Solving Travelling Salesman Problem(TSP) with Genetic Algorithms and Simulated Annealing.
University "Alexandru Ioan Cuza" of Iași

Habasescu Andrei E2, Zmeu Ion E4

January 10 2023

Abstract

The aim of this study is to analyze and compare the efficiency of Genetic Algorithms(GA)[1] with Simulated Annealing(SA) heuristics in solving Traveling Salesman Problem(TSP). Both algorithms are often used for optimization problems,but in comparison to Simulated Annealing, The Genetic Algorithm outputs desireable results(under 10% error) within a reasonable amount of time therefore being a reliable method of finding solutions to TSP problems of variable sizes, Simulated Annealing being a much faster algorithm optimal for working with small samples of the TSP problem, however as the number of dimensions increase the results are far worse than those of The Genetic Algorithm, both time and result wise.

1 Introduction

This paper analyses the Genetic Algorithm ability to solve Traveling Salesman Problem (TSP) and compares it with heuristic method Simulated Annealing(SA).Comapring the time and results we will conclude that the Genetic Algorithm gives better results for larger instances compared to Simulated Annealing which is a better fit for small input TSP problems.
In this article we will use 10 data sets taken from TSPLIB[2] which are :**berlin52, tsp225, pr439, rl1304, fl1400, u2319, pr2392, pcb3038, fnl4461, rl5934** using the sample size 30 and the coresponding dimension being the number in their names.
We tested and concluded that the most optimal configuration for time and performance for The Genetic Algorithm is population of 100 and 1000 generations, having implimented elitism which makes up 15% of the population in order to ensure the passing of favourable mutations. For the Simulated Annealing algorithm we are using a base temperature of 1000 with the rate of cooling being 0.97.

2 Algorithm Configurations

2.1 Simulated Annealing

Simulated Annealing[3] has a base temperature of 1000 with a rate of cooling of 0.97 and a minimum temperature of 0.00001 which also serves as stoping condition, favouring exploration over time efficiency potentially computing better results.

This algorithm is a metaheuristic method used to approximate the global optimum in a large search space for optimization problems, it uses temperature as margin of difference between potential solutions which, which can prove detremental but also allowing for exploration a larger space. It makes use of operations such as Mutation, Inversion and Swap in order to generate new potential solutions to the TSP problem by applying the said operations on the previous generation and evaluating whether the change is favourable. [4]

2.2 Genetic Algorithm

The Genetic Algorithm uses a population of 100 and 1000 generations, optimal mutation chance being 0.75 and crossover with a chance of 0.8, being this high in order to ensure the generation of new possible solutions in the later part of the execution where we must ensure the preservation of spontanous mutations which are favourable, while also increasing dramatically the probability of such. In order to ensure a single mutation's ability to influence the entire population we have implemented elitism, having saved 15 samples of the best chromosome which we identified with the help of a fitness function, which later would crossover with the other individuals.

This Algorithm starts with a randomly generated population and using a fitness function it creates a selection (in our case the Wheel of Fortune) where it favours the best chromosomes to be chosen, modified and operated on to obtain a potential solution to our instance of the using mutation and crossover operators.

2.2.1 Wheel of Fortune[5]

```
1.   Evaluate P                  for i:=0 to POP_SIZE
2.                                    eval[i] = f( P(i) )
3.   Total fitness               for i:=0 to POP_SIZE
4.                                    T += eval[i]
5.   Individual sel.prob.        for i:=0 to POP_SIZE
6.                                    p[i] = eval[i] / T
7.   Accumulated sel.prob.                    q[0] = 0
8.                                for i:=0 to POP_SIZE
9.                                    q[i+1] = q[i] + p[i]
10.   Selection                   for i:=0 to POP_SIZE
11.                                   uniform random r in (0,1]
12.                           select for the next generation the individual j
13.                                   for which q[j] ¡ r ¡= q[j+1]
```

3 Results

Problem			G.A.				S.A.			
Name.	No.	Optimum	Best	Time	Avg	Worst	Best	Time	Avg	Worst
berlin52	52	7542	7544.36	3s	7544.36	7544.36	7544.35	3s	7544.36	7716.6
tsp225	225	3916	3990.87	15s	3997.72	3997.72	4018.34	6s	4031.596667	4043.2
pr439	439	107217	110497	41s	111182.36	111730	114962	19s	116994.6667	120099
rl1304	1304	252948	259685	343s	263834.2	266286	442129	155s	444098	447966
fl1400	1400	20127	21711.5	291s	22019	22398.5	30601.8	232s	31800.43333	34197.7
d1655	1655	62128	66838.8	293s	67627.8	68700.7	115395	168s	188374.7544	118283
u2319	2319	234256	244831	759s	245512.6667	246004	415333	342s	417522	419292
pr2392	2392	378032	396888	576s	399407.66	402273	906416	349s	918359.6667	928465
pcb3038	3038	137694	146750	1003s	147262.6	147683	382504	752s	385000.3333	387739
rl5934	5934	556045	595947	3480s	597871.93	600544	1105486.4	2804s	1204786.5	1251582

4 Conclusion

The Genetic Algorithm outputs desireable results(under 10% error) within a reasonable amount of time therefore being a reliable method of finding solutions to TSP problems of variable sizes, Simulated Annealing is a much faster algorithm optimal for working with small samples of the TSP problem, however as the number of dimensions increase the results are far worse than those of The Genetic Algorithm, both time and result wise.

The Genetic Algorithm is doing much better for larger classes of problems. Initially it offers worse results than Simulated Annealing, but more balanced. As the number of cities increases, the Genetic Algorithm begins to gain both in terms of time and solution quality.

5 References

[1]https://en.wikipedia.org/wiki/Genetic_algorithm
[2]http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
[3]https://www.youtube.com/watch?v=XNMGq5Jjs5w:Simulated Annealing with Python by Johnnyboycurtis
[4]https://en.wikipedia.org/wiki/Simulated_annealing:
[5]https://profs.info.uaic.ro/~eugennc/teaching/ga/:Explanation for GA by Eugen Croitoru