CS6380 Artificial Intelligence Assignment 1

Vedant Saboo, CS19B074

October 15, 2021

Abstract

Primarily as a report, for the Assignment 1 in the course CS6380 Artificial Intelligence, focusing on solutions for the Travelling Salesman Problem, with the problem specification also including additional information about whether the data points being euclidean, or general (distance matrix explicitly specified.) Asymmetric data sets are included.
This document includes the briefs of the algorithm used for solving the problem, with a recorded average optimality of 96% over selected test cases. The basic idea is to have a Genetic algorithm setup, with additional enhancements in the form of a good starting deterministic solution (Christofides Algorithm [1]) and occasional use of perturbations in the form of 2-opt edge exchanges and 3-opt edge exchanges. As an addition to accommodate large n case, where genetic algorithm have a difficultly working (iteration complexity is huge, and thus performance of genetic algorithm is less in the given time frame), a Simulated Annealing framework has been incorporated.
Exact implementation and other details is what this document will contain. It will also contain performances on some of the standard TSPLIB instances.

1 Problem Statement

Travelling Salesman Problem: Given a set of cities and distance between every pair of cities, find the shortest possible route that visits every city exactly once and returns to the starting point. With time limit of 300s of execution time.
Shortest in terms of **cost**, where cost of a tour is the total distance taken to travel the tour, in a particularly specified direction, where either euclidean, or explicit distances may be considered, depending on the problem.

2 Algorithm Implementation and Development

The Algorithm is implemented basically as a **genetic algorithm** framework. The basic steps are:

1. **Evaluate** the current population, assign scores as $\frac{MAX\_COST}{COST[i]}$ Here we also detect SATURATION, that is, all the candidates have the same score. In which case we can end the loop, or reset population and start again. Currently total population is $psize = 4 \times n$ where n is dimension of the problem.

2. **Select** Construct a gene pool, based on the scores. Some of the fit solutions will be multiplied. Shuffle this gene pool and randomly select **rsize** number of parents for the crossover. Currently $rsize = 2 \times n$ where n is dimension of the problem.

3. **Crossover** Take the parents and cross them, two at a time, if both parents are not same. Algorithm used for crossovers are partially mapped crossover (PMX), ordinal representation single point cross over, and cycle crossover. Currently, PMX is set as default, while implementaions for ORD-SinglePoint and CYCLE crossovers are available. Ideally, n pairs are crossed, forming $2n$ children.

4. **Mutate** While the children are produced, one out of **four** children are mutated, using inversion mutation algorithm, where any two points are selected randomly and the subtour is reversed. This is similar to two edge exchange.

5. **Retain** All children are added to the population, and the population is trimmed back to psize.

This completed one iteration. This is resumed a number of times, an upper bound set to INT_MAX and another terminating condition that is set, is if the SATURATION flag is set continously for 500 times, then the algorithm terminates. Or if 300s run out, whichever comes first.

For $n > 1500$ case, we switch over to **Simulated annealing.**

1. **Parameters:** We keep temperature varying from INFINITY (C++ INFINITY) and 1.0e-20. Cooldown factor is kept at 0.99. For each temperature, 100 inner iterations are taken.
2. **MoveGen:** We keep a random 2-opt exchange as a random MoveGen.
3. **Eval:** Evaluation function is $eval(diff, temp) = e^{\frac{-diff}{temp}}$ where diff is neighbour fitness - currnode fitness and temp is current temperature.
4. **When to go ahead?** When either the neighbour is better than currnode, or the evaluation is better than any random number of 0 to 1.
5. **Cooldown:** multiple current temperature with cooldown factor alpha.

3 Enhancements

A few enhancements make this genetic algorithm into a Hybrid Genetic Algorithm. These are based on simple observations.

1. **Start well to end up better.** We pre-process before starting the first iteration, using Christofides algorithm [1] to find a tour. The algorithm is a known 3/2 optimal approximation for points satisfying triangular inequality, but even in general, it is good tour to start with. It gives out a good path in general, and it is quick to do it.
   (a) Finds a minimum spanning tree, using Prim 's greedy algorithm.
   (b) Finds a perfect matching
   (c) Finds an appropriate Euler tour, the best one, and makes it into Hamiltonian tour
2. **Climb the ascent to the highest if you see a new one.** Whenever we find a new best solution, we will use iterated 2-opt exchanges and iterated 3-opt exchanges to better the path, getting the best tour of what it could offer. This may be a local optimum, in which case we may get stuck, but in case it is indeed a global optimum, we have found out solution.
   (a) 2-opt : Start with a tour, choose two edges, delete them and add two more, so that the path length is decreased and the tour is valid. Iterate for all possible combinations, and move to the best tour found. Continue, starting with this tour, making it better, until you cannot do it anymore.
   (b) 3-opt : Start with a tour, choose three edges, delete them and add three more, so that the path length is decreased and the tour is valid. Note that here seven recombinations are possible; try them all. Iterate for all possible combinations, and move to the best tour found. Continue, starting with this tour, making it better, until you cannot do it anymore.

4 Computational Results

See Table 1 to refer to the results obtained by running the algorithm over some TSPLIB instances.

These tests have all been made with a timeout of 300s and in identical circumstances. These are the best results found after a lot of runs. While a few runs sometimes ran into local minima without reaching these values, these values reiterated quite a number of times. Thus we can say that these are the best results, while also being very close to the average results.

	Name	Number of points	Type	Cost of solution found	Optimal cost
1	custom5	5	Non Euclidean	19*	19
2	bayg29	29	Non Euclidean	1610*	1610
3	bays29	29	Non Euclidean	2020*	2020
4	att48	48	Non Euclidean	10970	10628
5	berlin52	52	Euclidean	7590	7542
6	gr17	17	Non Euclidean	2085*	2085
7	fri26	26	Euclidean	937*	937
8	dantzig42	42	Euclidean	699*	699
9	dsj1000	1000	Euclidean	20010456	18659688
10	eil101	101	Euclidean	655	629
11	ch130	130	Euclidean	6418	6110
12	ch150	150	Euclidean	6712	6528
13	kroA100	100	Euclidean	22137	21282
14	kroB100	100	Euclidean	22141*	22141
15	kroC100	100	Euclidean	20901	20749
16	kroD100	100	Euclidean	21693	21294
17	kroE100	100	Euclidean	22525	22068
18	brd14051	14051	Euclidean	23587607 (trivial path)	469445
19	uy734	734	Euclidean	84564	79110

Table 1: Results on some of the TSPLIB instances. *Optimal Tour Found

5 Summary and Conclusions

To summarize, we present a solution to the traveling salesman problem, both euclidean and generalised. The algorithm is not exact, but produces good results (mostly within the margin of 5 percent) in most case and upto a limit. It may however be noted that huge point set takes a considerable time penalty, which is why tour found is suboptimal. In a close range, results are good enough. In many case it also finds the optimum, which if otherwise left for brute force and exact algorithms, would have taken a huge amount of time. (super exponential complexity is what this problem fashions normally).

References

[1] Michael T. Goodrich. *"18.1.2 The Christofides Approximation Algorithm"*.