Problem choice:

The idea is to pick a problem for each one of our algorithms to peak, MIMIC does well when the problem has an underlying structure that can be estimated to have good predictions, SImulated annealing is a very good approach when the problem does not have too many peaks but remains decent when it does and finally genetic algorithms are notorious for always being a best “second option” which means that it will probably outperform the other algorithms if we come up with a problem that has many peaks and a not too strong underlying structure.

1- Traveling Salesperson:

This problem consists of having a number of cities and distances between them, the goal is to find an optimal between that will visit all cities whilst travelling the least possible (we need to minimize a function so our fitness to maximize will be the opposite of the sum of the distances). This problem definitely has an underlying structure, but it should also have a distinctive peak when the number of cities (dimension of state vector) is not too high. We are hoping that simulated annealing would work best with this problem.

2- 2-Color Maximization:

This problem is about assigning colors to nodes in a graph and trying to maximize the same color neighbors, this has a very strong underlying structure as there are strong dependencies between the states (coming from for the dependencies between nodes and their parents), and it has also several peaks (as opposed to when we try to minimize the 2-color error). MIMIC algorithm will probably do the best job at solving this problem.

3- Four-Peaks problem:

This problem is about having a bit word as a state and score it based on the max between the number of leading ones and trailing zeros while applying an offset if both those scores exceed a threshold.
The problem has four peaks : 2 local and 2 global, and not much of an underlying structure. That means that both MIMIC and SA will struggle with it, which will allow the genetic algorithm to prove to us indeed that it is always a decent solution.

Random Optimization algorithms evaluation

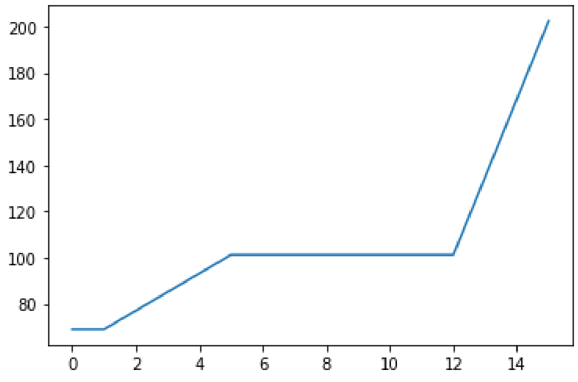
1.Traveling Salesperson:

Tuning of algorithms parameters for a traveler salesperson with 30 cities and randomized distances.

a.Random Hill Climbing tuning

Figure 1 - number of

The more our algorithm the hill climb would just starting point.



Score RHC as a function of restarts

there are restarts the likelier is to find a better solution as is completely random, we hope to have a "better"

b.Simulated Annealing Tuning

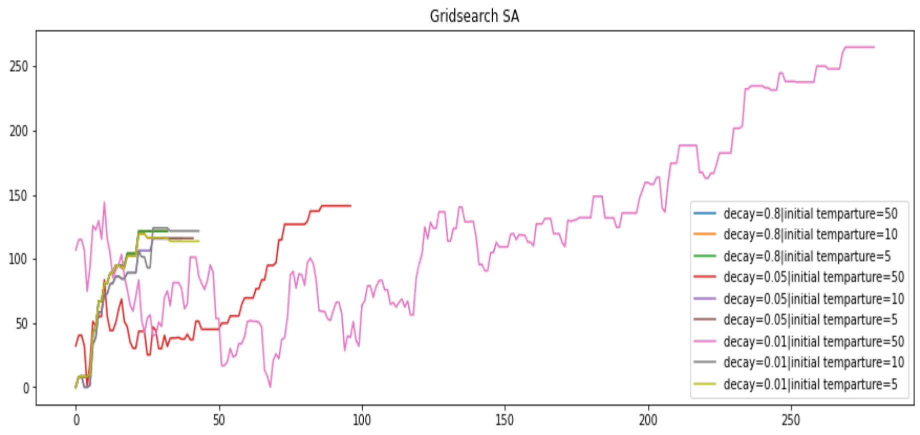


Figure 2 - Score SA function of iterations for different values of Temperature and decay

It is interesting that the parameters associated with the best score are the ones that fluctuate the most at first. Which makes sense because it corresponds to a high initial temperature and a low exponential decay, so it takes a lot of time to explore but eventually finds optimal solutions.

c.Genetic Algorithm

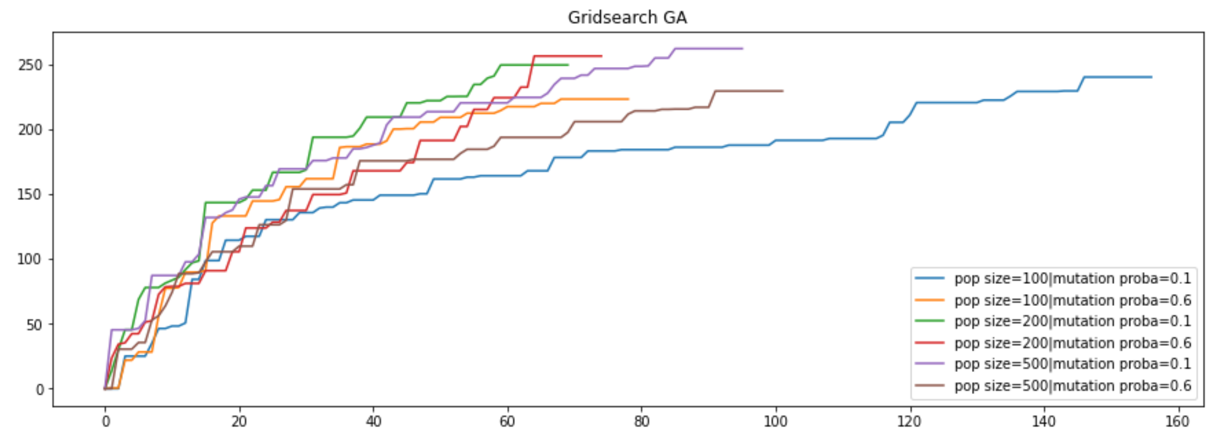


Figure 3 - Score GA function of iterations for different values of 'population size' and 'mutation probability'

Lower probabilities increase the necessary iterations for convergence, we get the optimal tuning for a population size of 500 and a mutation probability of 10% it is equivalent to the slower decay working better in the SA case.

d.MIMIC

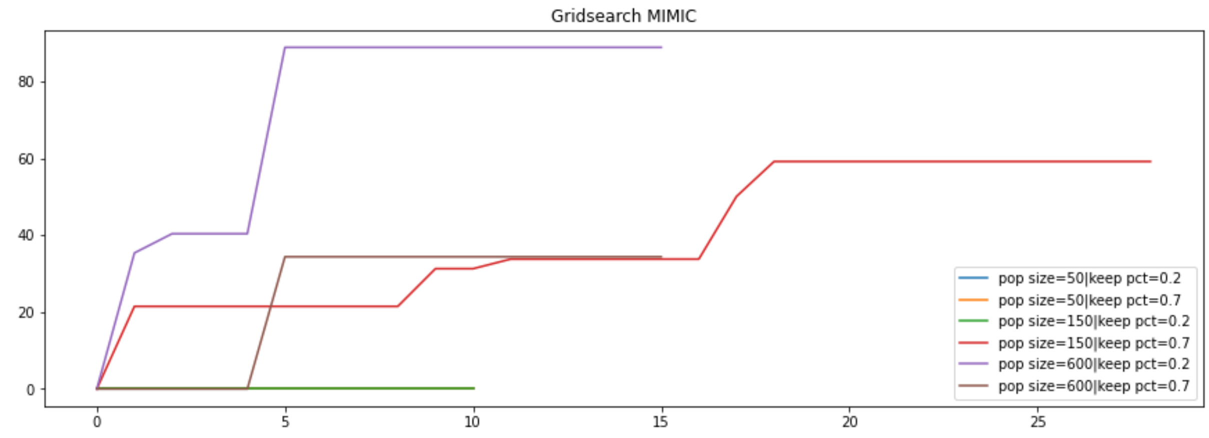
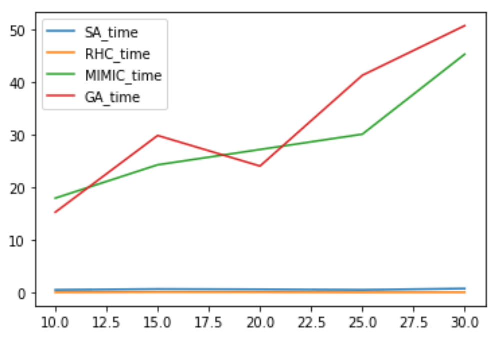


Figure 4 - Score MIMIC function of iterations for different values of 'population size' and 'percentage keeping'

Tuning is very crucial for MIMIC algorithms since we can't really afford too many iterations as they increase running time exponentially, but we can see with the right tuning (size =600, percentage keeping = 20%) we can have a nice convergence in a few iterations.

e.Runtime and score comparison on problem size



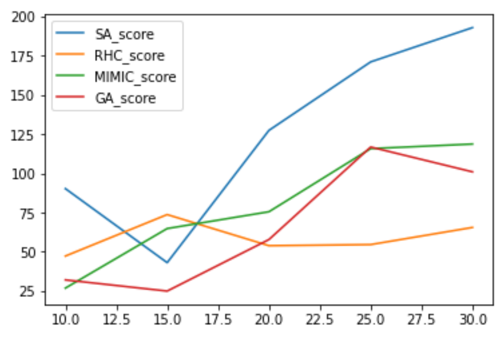


Figure 5 - Scores (right plot) and runtime (left plot) for all four algorithms as a function of problem size (number of cities)

The running times of RHC and SA are negligible in comparison with those of MIMIC and GA that increase strongly with the size of the problem.
Score-wise, the simulated annealing does best as expected for the scales of the problem we chose (results do differ when there is a very high number of cities).

2.2-Color Maximization

Tuning of algorithms parameters for a 2-color maximization with 15 nodes and 50 edges.

a.Random Hill Climbing

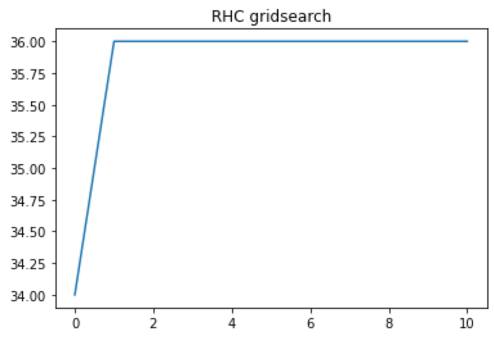


Figure 6 - Score RHC as a function of number of restarts

The number of restarts does not seem to affect the optimum that the model ends up in, this means that no matter where we start we end up stuck in a local optimum, which is why this algorithm is weak on this particular problem.

b.Simulated Annealing

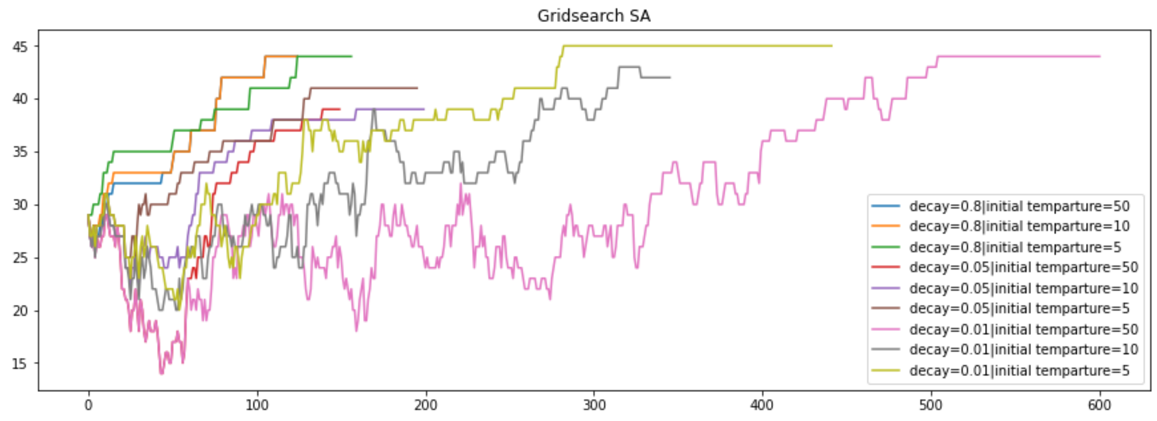


Figure 7 - Score SA function of iterations for different values of Temperature and decay

The problem here is the same as with RHC with the exception that the exploration part during the high temperature allows the model to escape its local optimums and it ends up performing better but still not optimally.

c.Genetic Algorithm

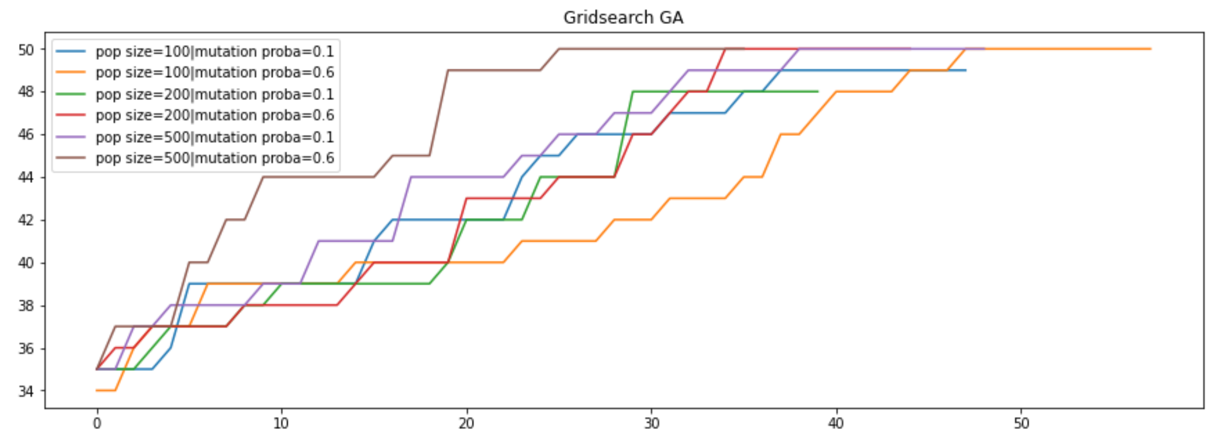


Figure 8 - Score GA function of iterations for different values of 'population size' and 'mutation probability'

d.MIMIC

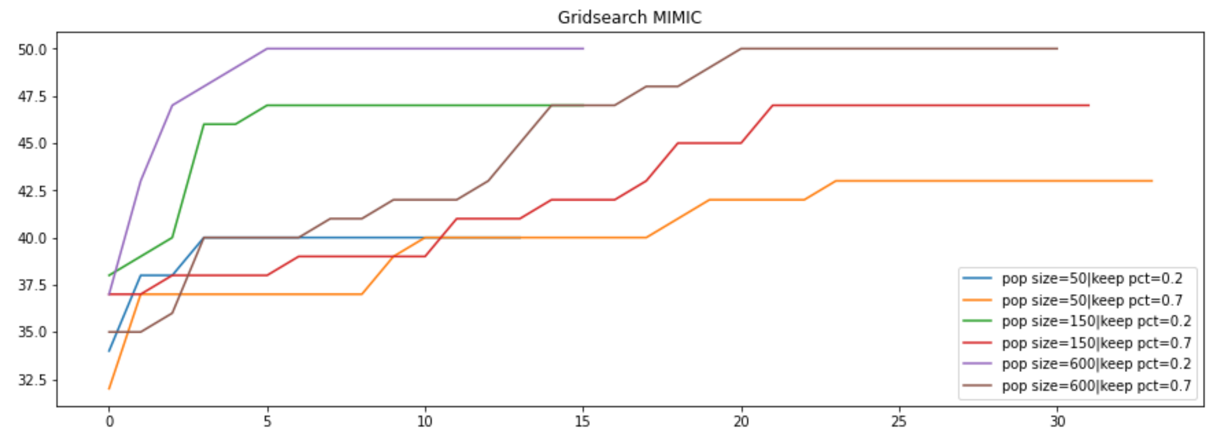


Figure 9 - Score MIMIC function of iterations for different values of 'population size' and 'percentage keeping'

Comparing figure 8 and 9 we can see that these two algorithms converge the best (better score and faster) for high population size and high mutation probability/ keeping percentage.

e.Runtime and score comparison on problem size (edges number = 3xnodes number)

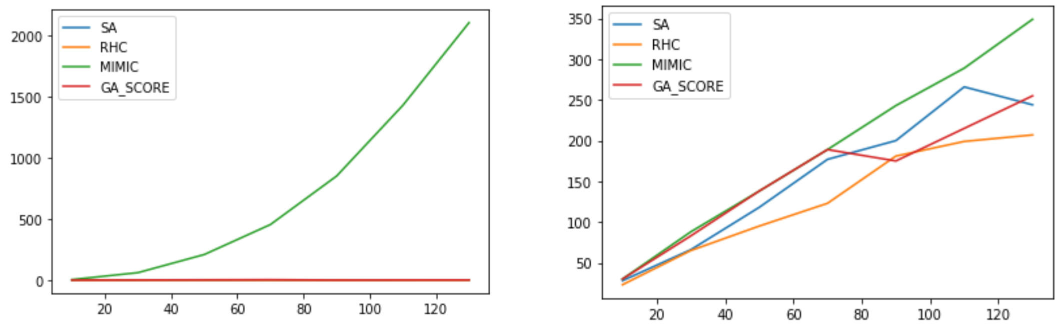


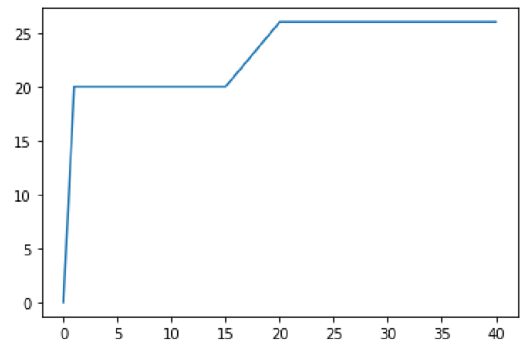
Figure 10 - Scores (right plot) and runtime (left plot) for all four algorithms as a function of problem size (number of nodes)

The Mimic algorithm clearly outperforms the other algorithms, it is because there is a lot of information that is captured between state changes, but it also has an “exponential’ increase in runtime when we increase the problem size (number of edges).

3. Four-Peaks

Tuning of algorithms parameters for a 4-peaks with a size 15 vector.

a.Random Hill Climbing



This problem has 4 peaks, 2 optimal and two local, so it is clear that increasing the number of restarts should allow the algorithm to have better chances of starting somewhere when he can find a global optimum when the vector’s dimensionality is not too high.

Figure 11 - Score RHC as a function of number of restarts

b.Simulated Annealing

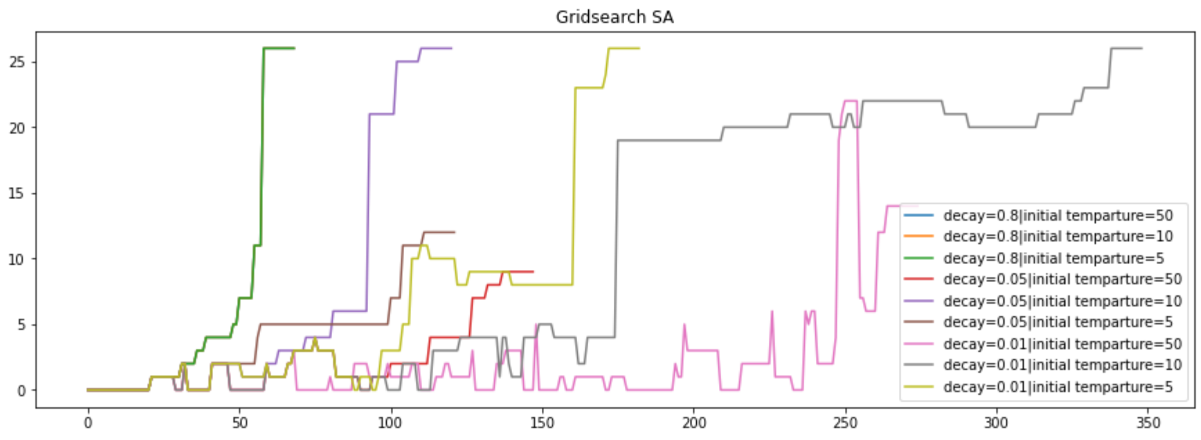


Figure 12 - Score SA function of iterations for different values of Temperature and decay

The simulated annealing also does fairly well, especially when we find a right tuning, strong decay and not too high initial temperature, there isn’t a big need for exploration, since the problem is not maxima’s numbers which is only four, but rather the somewhat even distribution of states around them.

c.Genetic Algorithm

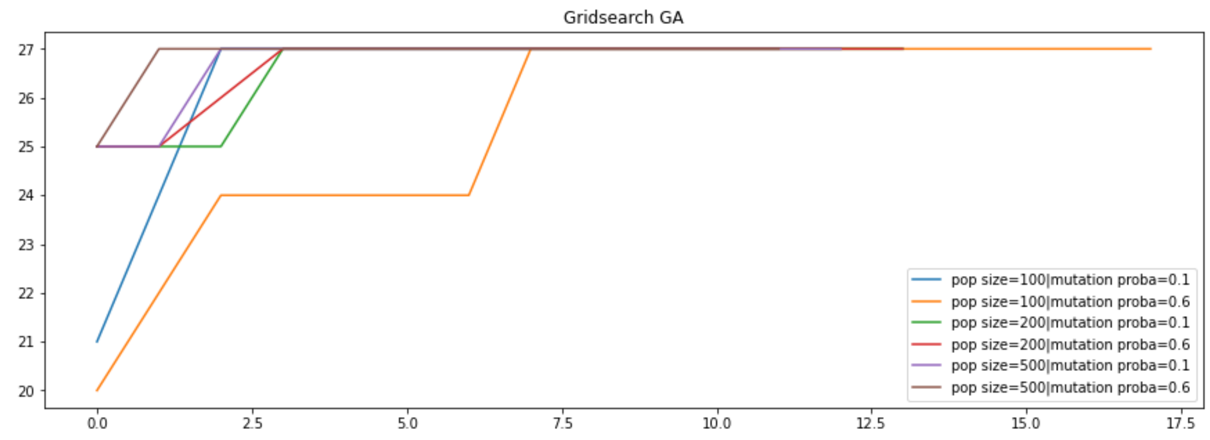


Figure 13 - Score GA function of iterations for different values of 'population size' and 'mutation probability'

It seems that this algorithm is not very sensible to tuning, it converges fairly fast to the optimal solution for this problem size.

d.MIMIC

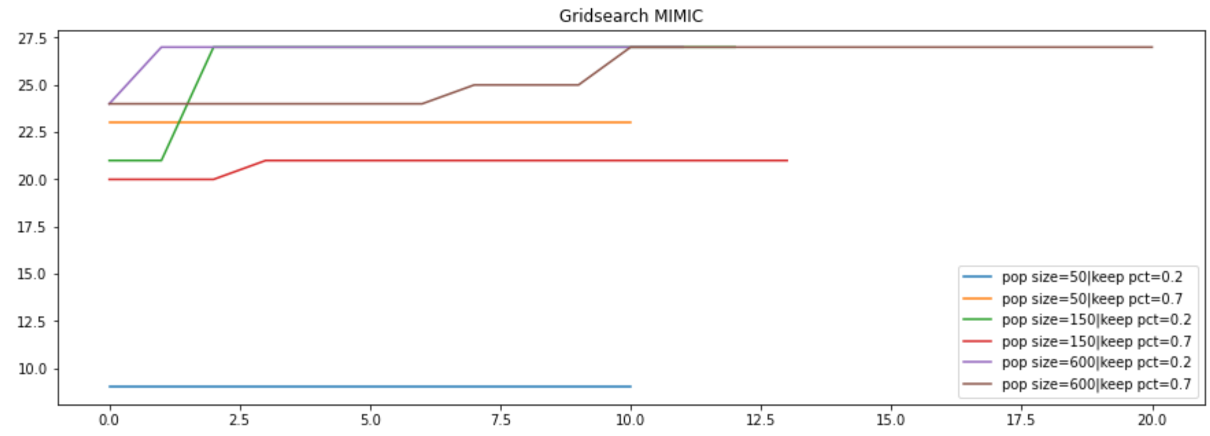
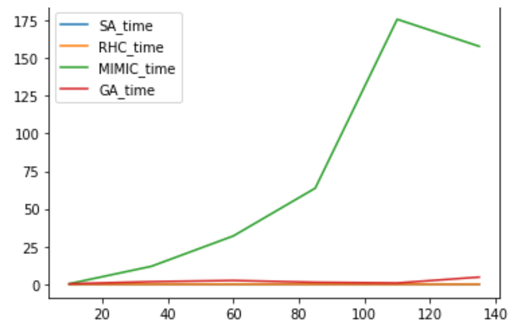


Figure 14 - Score MIMIC function of iterations for different values of 'population size' and 'percentage keeping'

We notice here that the per iteration changes are very limited, there is also a lot of randomness in this case, because there isn't really a strong underlying structure for this problem.

e.Runtime and score comparison on problem size



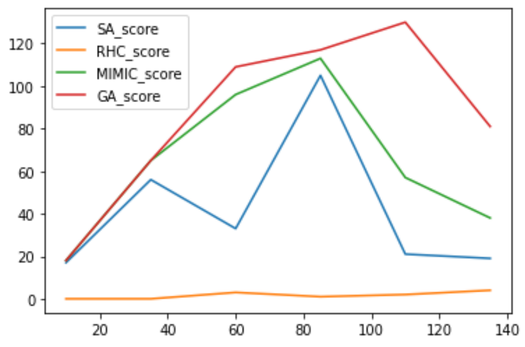


Figure 15 - Scores (right plot) and runtime (left plot) for all four algorithms as a function of problem size (vector dimension)

The genetic algorithm reaches the best score for every problem size, while having a higher runtime than increase logarithmically with size. The fact that MIMIC and SA were unlikely to be optimal for this particular problem makes that the genetic algorithm is the most decent option.

Neural Network Weights Optimization

Dataset and evaluation:

This will be based on a binary dataset that we previously explored in the first assignment, which is a dataset that based on some patient information (age, gender, some chemicals's concentration etc…) predicts whether the patient has a liver disease.

We previously achieved the best results with a one layered Neural Network with around 20 neurons, we'll fix that as our baseline and we will use a similar architecture but with a weight calculation based on the optimization methods seen above.

We will evaluate the algorithms based on the accuracy and f1-score to obtain insight on the classification's accuracy. But we will also evaluate the runtime of the methods, the number of iterations and the score per iteration to measure the quality of our randomized optimization. Metrics for gradient descent :

accuracy train: 0.75
accuracy test: 0.74
f1_score train: 0.52
f1_score test: 0.44

Algorithms Tuning Comparison

RHC Tuning							
Restarts	**Train Acc**	**Test Acc**	**Train F1**	**Test F1**	**Iterations**	**Time (seconds)**	**Time per It**
1	0.29	0.25	0.45	0.41	120	0.41	0.003
2	0.65	0.58	0.33	0.22	1000	6.09	0.00609
5	0.69	0.68	0.24	0.14	1000	12.5	0.0125

SA Tuning							
Init Temp/Decay	**Train Acc**	**Test Acc**	**Train F1**	**Test F1**	**Iterations**	**Time (seconds)**	**Time per It**
T=50, decay=0.8	0.68	0.69	0.27	0.19	1000	2.71	0.002
T=20, decay=0.5	0.69	0.69	0.46	0.43	1000	2.86	0.00286
T=20, decay=0.01	0.7	0.73	0	0	1000	2.67	0.00267
GA Tuning							
Pop size/Mutation prob	**Train Acc**	**Test Acc**	**Train F1**	**Test F1**	**Iterations**	**Time (seconds)**	**Time per It**
S=100, p=0.1	0.72	0.73	0.1	0.06	100	18.5	0.185
S=50, p=0.2	0.72	0.75	0.08	0.06	100	10.13	0.1013
S=10, p= 0.8	0.55	0.45	0.3	0.22	100	2.95	0.0295

Figure 16 - Interesting gridsearch results for all the algorithms (score and time comparison)

Figure 16 shows some notable examples for a gridsearch to find the best optimizers to fit the weights.

GA always has low scores for F1 combined with high accuracy which means that it just classifies everything as the predominant class, which is a local optimum for the loss function and therefore the fitness. It also seemed to do better on smaller population sizes.
Running time for GA was the highest, but the fact that it did not need high population sizes makes it still acceptable.

For the RHC and SA the starting points are very important, which means that there is a lot of randomness on the final result we achieve.

- For RHC we seem to have gotten a lower f1 score by increasing the number of restarts when in reality it just means that the initialization was just luckier in the case of one restart which is why the algorithm also converged faster (only 120 iterations).

- SA requires us to allow a lot of exploration without making it completely random, we do not want it to be stuck in a local minimum like the GA but we still want it to learn, so in this case an exponential decay with a temperature of 20 and a decat of 0.5 does the best.

These two algorithms have a low and constant cost per iteration and when we tune them right, they can give a result almost as good as with the gradient descent.
“Almost” because solving numerically for the best loss using a gradient is a very hard method to beat.

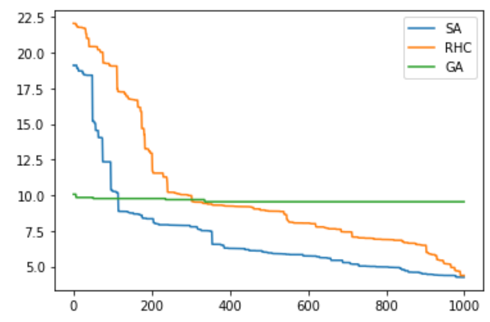


Figure 17 - Fitness score per iteration comparison for the three algorithms

This figure shows that SA and RHC both end up converging towards the same fitness i.e the same loss, but a problem with GA is that it gets rigidly stuck. Indeed the weights update just enough to classify all the examples as the predominant label, then just diverge in this same direction (very negative weights that will always set the activation function to 0, no matter the attribute values) without ever finding the global optimum, and this no matter what the values of population size, mutation probability, and even learning rates. My guess is that it would converge and probably yield better results than RHC and SA, but it would just take an insane amount of time (iterations) as the vector is 240 dimensional, and we would need to mutate enough towards our global optimum which is against the updates that the model does on local steps towards local maximas.