Optimization methods - Starting project report

Introduction

Aim of this project

The aim of this project was to get familiar with Evolutionary Algorithms, particularly with the Genetic Algorithm.

Problem

We were given a TTP - Traveling Thief Problem, which is a combination of two NP-hard problems, Traveling Salesman Problem(TSP) and Knapsack Problem(KP).
The objective of this problem is to visit all cities (similar to TSP problem), but at the same time “collect” items from the cities, in order to maximize the profit.
Each item has it’s profit(it’s worth) and it’s weight.
The thief is able to pick only 1 item from each city, and his bag, where he stores his items, has limited capacity.

Used algorithms

As instructed, I will use 4 different algorithms in order to optimize given problem, these algorithms will be:

- Random Search
- Greedy Algorithm
- Simulated Annealing
- Genetic Algorithm

Implementation of Genetic Algorithm

I've implemented the algorithm with the following parameters:

- popSize -> the size of the population
- numberOfIteration -> number of iterations, after which the algorithm will stop
- selectionType -> the type of the selection of population, either tournament or roulette
- tournamentSize -> the size of the tournament
- roulleteAmountChosen -> the amount of pops to be chosen in roulette selection, this is picking with replacement
- crossoverChance -> the chance of the crossover between the 2 parents
- mutationChance -> the chance for a single individual to mutate

The starting population is initialized randomly, meaning that the individual is created by running algorithm that traverses the cities and picks up the items randomly.

The mutation is simply swapping the places between 2 cities and 2 items in the individual's solution

For the crossover I implemented the order crossover, that creates 2 children from 2 parents.

Roulette selection has been implemented with replacement, but the tournament selection doesn't allow the same individual to win more than 1 time.

Picking M best individuals happens before the selection process and picked individuals are not involved in the selection process.

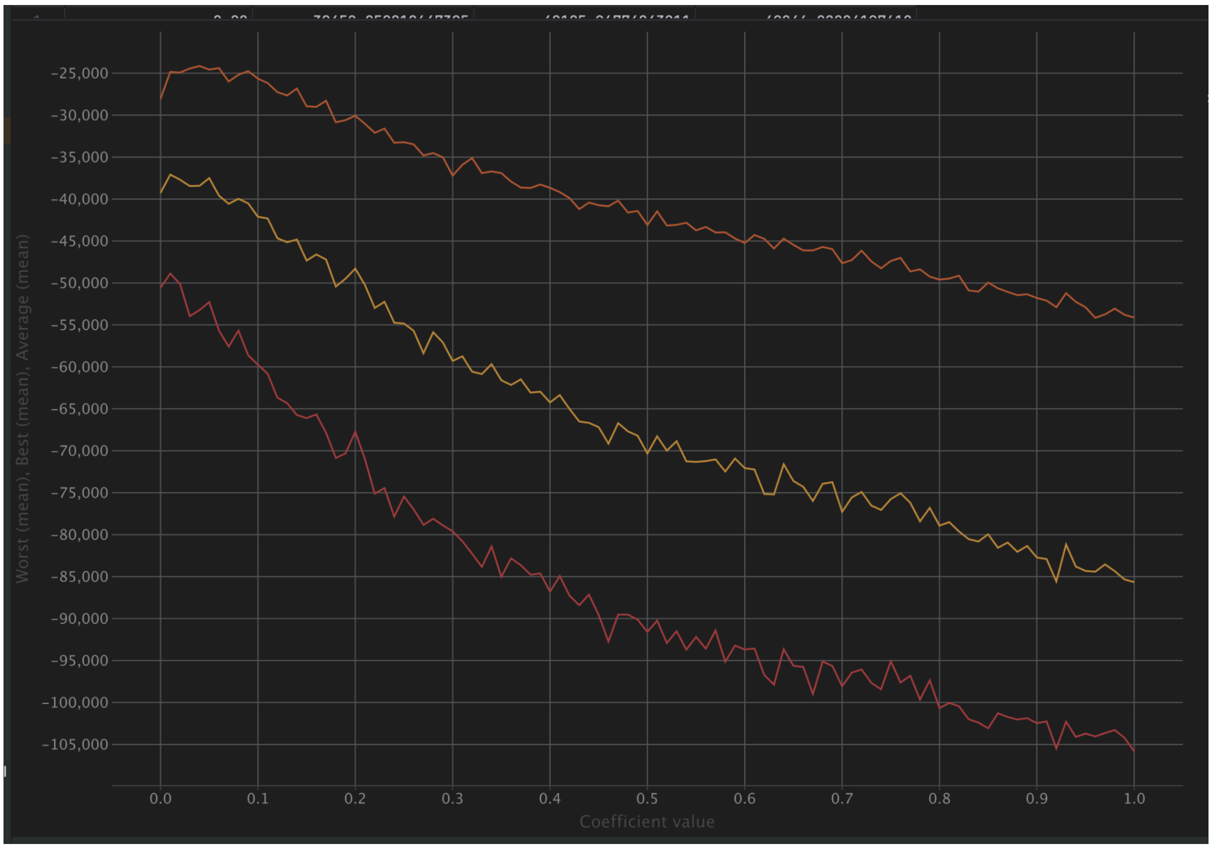
Throughout the iterations of the algorithm I'm keeping the population the same size.

Experiment

Choosing coefficients

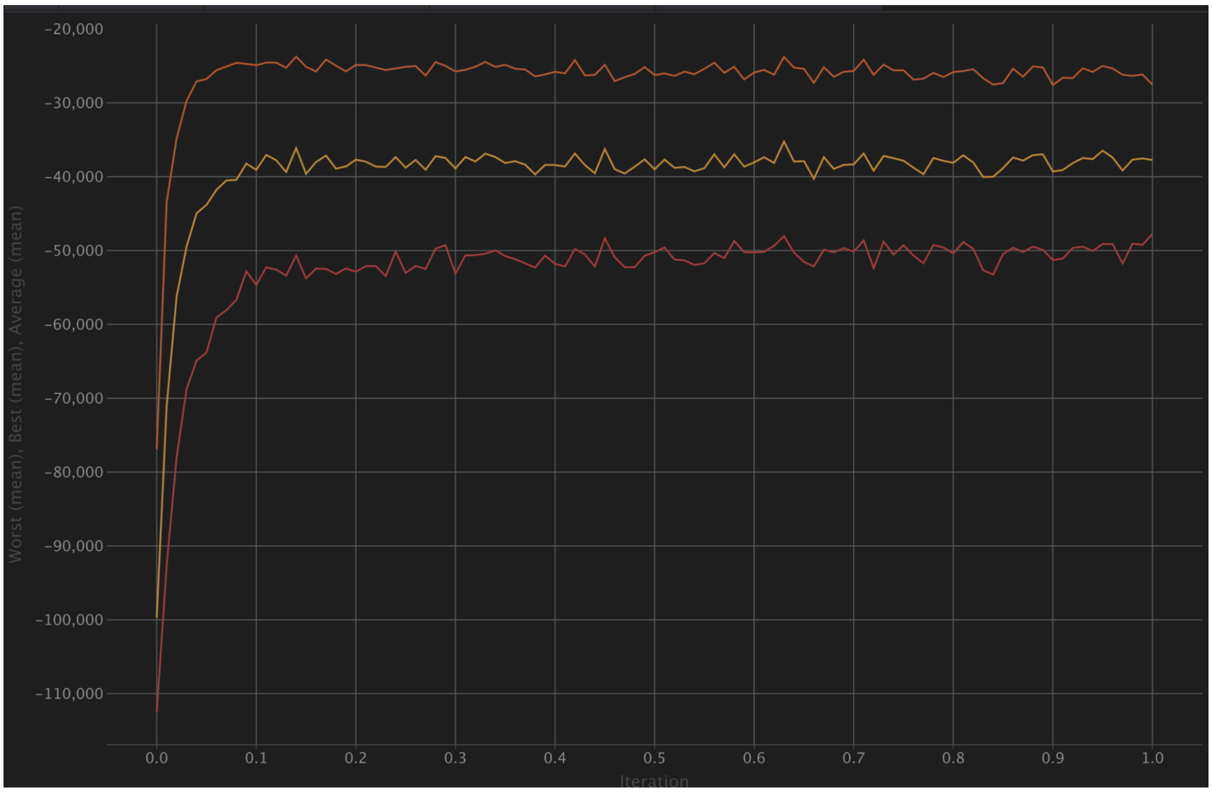
In order to test Genetic Algorithm I've decided to test on one of the test files, what value of crossover and mutation rate results in the best fitness of the algorithm.

Mutation chance plot



Here when it comes to the fitness of the best solution "orange line", the best mutation chance is %4 (0.04 in the graph). Increasing mutation further results in more "chaotic" change in the genotype of individuals, which results in a gradual worsening of resulting fitness.

Crossover chance plot



For the crossover, the fitness of the population quickly increases until the crossover chance is equal 10% (This run assumes that the mutation chance is equal to 4%). The crossover chance that gives the best result is equal to 63% although we can see that this crossover chance doesn't vary much from the crossover chance between 50% and 80%)

Chosen data

In order to test the Genetic Algorithm, I'm going to use 7 different test files from data which we were given, those files are:

1. a280_n279_bounded-strongly-corr_01.ttp
2. a280_n279_uncorr_01.ttp
3. a280_n1395_uncorr_05.ttp
4. berlin52_n51_bounded-strongly-corr_09.ttp
5. berlin52_n153_uncorr-similar-weights_05.ttp
6. berlin52_n255_uncorr_06.ttp
7. berlin52_n510_uncorr_08.ttp

I'm going to run for each test case, each algorithm 10 times and measure it's best, worst and average result and also the standard deviation.
For each algorithm I'm going to assume a population of 200, 300 iterations, crossover chance of 64% and mutation chance of 4%. Tournament size will be 10 and I will always keep 1 best individual from the population.

Results

Test case	**Random Search**				**Greedy Algorithm**				**EA**				**SA**			
	min	avg	max	std	min	avg	max	std	min	avg	max	std	min	avg	max	std
1	-1614598	-1575803	-1540695	24981	-144463	-132921	-124470	5501	**-78725**	**-75844**	**-73453**	**1659**	-1254180	-1143239	-1072769	50759
2	-1715591	-1687737	-1648542	23669	-169409	-160589	-152924	5276	**- 82842**	**-73122**	**-66197**	**4352**	-1249837	-1160716	-1098241	55541
3	-3062162	-2975761	-2916802	43896	**-295697**	**-280143**	**-265975**	**9509**	-867898	-772909	-721164	60434	-2191456	-2069230	-1949948	75150
4	-118617	-104637	-84409	10705	-65765	-54286	-33408	4317	**-9734**	**-5803**	**-2472**	**2005**	-189549	-157002	-110150	25297
5	-91132	-83480	-86935	2156	-9985	-6750	-2082	3467	**-5554**	**-2392**	**1339**	**1925**	-93678	-76862	-66902	9732
6	-128965	-120621	-115193	3921	**-13843**	**-11666**	**-8324**	**2357**	-21933	-16005	- 12577	2468	-124508	-103010	-84428	13078
7	-291294	-280853	-274148	5540	-63288	-59207	-57898	1613	**-64115**	**-56509**	**-5035**	**3591**	-280108	-230428	-198849	24552

In most of the cases the Genetic Algorithm produced the best solution for a given problem, but in 2 cases the greedy approach was better, in the case number 3 the greedy approach completely outclassed the Genetic Algorithm and in the case 6 it was slightly better. The reason for that may be that this data consisted of cities that had a larger number of items available in them compared to the other cities.
Because the initial population for my implementation of GA is initialized randomly, this increased the set of possible solutions which resulted in a worse initial population, thus worsening the final fitness. I will discuss the possible improvement later in the report.

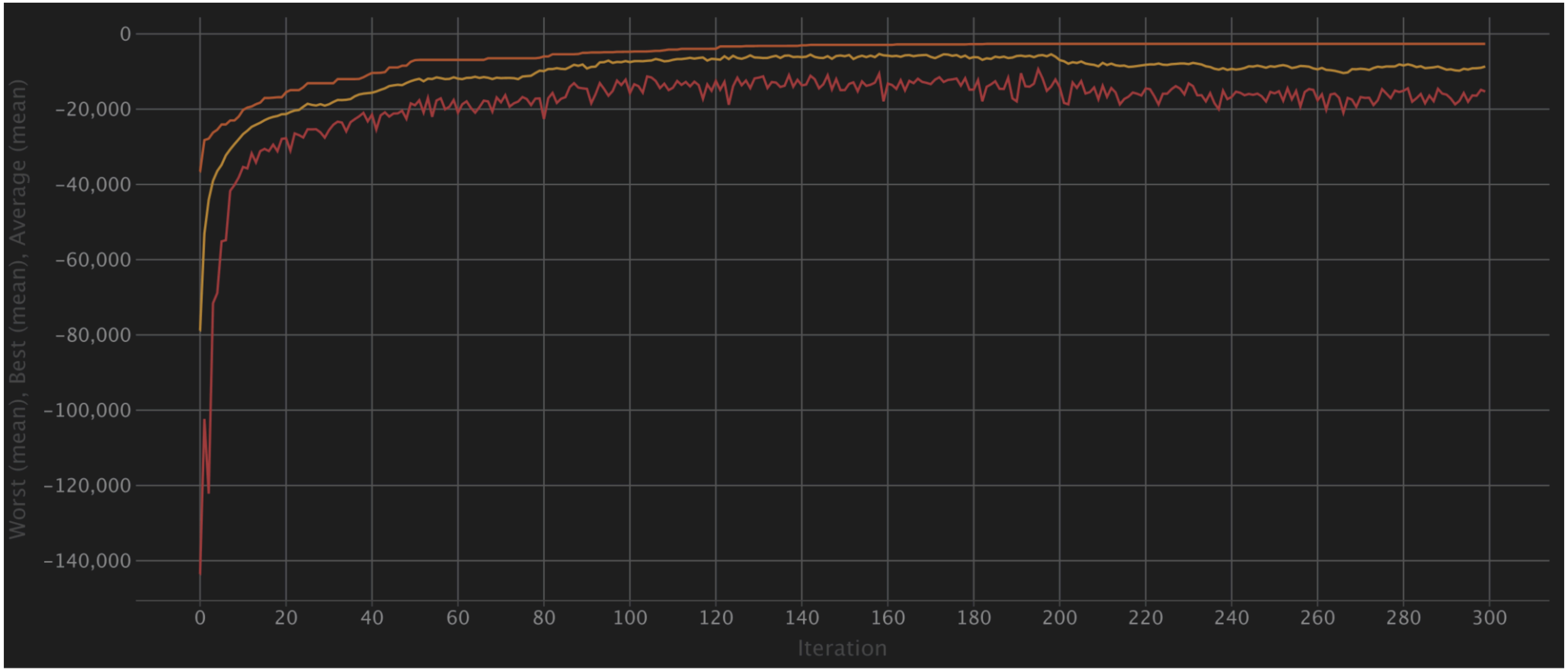
The influence of parameters

During the experiment, I’ve found(by method of trial and error) that keeping the tournament size at around 1-5% of the size of the population results, also I keep the M best individuals so they make around 15-20% of the population that results from the selection process.

When it comes to crossover and mutation I kept mutation chance at 5% because from what I’ve concluded that too big mutation chance doesn’t affect the solution positively, making the changes too chaotic, which results in slower improvement of population over the iterations

Data sets like a280_n1395_uncorr_05.ttp turned out to be difficult case for GA. Large number of possible solutions turned out to be too much for my GA, compared to the Greedy Algorithm. My idea regarding such case, where we have large set of possible solutions, would be to use greedy algorithm in order to increase the fitness of the base population, for example 90% of the population would be initialized randomly and 10% using the greedy approach.

Example run of Genetic Algorithm



The data here comes from the berlin52_n51_uncorr-similar-weights_07.ttp file.
The parameters of GA here were following

Population_size: 300, iterations: 300, crossover_chance: 0.7, mutation_chance: 0.05, tournament_size = 10, keep_m_best_pop = 5

Summary

In this experiment I've shown the basic implementation of Genetic Algorithm and compared it with other popular metaheuristics. The attunement of parameters heavily depends on the data that we are working on. Not only that but in some cases, when the number of possible solutions is huge, we should also think about the way that we initialize the starting population. I really enjoyed tinkering with this algorithm and also creating it from scratch, I know that my investigation of the parameters values may've been better but I think that during this experiment I've vastly deepened my understanding of basics of genetic algorithms and their behavior.