of switches between neighboring cities in a solution. Higher noise means more neighboring cities will be switched.
SelectPop – Choose the best solutions from the original population and mutated children. The resulting number of solutions is equal to the starting population size.
Min_Value – From a given population, returns the member that has the best solution value.

In this assignment, Population size was 10. Number of children produced each round was 5. Noise was decreased from 1 to 0 based on current iteration number. For this algorithm, the total number of iterations was choosen as 2,000. This is five times less than the other two approaches in order to have a similar number of total generated solutions.

3.3 Monte Carlo Tree Search

The implementation used in this assignment is not optimized. As a result, it has slow run times but is able to find solutions. For each iteration, the best solution so far has been stored in memory. Alternatively, at the end of runtime, a greedy path can be followed through the tree to a leaf.

Main Algorithm

InitializeTree()

For i < TOTALITERATIONS:

Parent = PickParent(Tree)

Node = PickChild(Parent)

Leaf_Value = DescendTree(Node)

BackPropogateValue(Leaf_Value)

DescendTree

If Node has Children

Node = PickChild(Node)

DescendTree(Node)

Else: PlayOut()

InitializeTree – Initializes tree structure. Nodes are initialized with best value to promote exploration.
TOTALITERATIONS – Number of solutions to try
PickParent – Probabilistically chooses the first node of the tree based on value.
PickChild – Probabilitically choose a child from a node. If the node does not exist, create it.
DescendTree – Subfunction that continues to descend the tree in a probablisitic manner based on node value
PlayOut – Determines value of a node on its first visit. The remaining cities are choosen at random to create a valid path. Returns the value of the full path.
BackPropogateValue – Value of the current node is pushed to the parent. The parent adds the new value to its current value based on the total number of visits. For example, if a parent has been visited once before, then $(Old\_Value * Number\_Of\_Visits + New\_Child\_Value)/(Number\_Of\_Visits + 1)$. This is effectively an average value of the node based on all explored solutions. A node with a good value is perceived as more likely to lead to a good solution than a node with a worse value.

4 15 Cities

4.1 Results

For 15 cities, the algorithms all perform decently well. The evolutionary algorithm was able to find the best solution. SA and MCTS are both able to find near optimal algorithms. EA takes significantly longer than the other two algorithms. Additionally, SA has the largest variance of each iteration of all the methods between runs while EA and MCTS have similar amounts of variance in each iteration during the search process.

One reason for EA's better performance is that it holds many more solutions in memory at a time and can compare them against each other. The other methods hold only one or two solutions at a time and make comparisons based on those two only. In this implementation of EA, the best solution can never be removed. If an optimal solution is discovered, it will not be discarded which is true for MCTS, but not SA. In all cases, EA decreases in distance the quickest. This may be due to having acces to many solutions that it can quickly hone in on a promising solution. Although, this may be problematic later on if this is a local minimum.

SA may struggle to get to the optimal solution if it is currently at a state that is near the optimal solution. If more than a single switch of two consecutive states is requried to achieve a solution, SA is unlikely to get to that solution.

MCTS has to hold a tree structure which can get quite large in memory. Each iteration can add a new node to the tree and requires several calculations to update the tree with every iteration.

Algorithm	Total Solutions Generated	Average Min Distance	St Dev	Average Run Time (s)	St Dev
Simulated Annealing	10,000	7,685	994	4.72e-6	3.8e-6
Evolutionary Algorithm	10,000	5,861	359	4.39	0.17
Monte Carlo Tree Search	10,000	5,916	289	2.19	0.08

Table 1: Comparison of Solution Quality and Run Time for Each Method with 15 Cities

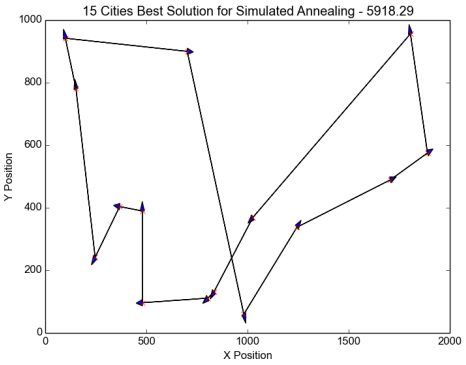

Figure 5: Best Solution for 15 Cities with Simulated Annealing

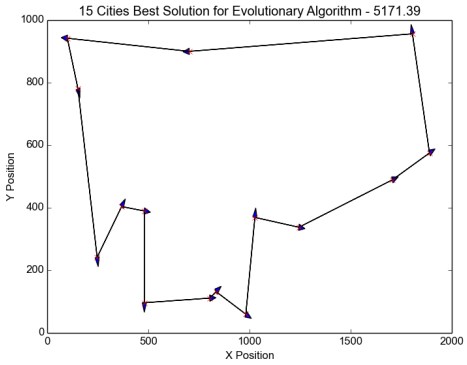

Figure 6: Best Solution for 15 Cities with Evolutionary Algorithm

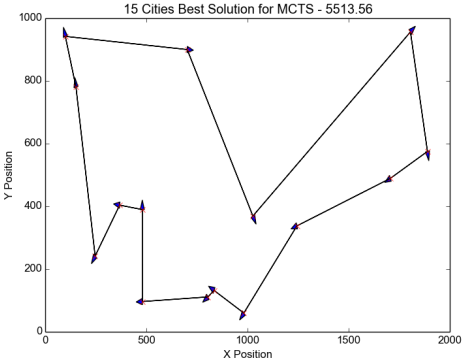

Figure 7: Best Solution for 15 Cities with MCTS

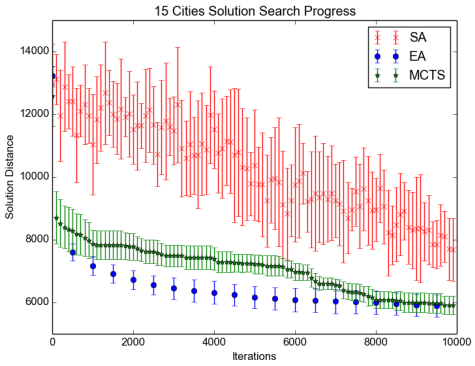

Figure 8: Solution Progression for 15 Cities

5 25 Cities

5.1 Results

For 25 cities, the solution quality decreases. All of the methods are not that close to the optimal solution. EA again performs the best.

Algorithm	Total Solutions Generated	Average Min Distance	St Dev	Average Run Time (s)	St Dev
Simulated Annealing	10,000	10,598	1,567	3.84e-6	6.07e-7
Evolutionary Algorithm	10,000	8,035	581	10.05	0.08
Monte Carlo Tree Search	10,000	8,693	529	3.76	0.19

Table 2: Comparison of Solution Quality and Run Time for Each Method with 25 Cities

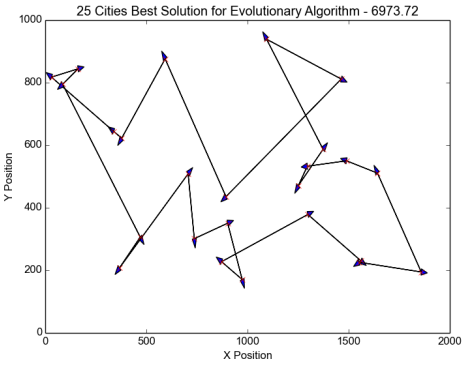

Figure 9: Best Solution for 25 Cities with Evolutionary Algorithm

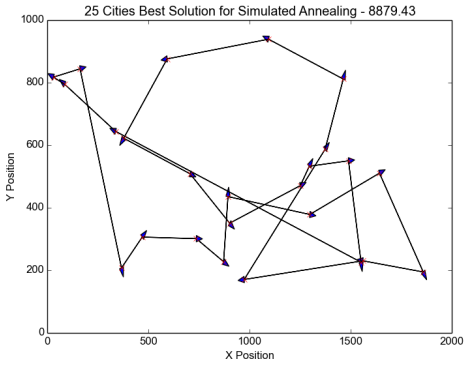

Figure 10: Best Solution for 25 Cities with Simulated Annealing

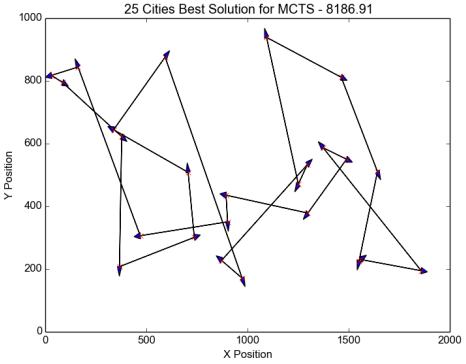

Figure 11: Best Solution for 25 Cities with MCTS

Figure 12: Solution Progression for 25 Cities

6 25A Cities

6.1 Results

For 25 cities in a ring, EA performs best. All of the methods struggle with overcoming paths that go vertically across the circle. Any single switch does not lead to a better solution. Instead, multiple swaps are required to find a better solution. Only MCTS might be able to overcome this; looking at the solution, it seems that it is approaching a better solution, but did not have enough iterations.

The distribution of the cities affects the solution quality. For the ring, switching two consecutive cities is all that is needed for a better solution. This does not apply when two consecutive cities are directly across the the Y direction of the circle. The ovalness of the distrution means directly adjacent cities are not closer. This is a local minimum which is hard to escape. A similar problem can be observed in the other 25 city distribution. However, here the lack of structure in the solution makes many of the switches not as productive. Instead, multiple consecutive city switches are required to find a better solution. This makes the problem more challenging.

Algorithm	Total Solutions Generated	Average Min Distance	St Dev	Average Run Time (s)	St Dev
Simulated Annealing	10,000	13,796	1,742	3.70e-6	8.94e-7
Evolutionary Algorithm	10,000	12,222	1,337	9.89	0.03
Monte Carlo Tree Search	10,000	12,047	937	3.70	0.15

Table 3: Comparison of Solution Quality and Run Time for Each Method with 25 Cities

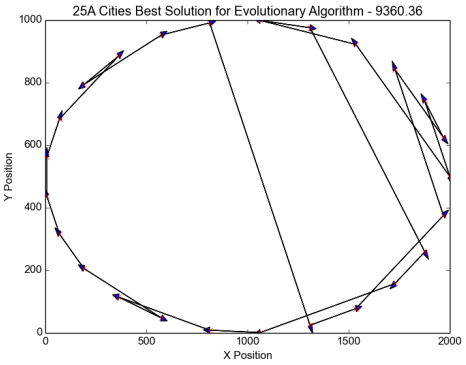

Figure 13: Best Solution for 25A Cities with Evolutionary Algorithm

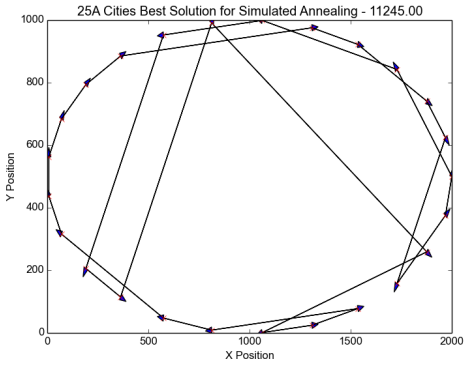

Figure 14: Best Solution for 25A Cities with Simulated Annealing

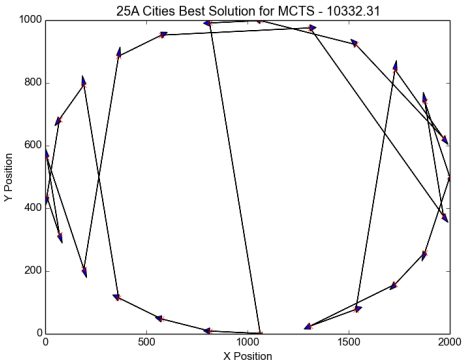

Figure 15: Best Solution for 25A Cities with MCTS

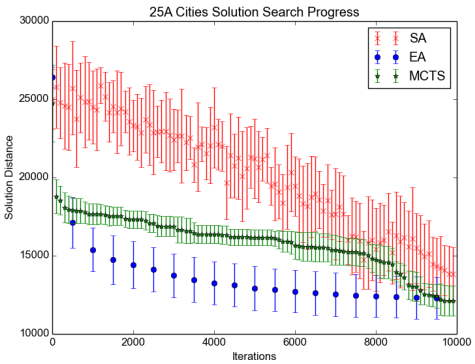

Figure 16: Solution Progression for 25A Cities

7 100 Cities

7.1 Results

For 100 cities, the best solution is SA. EA performs similarly, but has a larger standard deviation. SA may be better at exploring the state space. More variety in solution allows it to continue to hone in on a good solution. The other two methods appear to get stuck in to a local minimum and then have trouble making large improvements from that solution. The large exploratin also explains the steep decreasing slope for SA.

Algorithm	Total Solutions Generated	Average Min Distance	St Dev	Average Run Time (s)	St Dev
Simulated Annealing	10,000	51,846	1993	4.79e-6	4.32e-7
Evolutionary Algorithm	10,000	57,985	1886	127.00	0.47
Monte Carlo Tree Search	10,000	63,638	1865	18.70	0.59

Table 4: Comparison of Solution Quality and Run Time for Each Method with 100 Cities

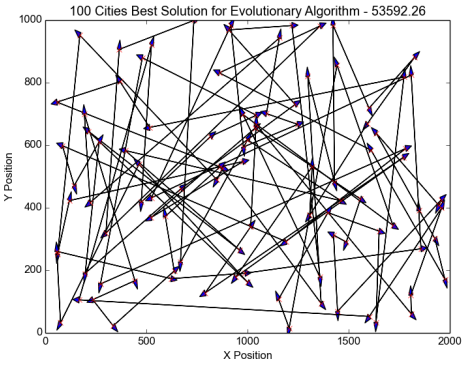

Figure 17: Best Solution for 100 Cities with Evolutionary Algorithm

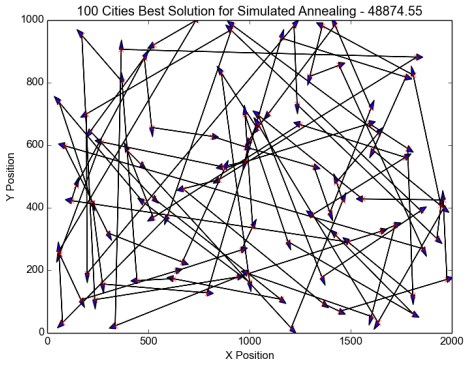

Figure 18: Best Solution for 100 Cities with Simulated Annealing

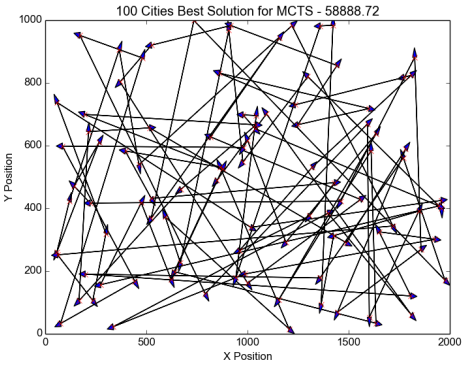

Figure 19: Best Solution for 100 Cities with MCTS

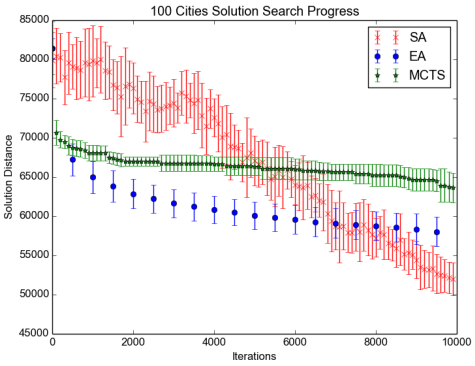

Figure 20: Solution Progression for 100 Cities

8 Solution Size

As the number of cities to traverse increases, the possible solutions increases exponentially. The number of the solutions is calculated as $N!$ where N is the number of cities in the tour. For all the experiments run in this assignment, 10,000 solutions were generated before stopping the search. A generated solution could have been a duplicate of a solution already produced. As a result, the number of solutions searched over cannot be determined for sure. However, as an upper bound, the uniqueness of solutions will be assumed.

Even with only 15 cities, less than 1 millionth of a percent of the space searched is sufficient to find a near optimal solution. SA can introduce a significant amount of randomness into a solution early on. In my implementation of EA, the amount of variability is decreasing throughout the search based on the number of iterations. However, this noise schedule may not be appropriate. For larger spaces, large noise should remain for longer. The reduction in noise is causing search to find a local minimum instead of exploring sufficiently. MCTS does not seem very well suited for this problem. The most common use of MCTS is in games that have very large state spaces like Go. An intelligent playout can help direct the search. In this example, the playout is just randomly selecting the remaining states. The benefit of MCTS is that it can combined with another search algorithm to help with playout. If random playout was instead replaced with SA playout, the results would increase. Having a good playout algorithm is one feature that allowed google to create AlphaGo.

Number of Cities	Solutions	Percent Searched
15	1.31e12	7.65e-7%
25	1.55e25	6.45e-20%
100	9.33e157	1.07e-152%

Table 5: Comparison of Amount of Space Searched

References

[1] Nina Totenberg. This supreme court case could radically reshape politics, Oct 2017.