Machine Learning – Assignment 2

Author: Mohamed Amine DASSOULI

Table of Contents

1. Introduction

In this report, we are going to approach randomized optimization techniques, which tend to find global optimums of a fitness function. We are going to evaluate and compare them in different situations. There are two main parts in our analysis:

- The first part is about optimizing the weights of a Neural Network.
- The second part is about three optimization problems that we are going to define later on.

A presentation of the algorithms and the technologies used is following next.

1.1. Algorithms

1.1.1. Random Hill Climbing (RHC)

It is based on Hill Climbing which starts from a random state and tries to find a better solution to the problem at each iteration using an incremental change, until there is no more possible improvement. Random Hill Climbing repeats the Hill Climbing process many times, from different initial solutions, in order to avoid local optimums.

1.1.2. Simulated Annealing (SA)

It is similar to Hill Climbing, but it uses another trick to avoid local optimums. The algorithm introduces the concept of temperature 'T', which allows to occasionally accept worse solutions. 'T' is initially high, allowing to do a lot of exploration, and it decreases at each iteration following a rate, until the system is cooled and only the best solutions are accepted in the end.

1.1.3. Genetic Algorithms (GA)

It is a method inspired by the theory of natural evolution, where the fittest individuals are more likely selected for reproduction. Genetic Algorithms pair up the parents from a population and place the new offspring in the children, using crossover while allowing mutation with a certain probability. The process is then repeated until an optimal solution is found.

1.1.4. Mutual-Information-Maximizing Input Clustering (MIMIC)

Similarly to GA, MIMIC studies a population, however it uses a different approach based on a density estimator. At each iteration, the samples are evaluated using the estimator, and then only a part of it is kept according to a threshold. A new threshold is then established and a new density estimator is created, repeatedly.

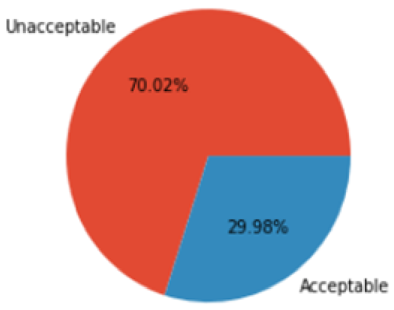
1.2. Technologies used

To get our results, we used Python. Concerning the libraries, we used *"mlrose"* for the first part, and we overwrote several classes and functions (The Python files can be found in the GitHub repository). Concerning the second part, we used *"mlrose_hiive"*. For each part, there is a Jupyter Notebook also available in the Git.

2. Neural Network weights optimization

2.1. Dataset

We are going to use the dataset *"Car Evaluation"* from the first assignment, available in Kaggle "https://www.kaggle.com/ankiijindae/car-evaluation/download". It is a multiclass classification problem about evaluating the quality of a car offer based on its physical qualifications. The dataset has a size of (1728,7), there are 1728 samples and 6 features.



- There are four different labels: *"unacceptable"*, *"acceptable"*, *"good"* and *"very good"*. But for the sake of having a binary classification problem and a more balanced data, we can combine *"acceptable", "good"* and *"very good"* into a single class *"acceptable"*. So in the end, we have two classes:

- Negative class with a ratio of 70.02%
- Positive class with a ratio of 29.98%

Before merging the three minority classes, the minority class was representing 3.76% making the data very imbalanced, so we used the balanced accuracy (average recall) as a metric in the first assignment, but now that the minority class represents almost 30%, the data is way less imbalanced and therefore we can simply use **the accuracy** as a metric.

The dataset is made according to a structural decision model, so our prediction models (different neural networks) can reach 100% accuracy if they find the correct correlation between the features, that's the interesting part about this dataset.

2.2. Model Architecture

The neural network we chose has:

- An input layer made of 21 nodes corresponding to the encoded features
- A first hidden layer containing 16 nodes, with a RELU activation function
- A second hidden containing 4 nodes, with a RELU activation function
- An output layer containing a single node, with sigmoid activation function

The model is fitted using 80% of the data (training set) and tested on the last 20% (testing set).

2.3. Tuning random optimization algorithms

2.3.1. Random Hill Climbing

RHC is a quite simple algorithm comparing to Simulated Annealing and Genetic Algorithms, it will just check the fitness values of the neighbors and go to a location with higher value, so there is not much to tune.

2.3.2. Simulated Annealing

The essence of SA is about the notion of temperature. When T is high, there is a high probability to go to a location with lower fitness value and the algorithm becomes similar to a random walk. In the opposite, when T is low, the probability to go to a neighbor with a lower value of fitness becomes almost null, and therefore the algorithm starts acting like a Hill Climbing. Therefore, controlling the evolution of T is what it is important in SA, and it is done using a decay schedule. It can be exponential, geometric, arithmetic or some custom one.

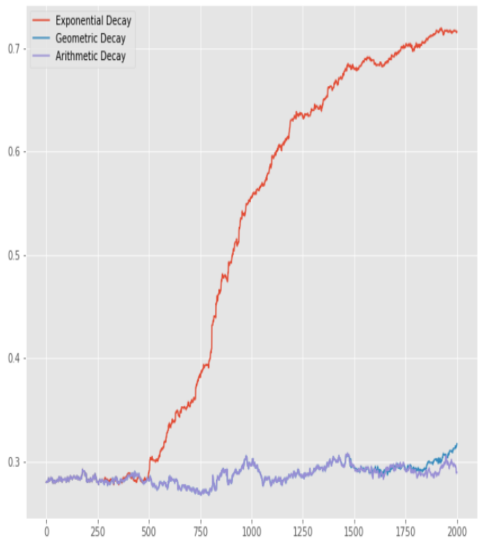


Fig 2.3.2.a: Comparing Decay schedules

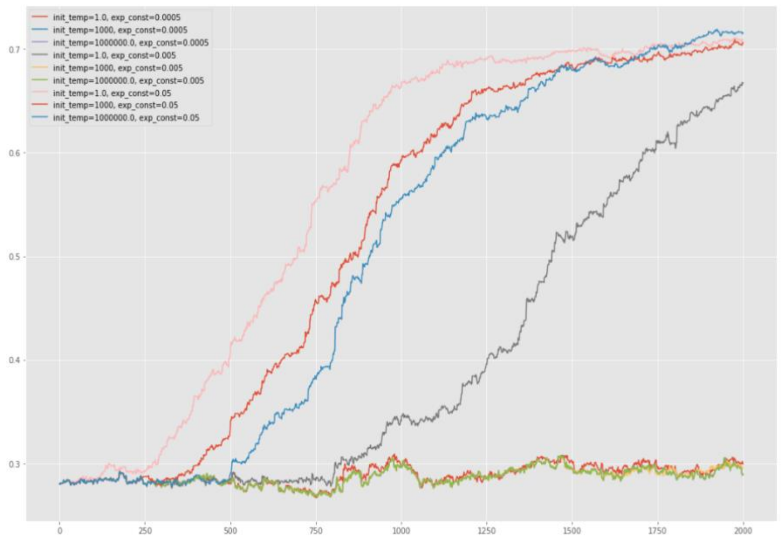


Fig 2.3.2.b: GridSearch Simulated Annealing

Fig 2.3.2.a shows the result (fitness value, function of number of iterations) for the three decays, with default values. The exponential is way better than the two others, and it is explainable. The arithmetic decay is linear **($T(t) = T0 – rt$)**, and it is not suitable for our problems because if T is big, then no matter the value of r, we are going to spend a lot of time exploring comparing to improving fitness locally, since T will reach 0 in very few iterations once in cooling zone. The geometric decay makes a better compromise between the time spent in exploration and the time spent in cooling area, so it might work better, but the exponential decay remains the best because it gives a perfect trade-off (the exponential term allows a quick decrease of temperature, while in high values, and slower decreases as the lower values are reached).

Next, we are going to use the exponential decay, so there are two parameters to tune: the initial temperature and the exponential constant. **Fig 2.3.2.b** shows the result of the Grid Search done, the best value of fitness corresponds to an initial **temperature of 10^6** and **0.05 for the exponential constant**.

2.3.3. Genetic Algorithms

GA has a more complicated approach. Each iteration, it computes the fitness for all population, it selects the most fitted individuals and then pairs them up, either by crossover or mutation (with a certain probability).

Therefore, there are two parameters to tune: the size of population considered and the mutation probability.

Fig 2.3.3 shows the result of GA for different pairs of values, the best one corresponding **to a population size of 200** and **a mutation probability of 0.05.**

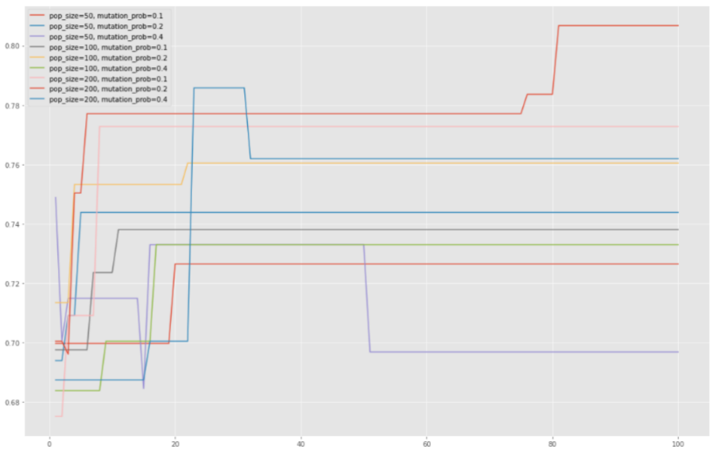


Fig 2.3.3 : GridSearch Genetic Algorithms

2.4. Comparing results

Here below are the results for the three algorithms, plus back propagation (BP). **Fig 2.4.a** shows the accuracy of the four neural networks on the testing set at each iteration, and **Fig 2.4.b** shows the global running time of the algorithms at each iteration.

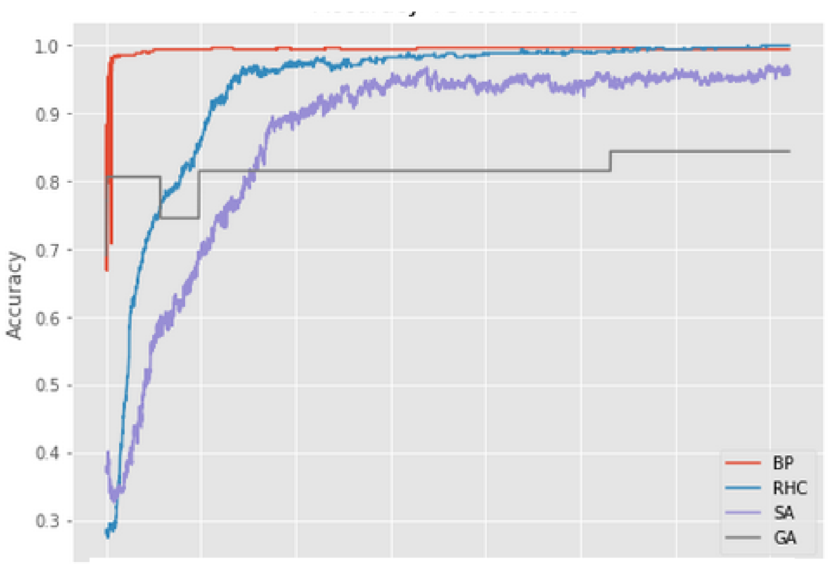


Fig 2.4.a: Performance of the algorithms on the testing set

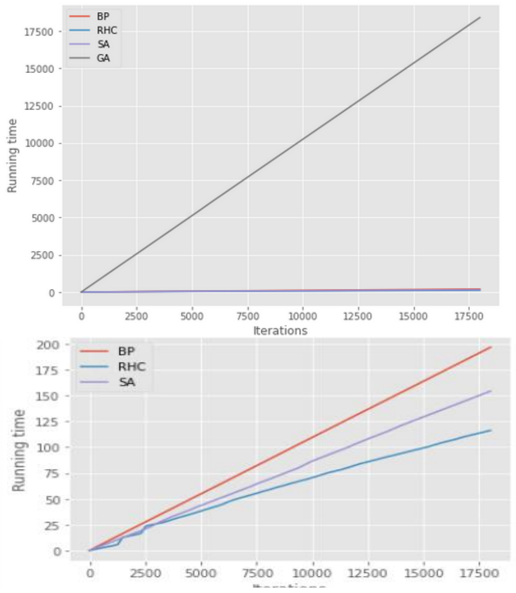


Fig 2.4.b: Running times (s)

- Back propagation (BP) takes 10.93 ms/iteration and it converges in very few iterations to 100% accuracy, making it the most efficient algorithm out of the four.
- Random Hill Climbing (RHC) is the fastest algorithm in term of time per iteration, it takes only 6.46 ms/iteration and it succeeds on reaching 100% accuracy, it is a very simple and computationally light solution. However, a lot of iterations were required to find the optimal solution, it is because the algorithm checks only the neighbors' values at each iteration and slightly updates the weights without sudden change, making the convergence slow. On the opposite, BP looks at the derivatives and adjusts all the weights at once, allowing it to make bigger steps toward the optimum and converge faster.

- Simulated Annealing (SA) is a quite fast algorithm as well, it takes 8.57 ms/iteration. It is faster than BP but slower than RHC because there is more computation (temperature, probabilities…) in order to determine the neighbor to jump to. The model does not converge though, probably because it did not have enough time. Indeed, the algorithm does not get invested directly in improving the fitness like the RHC, but it starts with an exploration phase, allowing it to avoid local optimums. This can be seen in the left figure: the accuracy of SA decreases during the first iterations, it is because of the high temperature in the beginning, which makes the algorithm likely to accept and move to neighbors with lower fitness value. In the end, the temperature becomes low and the algorithm tends to act like a Hill Climbing, accepting only higher values of fitness, and therefore getting more accuracy score on the testing set.
- Genetic Algorithms (GA) is very slow comparing to the three other algorithms, each step takes approximately 1 sec/iteration, which is the equivalent of 158 iterations for RHC. This is because an iteration of GA is highly computational. Each step, the fitness is calculated for all the population, then the fitted individuals are selected and paired up. The model seems to converge in the end, however the accuracy does not exceed 84.39%. There are different potential scenarios that can explain that. The first is that the model did not truly converge, even though the accuracy did not change for the last 5000 iterations. It is a plausible scenario since it already happened in the iteration ~2500: the accuracy remained 81.50% for more than 10 000 iterations before moving to 84.39% at the iteration ~13200. A second possible explanation is that GA does not fit well our dataset. The algorithm tries getting to closer positions to the global optimum by doing mutations and crossovers of weights' positions, so the new positions are usually a bit far from the parents. This is a good approach to avoid getting stuck at local optimums, however it makes finding global optimums very hard if they are situated on a very tiny convex area. In this hypothetical case, GA will be more likely to go out or zigzag over the area without ever getting inside it, contrary to BP for example whose step size would be small if taking a low learning rate, allowing him to find the optimum. Overall, GA has the worst performance out of the four algorithms.

2.5. Ways of improvements

- For RHC, we can maybe implement a version which looks at an extended neighborhood and makes bigger incremental changes, to accelerate the convergence. And in a more general perspective of the algorithm, we can cache all the visited positions to accelerate even more the fitness calculation.
- For SA, the model still hasn't converged in the end of the 18000 iterations, so we could have given it more time to converge by increasing the number of maximum iterations, the accuracy score would have probably reached 100%. Another approach would have been to accelerate the cooling down, either by decreasing the initial temperature or by increasing the exponential decay constant. In the grid search **Fig 2.4.a**, the curve of init_temp=10^3 was very close to the one we used with init_temp=10^6 (for the same decay constant), so it could have had a better evolution after few iterations, giving us better results.
- For GA, the computational time depends on the population size and the mutation rate. The bigger is the population, the higher is the time per iteration, also the higher is the mutation rate, the higher is the time per iteration (more details are in the notebook). So in order to improve the speed of the algorithm, we can try a lower population size (we only

tried 50, 100 and 200 in the gridsearch, and 50 was optimal), we could have also tried 20 or 40 for example. Increasing the number of iterations might also improve the final performance of the model, if the second scenario is wrong and the area around global optimum is not that tiny.

3. Optimization

3.1. Problem 1: Travelling Salesman Problem

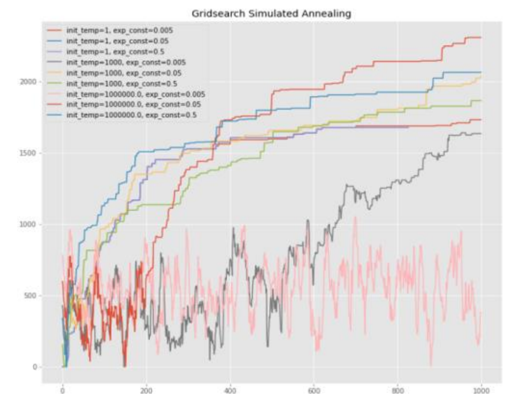
The problem is about a salesman who wants to find the shortest path to follow in order to visit a fixed number of cities, once each, and return to his starting point. For our study, we consider a maximum number of 50 different cities.

3.1.1 Grid Search

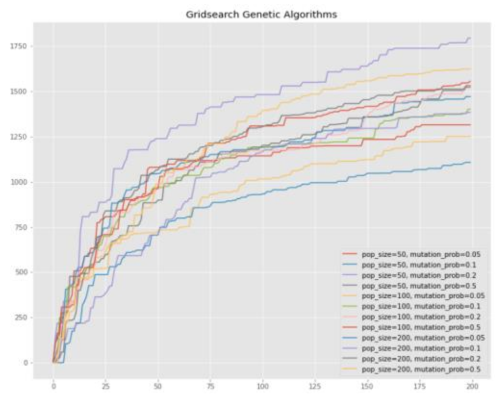
First, we are going to apply a GridSearch to tune the hyperparameters of each algorithm: Simulated Annealing, Genetic Algorithm and MIMIC.

We discussed the parameters of the first two algorithms in the previous part. Concerning MIMIC, there are two parameters to tune:

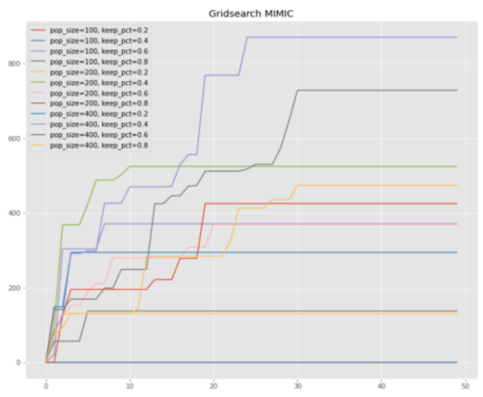
- 'pop_size': the population size to consider
- 'keep_pct': the proportion of samples to keep at each iteration



Best parameters: init_temp=10^6 & exp_const=0.05



Best parameters: pop_size=50 & mutation_prob=0.2



Best parameters: pop_size=400 & keep_pct=0.4

3.1.2 Results

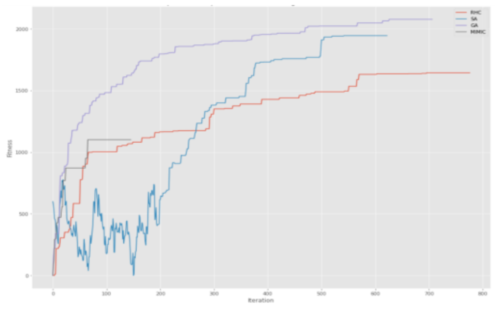


Fig 3.1.2. Optimization for Travelling Salesman Problem

Algorithm	Wall clock time
SA	0.11 s
GA	10.53 s
MIMIC	386.0 s
RHC	0.13 S

- MIMIC converged quickly (less than 150 iterations), but it has the worst wall clock time (very heavy in term of computation) and the worst maximum fitness value (probably because it got trapped in a local optimum). This was expected because MIMIC tries mostly to capture the global structure of the optimization algorithm, but here it faces TSP which has a very complex and large hypothesis space.

- RHC and SA are the fastest algorithms. They perform both better than MIMIC, and SA performs best. The temperature of SA was set at 10^6 in the beginning, allowing it to explore and find better start for hill climbing (the zigzags in SA curve in the first iterations show the exploration). In addition to that, it looks like RHC got trapped in a local optimum (it started converging to small fitness value); this shows the robustness of SA and its temperature concept against local optimums.

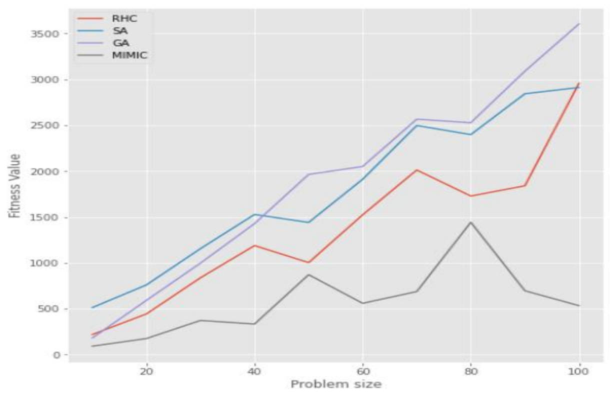
- GA takes more time per iteration than RHC and SA, but it performs better from the very beginning and it reached the highest fitness value at the end, making it the most suitable algorithm for Travelling Salesman Problem. This is explainable by the power of crossover and mutation in exploring the hypothesis space and finding more fitted individuals (solutions) at each step. This makes GA useful for complex fitness functions with a large space.

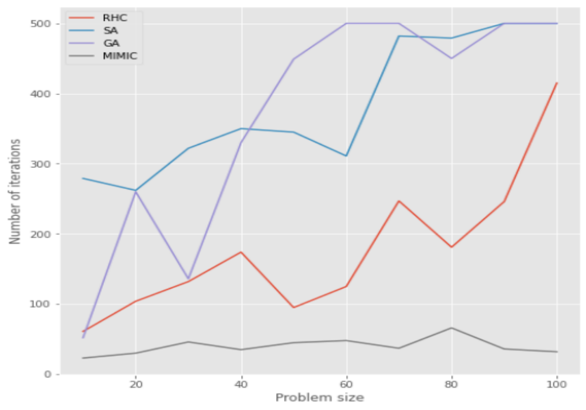
3.1.3. Ways of improvements

- MIMIC can be improved by making it more selective, so that it does not fall for local optimums anymore, increasing population size might help to achieve that.
- RHC needs more time to find the global optimum, because once stuck in local optimum, it starts a new hill climbing scenario looking for better result.
- For SA, we can increase its initial temperature allowing it to do more exploration since the problem is very complex.

3.1.4. Varying problem size of TSP

In order to see how the algorithms, and especially GA, behave when the size of the problem is changed, we tried different values and the results are shown in the figures below:



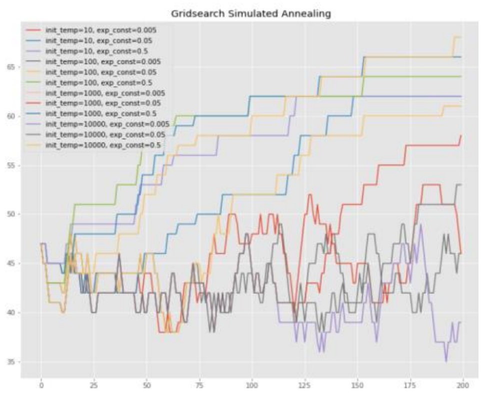


GA is always leading overall, followed by SA/RHC, and MIMIC being the worst. RHC and MIMIC always get trapped to local optimums (converge to low fitness value).

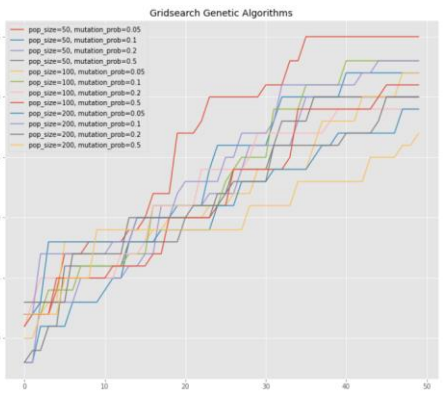
3.2. Problem 2: Flip Flop

Flip Flop is a bit string problem, it is about maximizing the number of flipped bits in a bit string. We consider a Flip Flop problem of size 80 for the following analysis.

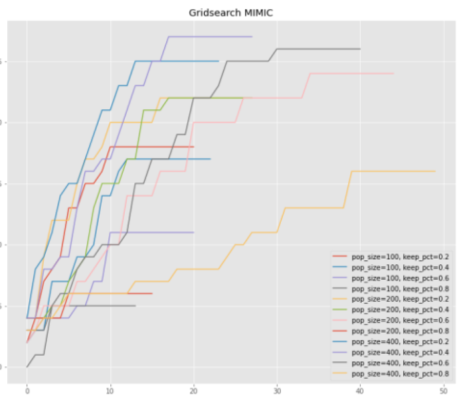
3.2.1. Grid Search



Best parameters: init_temp=10^6 & exp_const= 0.5



Best parameters: pop_size=50 & mutation_prob=0.05



Best parameters: pop_size=400 & keep_pct=0.4

3.2.2. Results

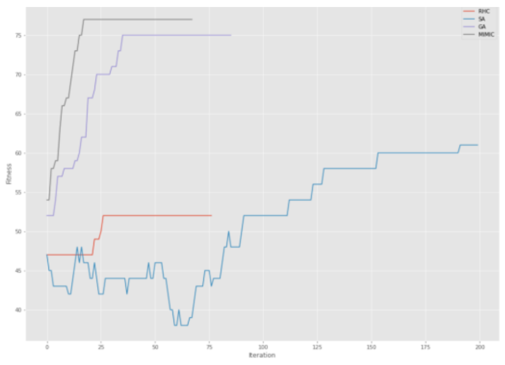


Fig 3.1.2. Optimization for Flip Flop

Algorithm	Wall clock time
SA	0.08 s
GA	1.18 s
MIMIC	234.48 s
RHC	0.02 S

- RHC and SA have once again the lowest wall clock time. RHC got stuck in a local optimum and the algorithm stopped after only 75 iterations due to the maximum number of attempts set to 50. SA didn't fall for local optimums, it achieved better performance than RHC, but it didn't have enough time to finish its hill climbing and converge. Both the algorithms did not perform well comparing to MIMIC, because of the problem nature. Flip Flop has an overall structure (the fitness is based on the relation between bits), and unfortunately RHC and SA only pay attention to the variation of bits' values on a singular level, independently from any kind of structure.
- GA has a lower time per iteration than the previous problems because the mutation probability is only 0.05, and it converges very fast (1.18 s) to a local optimum. It performs better than RHC/SA but worse than MIMIC. It is probably because the crossover does not help much with the Flip Flop since its the overall structure of the bit string that counts for the fitness function and not the single bit values.
- MIMIC converges in very few iterations and has the best fitness value (77), it is the best algorithm to use for Flip Flop, which has an overall structure. The algorithm, using density estimators, tries to capture it. Its memory allows him to consider the whole bit string, and not care about changes in the single values of bits; resulting on a

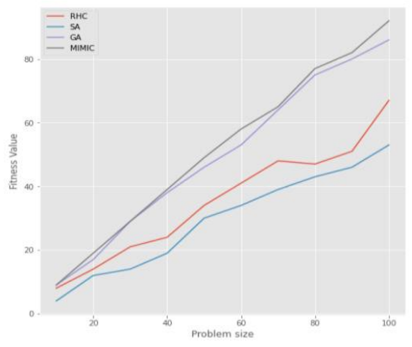
reconstruction of a higher fitness bit string at each iteration. The wall clock time of MIMIC is nevertheless still high because of all the necessary computation.

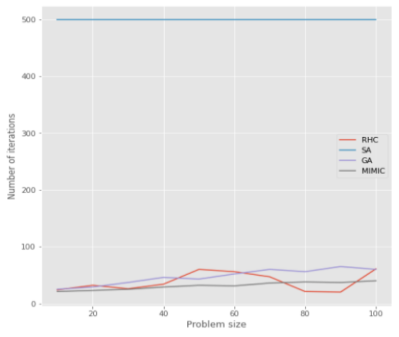
3.2.3. Ways of improvements

- For RHC and SA, we can increase the number of maximum attempts and the maximum number of iterations in order to do more Hill Climbings and find eventually the global optimum.
- For GA, mutation is the key against Flip Flop since it can change the structure of the bit string (for example '000' can become '010' after a mutation, making GA earn more 2 fitness points). We can therefore increase the mutation_rate.

3.2.4. Varying problem size for Flip Flop

From the figures, MIMIC gets the highest fitness value and the lower number of iterations for different problem sizes, make in it the most efficient algorithm to solve Flip Flop.

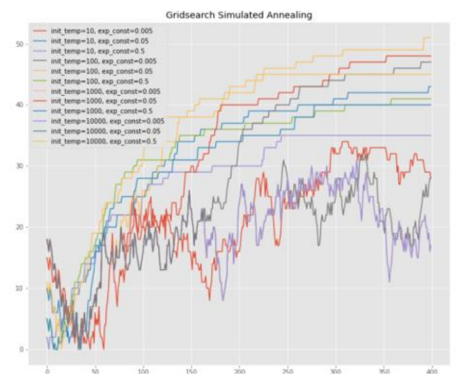




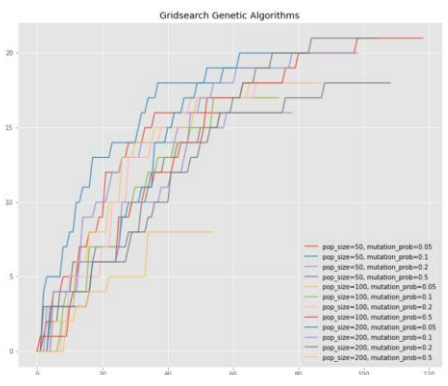
3.3. Problem 3: N-Queens

The N-Queens problem is about placing chess queen pieces on a N×N chessboard in such a way that no two queens threaten each other. In other words, two queens cannot share the same row, column, or diagonal. In the following parts, we consider a 40-Queens problem.

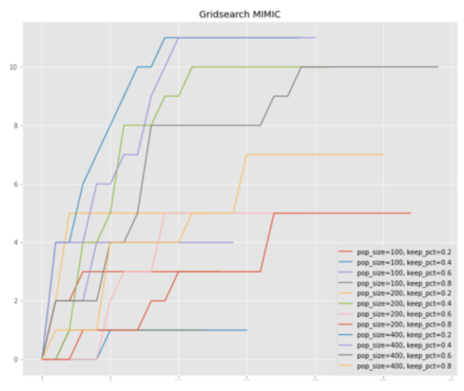
3.3.1. Grid Search



Best parameters: init_temp=10^2 & exp_const= 0.05

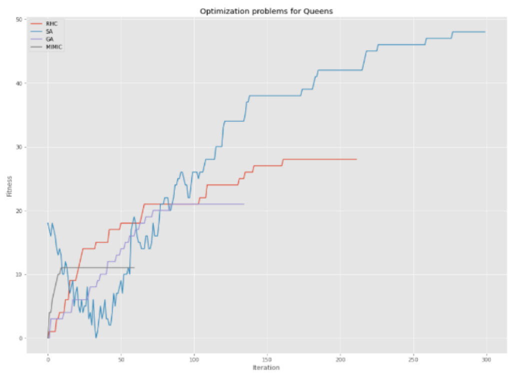


Best parameters: pop_size=200 & mutation_prob=0.2



Best parameters: pop_size=400 & keep_pct=0.2

3.3.2. Results



Algorithm	Wall clock time
SA	0.45 s
GA	25.93 s
MIMIC	82.29 s
RHC	0.23 S

- SA performs the best for N-Queens Problem, it reaches a high fitness value comparing to the other algorithms. A plausible explanation could be the large number of possible solutions for the problem, which makes the exploration easier. RHC would probably have reached a similar fitness value if it has been given more time, instead of getting stuck at local optimum, since the nature of the two algorithms is similar (based on Hill Climbing). However, SA remains the fastest to reach a global optimum thanks to the exploration.
- MIMIC finished the first in term of iterations, it looks like it also got stuck in a local optimum. This is probably caused because there is a lack of structure and a lot of possible solutions (combination of queen positions). The algorithm tries to understand the relation between the structure and the fitness value, which is not very unique.
- GA performs better than MIMIC but worse than RHC and SA. The fitness value was getting improved in the early iterations, but then the algorithm converged to a local optimum as well. It does not perform as well as SA, probably because the global optimum area is very tiny since the problem is defined with a lot of constraints (queens have high degree of liberty). Therefore, GA tends to it but is unable to access it.

3.3.3. Ways of improvements

For MIMIC, increasing the population size might help it get a more general structure and avoid local optimums. For RHC and GA, we can try increasing the maximum number of iterations and attempts, and also the mutation probability for GA.

4.Conclusion

The best algorithm depends on the nature of the problem we are facing, each one has its weaknesses and its strengths:

	RHC	SA	GA	MIMIC
Strengths	- Fast running time - Simple approach	- Exploration makes it good for exhaustive search - Avoid local optimums	- Crossover and mutation make it Good for complex problems	- Good for problems with structure - Low number of iterations
Weaknesses	- Memoryless - Can take a lot of time to converge	- Memoryless - dependent on the temperature and its decay	- Slow running time - Inefficient when tiny optimum area	- Very heavy computation - Difficulty to capture structure when many optimums