Introduction to Artificial Intelligence and Machine Learning

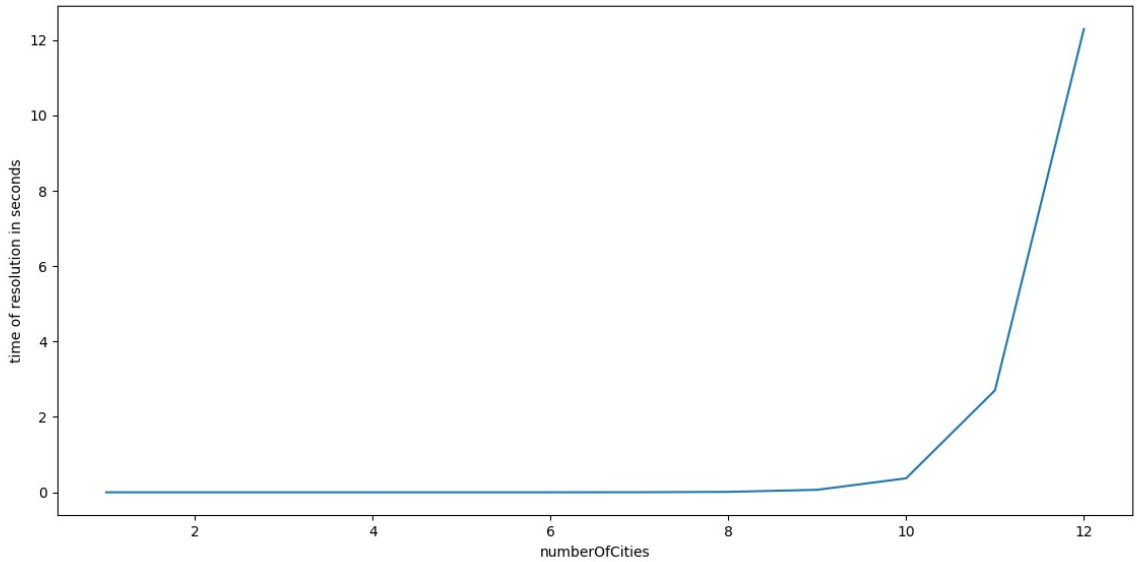
First Mandatory assignement report

Thibaut ELOY

To run my code go at the end of it and run the function « answer to questions » which takes the number of the question as a parameter. « Q1 » exhaustive search, « Q2 » hillclimbing, « Q3 » genetic algorithm, « Q4 » one run of the two hybrid algorithms, « Q4b » statistics on the two hybrid algorithms. « Q4c » average fitness plot through generations. Calls of the function are commented at line 490

Exhaustive Search :

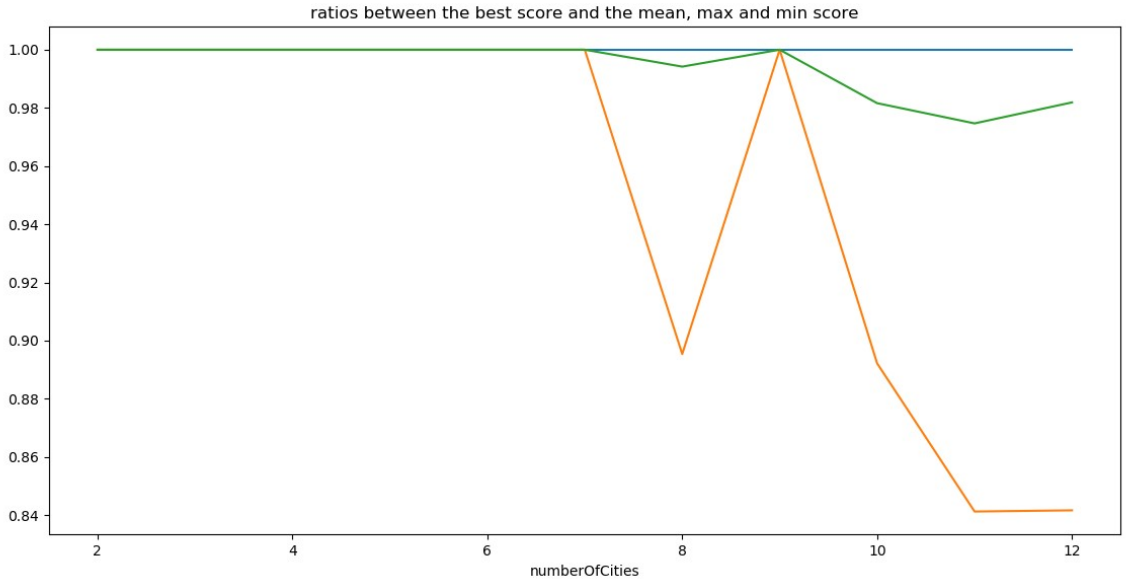
The best solution found by the exhaustive search algorithm is [0,1,9,4,5,2,6,8,3,7] which corresponds to ['Barcelona', 'Belgrade', 'Istanbul', 'Bucharest', 'Budapest', 'Berlin', 'Copenhagen', 'Hamburg', 'Brussels', 'Dublin']. The corresponding value is 7486.31. The running time ρ for 10 cities is 0.438. I plot the graph of running time depending on the number of cities :



We can't go further 12 cities in a resonable amount of time. The complexity is o(n !) which means that it exists a constant α such that ρ(n) = α*n ! (we assume some constant or polynomial running time are negligeable when n is large enough). For n = 12, ρ(12) = 12.2, so α = 12.2/12 ! = 2.55^{e}-8. Thus we have ρ(24) = 2.55^{e}-8 * 24 ! = 1.58^{e}16 seconds which approximately equals to 500 millions of years ! So it won't be solved before the deadline of the assignement that's why I coded smarter but not exact algorithm.

Hill Climbing :

The best solution found by performing localSearch from 20 differnents starting points for the 10 firsts cities has value 7486.31 which is the same as for exhaustive search. The mean is 7505.6, worst value 7737 and standard deviation 53.91. In order to compare performances of bruteforce I plot the ratio between score found by exhaustive search and best, average and worst score found by hillClimbing. We can see that hillClimbing always succeed to find the best solution in 20 runs for the 12 first cities.



Unlike bruteforce algorithm, hillClimbing is able to find a solution before the deadline for the 24 cities problem and it has found a solution of fitness 12745.68 which is [9, 4, 5, 23, 10, 14, 19, 21, 6, 8, 2, 17, 22, 15, 13, 16, 3, 11, 7, 12, 0, 18, 1, 20]. The worst is 16029, mean 14217 standard deviation 892.3.

I defined the neighboorhood at the set of permutations obtained by doing one swap from the current permutation.

Genetic/Hybrid Algorithms :

The parameters of my genetic algorithm are the population size, the number of generations, the list of cities , the mutation rate which corresponds to the probability that an individual mutates, the type of algorithm (« C » for classic, « L » for Lamarckian and « B » for Baldwinian. There is also a parametre n. During the survivor selection we select the n best fitted individual and the others are selected randomly among the others. So the maximum value of n is popSize.

An individual is represented by a couple made by a list (a permutation) and a float which is the value of the corresponding solution so that we do not need to evaluate solution each time we need its value. It saves a lot of time especially for Baldwinian algorithm in which we need to compute a local search to evaluate a solution.

For the parents selection I chose the following value of probability to select parent p_i :

$$P(p_i) = \frac{\max(\{d(p_k)\}) - d(p_i) + \varepsilon}{\sum_j (\max(\{d(p_k)\}) - d(p_j) + \varepsilon)}$$

The shortest the path is the highest is the probability. ε is there to avoid a zero division error if all the solutions have the same values.

I chose the partially map crossover for the reproduction which is adapted for conserving adjacency. Here the order is not important we can start from any city and turn in any direction the fitness will be the same if we conserve adjacency.

I chose a swap mutation of probability *mutationRate* to occur.

Here are the results found for 24 cities problem.

Algorithm	Population Size	Best	Mean	Worst	Stand Dev
Classic	10	13937	15404	16836	770
	20	13921	15001	16753	838
	30	12517	14126	15964	757
Lamarckian	6	12287	12970	13891	450
	10	12334	12738	13427	269
	14	12287	12506	12818	162
Baldwinian	6	12287	12453	12869	173
	10	12287	12436	12702	132
	14	12287	12471	12708	146

Parameters used : mutation rate = 1, number of generations = 100 for classic and 10 for hybrid algorithms, n = population size. The average is computed out of 20 runs.

For hybrids algorithms I chose smaller population size fewer generations it takes a lot of time to perform local search from every individual at each generations. We can see that the score 12287 is never beaten but very often reached, so it might be the global optimum or maybe a local optimum very hard to escape from.

We can also note that the performance increase singnificantly with the population size for classic genetic algorithm.

The three algorithms succeed to find the optimal solution for the 10 cities problem for almost 100% of starting populations. The running times are the followings in seconds :

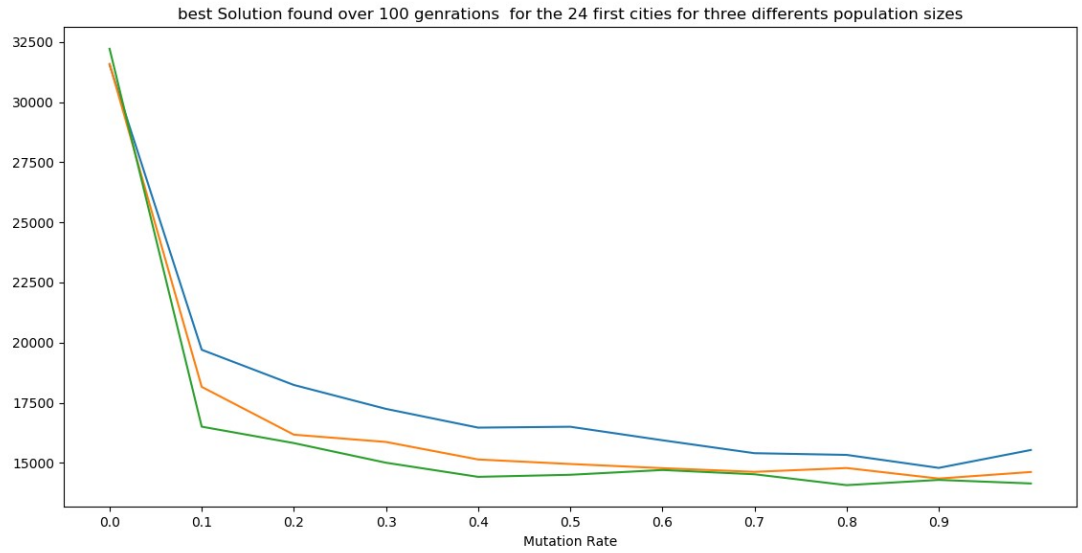
	Classic	Lamarckian	Baldwinian	Exhaustive
10 Cities	0.0914	0.0952	0.1235	0.4307
24 Cities	0.1699	0.9154	4.5768	A lot

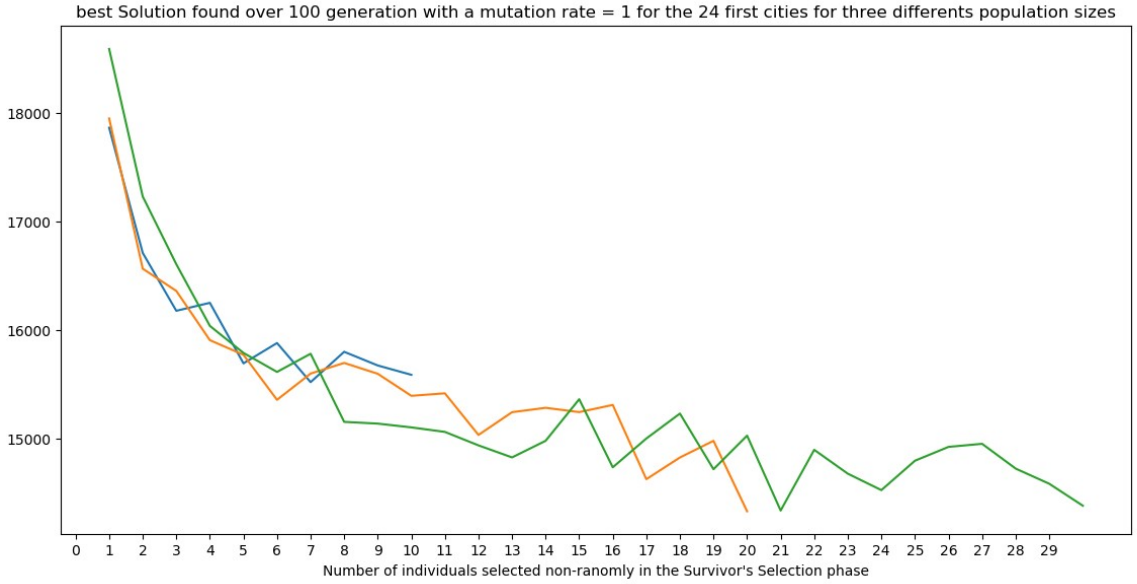
The number of generations is 100 for classic and 10 for Lamarckian and Baldwinian.

The difference of running time between Lamarckian and Baldwian can be surprising since both of them perform local search from each individual. But for Lamarckian children are made from local optimum so they are not far from local optimum themselves. Thus local search takes less time.

For classic genetic algorithm , at each genartaion we compute *popSize* fitness scores. So the the total number of evaluation is *popSize***nbGenerations* = 100*20 = 2000 << 24 !

For classic genetic algorithm I tried to figure out the influence of mutation rate and the selectivity of survivor selection (represented by n the number of non-randomly selected individual) I plot the average best results depending on these parameters.

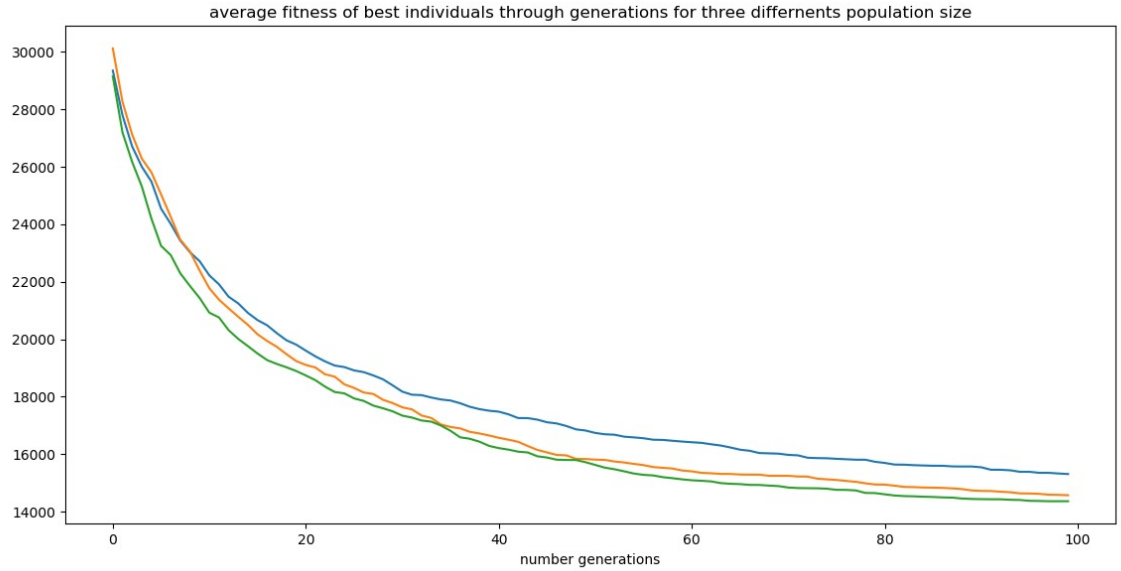


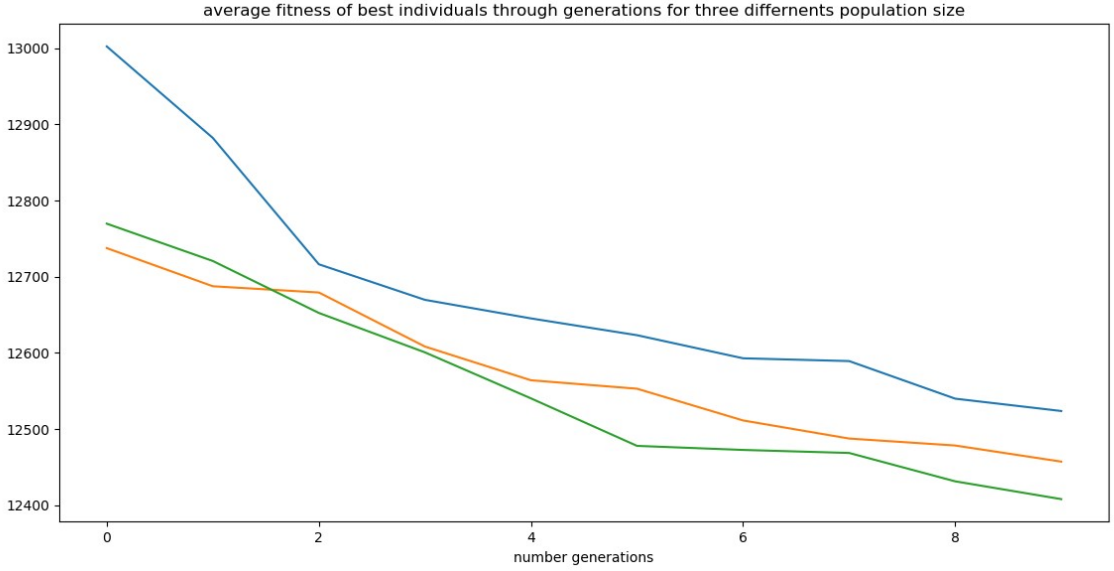


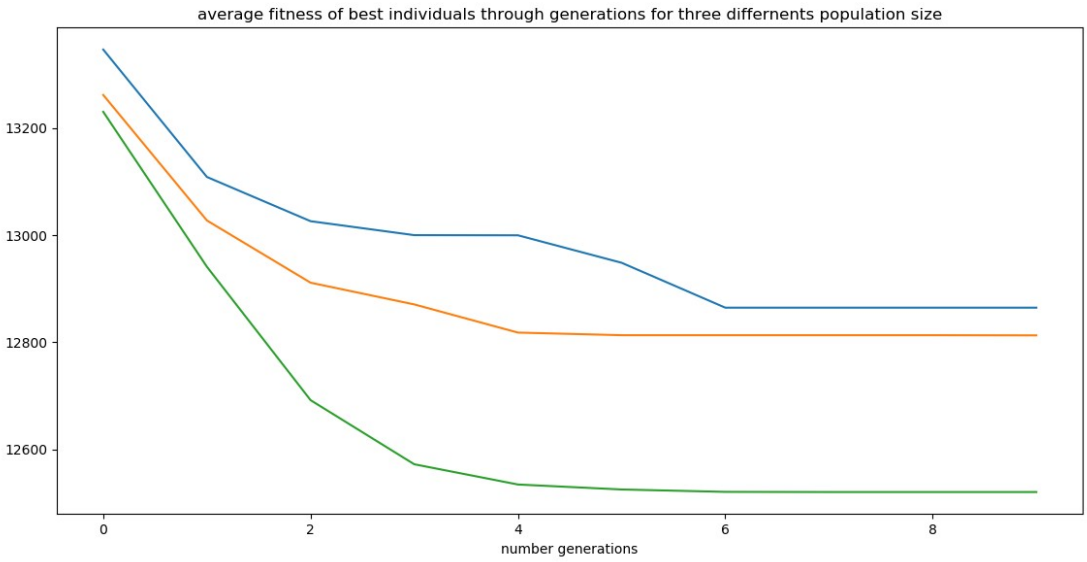
We can see that selection pressure tends to improve significantly the results, the best value of *n* is the population size according to the graph.

A very low mutation rate (0.1 or less) gives very bad results but for *mutationRate* > 0.1 it has not much influence on the performances.

I plot below the average fitness through generations for the three algorithms. From the top : Classic, Baldwinian, Lamarckian. For the three variants the distance tends to dicrease with the number of generations. Howeever it converges in fewer generations for hybrid algorithms. We can see that Lamarckian got trapped in local optimums after 6 generations which is not the case for Baldwinian.


average fitness of best individuals through generations for three differnents population size
30000
28000
26000
24000
22000
20000
18000
16000
14000
0
20
40
60
80
100
number generations


average fitness of best individuals through generations for three differnents population size
13000
12900
12800
12700
12600
12500
12400
0
2
4
6
8
number generations


average fitness of best individuals through generations for three differnents population size
13200
13000
12800
12600
0
2
4
6
8
number generations