Travelling Salesman Problem Project 1 Report
Name-Mohit G Kaduskar PSID-154418

Simple Approach

TSP being a NP-Hard problem cannot be solved by Dynamic Programming as there is exponential growth in terms of complexity as the number of cities go on increasing. Initially I tried Dynamic programming which more specifically relies on finding all possible permutations paths possible with number of cities.

As the algorithm (Held Karps) was taking exponential time and not giving out output in less than 20 minutes (as required) for more than 15 cities I identified the need and the use of artificial intelligence to solve the problem of TSP.

A star

I have attached the code A* as Simple approach and Simulated annealing as Sophisticated approach to solve the problem of TSP. Apart from these 2 algorithm I also tried Nearest Neighbor and Genetic Algorithm but as the above-mentioned algorithms performed better than these I went ahead with these.

The nearest neighbor gave was faster compared to A* and gave similar results for C1 but apparently couldn't stand well against the cost functions C2 and C3. The main difference between A* and Nearest Neighbor is the cost function. Nearest neighbor considers just the closest neighbor and not the path to be traversed to reaching again the starting point. A* on the contrary not only considers the closest neighbor but also makes use of its heuristic function to determine which node to explore next. The heuristic function which I am using is explained below and is the best admissible heuristic I could find as it never over estimates the final cost of the tour.

f(n)=g(n)+h(n)

g(n) – Cost to reach the closest neighbor;
h(n) –distance to the nearest unvisited city from the current city + estimated distance to travel all the unvisited cities (MST heuristic used here) + nearest distance from an unvisited city to the start city. Note that this is an admissible heuristic function.

Reference for heuristics: http://www.public.asu.edu/~huanliu/AI04S/project1.htm

Another important observation to compare between Nearest neighbors and A* is that for Nearest Neighbor to give good results I had to take all the nodes as starting point and then find the best solution but in case of A* the heuristic is stable enough not to produce substantial difference in outputs when the starting nodes change.

Pseudo code for A* - Simple Approach

A* makes use of a priority queue to determine which node to explore next. The priority here is determined on the basis f(n) described above. Moreover, it also makes a point to save only one instance of each node on the basis of f(n) i.e. it keeps Node with city x of minimum h(n) and eliminates others. To keep a track of such Nodes, integer array pathCost is used and the indexes are used as city Numbers

Each state is stored in structure Node having following elements:

- cityNo - ith city
- explored [] - Saves all possible states which can be further explored
- parent – This is an object of type Node which saves the parent of current node. Null in case of starting Node.

Flow

Step 1-Create Node startNode with cityNo as startCity with all cities as unexplored and parent as null

Step 2- Keep the cost f(n) as 0 as this is the first Node to be poped by default.

Step 3- pathCost int array explained above initialize with integer MAX value.

Step 4- Insert startNode in Priority queue.

While(Priority queue is not empty perform from steps 4-11 again and again)

> Step 4- currentNode= poll priority queue
>
> Step 5- pathCost[currentNode.cityNo]=Integer.MAX_VALUE;
>
> Step 6- currentNode.explored[currentNode.cityNo]=true; Mark the current city as explored in the node instance
>
> Step 7- Check if goal state reached i.e. whether all the states are visited(using explored array) for that particular Node print the path and total cost and break.
>
> Step 8- foreach Unvisited Neighbor of current Node do following steps
>
> > Step 9- Calculate fn explained above f(n)=g(n)+h(n)
> >
> > Step 9.1- Here gn is the sum of cost to reach the current node and cost between current node and current neighbor node.
> >
> > Step 9.2- Here hn is divided in 3 parts
> >
> > 1st part: distance to the nearest unvisited city from the current neigbour city.
> >
> > 2nd part: estimated distance to travel all the unvisited cities note that the cities neighbor city is considered to be visited. We calculate this by using MST cost.
> >
> > 3rd part: nearest distance from an unvisited city to the start city.
> >
> > Step 10- Once we calculate fn we check whether the current Neigbour node is part of priority queue incase not we add it
> >
> > Step 11-Incase it is already in priority Queue we add replace it on the basis of old f(n) which we store in array pathCost initialized above. If the new fn is less than we replace else not

Step 12 : Print the solution(Path and Cost).

A* Results

Number Of Cities\Cost Function	C1	C2	C3
10	Cost- 426 MEB- 30 Time Taken- 0 min 0 sec	Cost-56 MEB-55 Time Taken-0 min 0 sec	Cost-866 MEB-76 Time Taken-0 min 0 sec
20	Cost- 446 MEB-- 285 Time Taken- 0 min 0 sec	Cost-104 MEB-259 Time Taken-0 min 0 sec	Cost-7768 MEB-617 Time Taken-0 min 0 sec
30	Cost-466 MEB- 678 Time Taken-0 min 0 sec	Cost-124 MEB-513 Time Taken-0 min 0 sec	Cost-29642 MEB-1942 Time Taken-29642
40	Cost-486 MEB-1233 Time Taken-0 min 0 sec	Cost-157 MEB-1009 Time Taken-0 min 0 sec	Cost-70682 MEB-5003 Time Taken-0 min 1 sec
50	Cost-506 MEB-3263 Time Taken-0 min 4 sec	Cost-230 MEB-5045 Time Taken-0 min 3 sec	Cost-142952 MEB-8370 Time Taken-0 min 6 sec
60	Cost-526 MEB- 7339 Time Taken- 0 min 15 sec	Cost-222 MEB-5284 Time Taken-0 min 9 sec	Cost-235146 MEB-13661 Time Taken-0 min 17 sec
70	Cost-546 MEB-13422 Time Taken-0 min 58 sec	Cost-250 MEB-9370 Time Taken- 0 min 29 sec	Cost-364892 MEB-21046 Time Taken-0 min 46 sec
80	Cost-566 MEB-15785 Time Taken-2 min 7 sec	Cost-279 MEB-11954 Time Taken-0 min 56 sec	Cost-581242 MEB-31524 Time Taken-1 min 47 sec
90	Cost-586 MEB-37102 Time Taken-7 min 12 sec	Cost-302 MEB-18182 Time Taken-2 min 32 sec	Cost-788648 MEB-40565 Time Taken-3 min 23 sec

100	Cost-606 MEB-83294 Time Taken-22 min 32 sec	Cost-338 MEB-48223 Time Taken-7 min 24 sec	Cost-1091778 MEB-50764 Time Taken-5 min 56 sec

<u>Sophisticated Approach</u>

The two possible candidate options for the sophisticated approach were Genetic algorithm and Simulated annealing. Genetic algorithm involved three steps after creating initial random population.

- First Selection where I made use of tournament selection to pick one as parent for crossover.
- Second Crossover where I made use of 2 parents to create an offspring making sure that each child has not only unique cities present but also maintains the order of their parents.
- Last but not the least is the mutation step where I was initially mutating the offspring heavily by swapping all the cities but. Hence, I was getting good results as apparently it was not getting stuck at local maxima/minima but as the algorithm follows there are certain constraints when it comes to mutation we can mutate such that the offspring is close to its neighbor so it can be either by swapping 2 random pairs or swapping some neighbors or reversal.

On observing the fact that the Genetic algorithm gets stuck at local optimum and not giving the best results I decided to move to Simulated Annealing as this algorithm tries its best effort to overcome the issue of local optimum if the parameters are tuned well.

The parameters which I tried to tune (population, crossover and mutation probabilty) in case of Genetic algorithm were not giving me good results and that was another reason I decided to go ahead with Simulated annealing.

Simulated Annealing

This algorithm comes from process of annealing of metals where temperature plays an important role. Main idea of this approach is initially when the temperature is high the algorithm lets you move to worse solutions thereby letting you get out of local optimums. As the system temperature variable cools it tries to reach the global maxima. Though this approach doesn’t give a best solution but for NP hard problems it gives a good enough solution.

As mentioned above the most important reason to use this algorithm over genetic algorithm was its capability to try its best to avoid local optimums. Genetic algorithms crossover step also does the same but apparently I couldn’t find the parameters which give me results better than simulated annealing

Following are the tuning parameters in Simulated annealing.

- initial temperature – the starting temperature of the system
- cooling rate parameter – the parameter which reduces the temperature of the system
- minimum temperature –stopping condition

Pseudo code for Simulated Annealing Approach

Step 1- Set temp (initial temperature)
Step 2- Set cooling_parameter (cooling parameter)
Step 3- Set currentPath (Initial set of random shuffled valid tour solution)
Step 4- Create a variable called bestPath (keeps track of best solution generated by system at any point of time)
Step 4- While (temp > minTemperature) do the following steps 4-9 again and again

Step 5- Find the neighbor of the currentPath (neighborPath)
You can do this by swapping some random cities in the tour
Step 6- if(totalCost of currentPath> totalCost of neighborPath)
Step 6.1- Set currentPath=neighborPath
elseIf(. exp((totalCost of currentPath - totalCost of neighborPath) / temp))
Step 6.1- Set currentPath=neighborPath
Note this this calculation allows the system to accept the neighbor solution if it is better at any point of time or accept it if it is worse when the temperature of the system is high.
This logic is made available by the equation :
exp((solutionEnergy - neighbourEnergy) / temperature)
Step 7- Check if this is the best solution on the basis of cost
Step 8- temp *= 1-coolingRate (This will lower the temperature of the system at each iteration)
Step 9- If we reach the MEB then we terminate the program and return the best solution

Step 10 : Print the best solution(i.e Path and Cost).

The main idea for this approach to work exceptionally well is that the parameters of the algorithm should be well tuned. After trying multiple values for the parameter mentioned above following are the values which gave me the best solutions

- initial temperature – 1000
- cooling rate parameter – 0.00003
- minimum temperature – 0.0001

Another important part of implementation is that I have explored the maximum MEB for simulated annealing.

Simulated Annealing Results

For all results as mentioned I have explored maximum MEB- 200000 and the time taken is 0 seconds. Best result for random seed is highlighted.

Number Of Cities	Seed/ Cost Function	C1	C2	C3
10	0	Cost- 426	Cost- 56	Cost- 818
	1	Cost- 426	Cost- 56	Cost- 818
	2	Cost- 426	Cost- 56	Cost- 818
20	0	Cost- 456	Cost- 129	Cost- 7240
	1	Cost- 451	Cost- 125	Cost- 7238
	2	Cost- 451	Cost- 118	Cost- 7240
30	0	Cost- 485	Cost- 177	Cost- 25262
	1	Cost- 479	Cost- 237	Cost- 25266
	2	Cost- 490	Cost- 230	Cost- 25260
40	0	Cost- 513	Cost- 349	Cost- 60886
	1	Cost- 513	Cost- 323	Cost- 60884
	2	Cost- 511	Cost- 347	Cost- 60894
50	0	Cost- 556	Cost- 449	Cost- 120128
	1	Cost- 544	Cost- 417	Cost- 120116
	2	Cost- 536	Cost- 438	Cost- 120184
60	0	Cost- 557	Cost- 518	Cost- 209104
	1	Cost- 585	Cost- 533	Cost- 208954
	2	Cost- 586	Cost- 466	Cost- 209044
70	0	Cost- 620	Cost- 659	Cost- 333500
	1	Cost- 616	Cost- 617	Cost- 333398
	2	Cost- 606	Cost- 568	Cost- 333570
80	0	Cost- 648	Cost- 690	Cost- 499522
	1	Cost- 648	Cost- 726	Cost- 499524
	2	Cost- 629	Cost- 892	Cost- 499576
90	0	Cost- 658	Cost- 805	Cost- 713266
	1	Cost- 656	Cost- 921	Cost- 713338
	2	Cost- 667	Cost- 860	Cost- 713274
100	0	Cost- 708	Cost- 911	Cost- 980562
	1	Cost- 739	Cost- 1007	Cost- 980550
	2	Cost- 716	Cost- 950	Cost- 980966
110	0	Cost- 731	Cost- 1187	Cost- 1307862
	1	Cost- 740	Cost- 1006	Cost- 1307634
	2	Cost- 739	Cost- 1141	Cost- 1307528
120	0	Cost- 770	Cost- 1268	Cost- 1700086
	1	Cost- 817	Cost- 1219	Cost- 1699988

	2	Cost- 796	Cost- 1298	Cost- 1700794

Conclusion

The most important observation is one which we all know it is not at all necessary that we may get the best results with sophisticated approach (Simulated Annealing). There are times where the Simple approach performs better that Sophisticated approach, this completely depends on the cost function and the number of cities.

Below is the table where I compared which algorithm performs well for N=10...120 and what cost it give for each cost function.

Number Of Cities\Cost Function	C1	C2	C3
10	Cost- 426 Approach- Both	Cost- 56 Approach- Both	Cost- 818 Approach- Simulated Annealing
20	Cost- 446 Approach- A*	Cost- 104 Approach- A*	Cost- 7238 Approach- Simulated Annealing
30	Cost- 466 Approach- A*	Cost- 124 Approach- A*	Cost- 25260 Approach- Simulated Annealing
40	Cost- 486 Approach- A*	Cost- 157 Approach- A*	Cost- 60884 Approach- Simulated Annealing
50	Cost- 506 Approach- A*	Cost- 230 Approach- A*	Cost- 120116 Approach- Simulated Annealing
60	Cost- 526 Approach- A*	Cost- 222 Approach- A*	Cost- 208954 Approach- Simulated Annealing
70	Cost- 546 Approach- A*	Cost- 250 Approach- A*	Cost- 333398 Approach- Simulated Annealing
80	Cost- 566 Approach- A*	Cost- 279 Approach- A*	Cost- 499522 Approach- Simulated Annealing
90	Cost- 586 Approach- A*	Cost- 302 Approach- A*	Cost- 713266 Approach- Simulated Annealing

100	Cost- 606 Approach- A*	Cost- 338 Approach- A*	Cost- 980550 Approach- Simulated Annealing
110	Cost- 731 Approach- Simulated Annealing	Cost- 362 Approach- A*	Cost- 1307528 Approach- Simulated Annealing
120	Cost- 770 Approach- Simulated Annealing	Cost- 1219 Approach- Simulated Annealing	Cost- 1699988 Approach- Simulated Annealing

Hence from the above results we can very well observe that the simple results don't give that great results for cost function c3 but does exceptionally well for cost function c1 and c2.

But another issue being that A* cannot be used for larger number of cities as it exponentially increases the time frame hence for some test cases where number of cities are exceeding 100 specifically for cost function c1 and c2 we cannot get a cost.

As we had a bound namely MEB sophisticated algorithm has to stop when it reached MEB. If we remove this bound the algorithm does much better than the current results. When the system cools down but on parallel maximum MEB is explored it may not be able to reach the minima hence we don't get the best value.