Georgia Institute of Technology

MS Computer Science

CS7641 - Machine Learning

Randomized Optimization

Author:
Alexis Durocher
(adurocher3)

Supervisors:
Prof. Charles Isbell & Prof.
Michael Littman

March 11, 2018

1 Introduction

In this assignment, the goal was to use and analyze 4 optimization algorithms. Namely, Random Hill Climbing (RHC), Stimulated Annealing (SA), Genetic Algorithm (GA) and MIMIC. They all are called randomized optimization algorithms because each of them includes a random step during their exploration and/or exploitation process.

First, we will compare the performances of each of those algorithms in computing the weights of a Neural Net trained on the Pima Dataset (used in AS1, ref. Supervised Learning). Then we will define and solve three optimization problems to highlight as much as possible the pros & cons of our different algorithms.

2 Neural Network - Pima

2.1 Dataset Pima

This first dataset contains information about Indian women and whether they are diabetic or not. The first 7 columns corresponds to characteristics (medical attributes, number of times pregnant or age) describing each instance (woman). The 8th column is the label (diabetic or not).

This dataset isn't big in volume (532 instances recorded) but has a relatively high dimension (7) and no null values (already cleaned). This classification problem is binary since there is only two classes (diabetic or not).

Features are all numerical (continuous and discrete). Both classes are not equally represented but the difference remains acceptable ($\pi_1 = 1/3$ and $\pi_2 = 2/3$). By comparing the covariance matrix of each class, we can draw some similarities. Both classes seem to have the same distribution in their space and the features are not independent respectively to the class (covariance matrices are not diagonals). Some features are especially highly variated (ex : the glu feature) in both classes ($varGlu_1 = 589.85$ and $varGlu_2 = 977.50$). Finally the mean of each class is almost equal :

$$\mu_1 = \begin{pmatrix} npreg = 2.92 \\ glu = 110.02 \\ bp = 69.91 \\ skin = 27.29 \\ bmi = 31.43 \\ ped = 0.44 \\ age = 29.22 \end{pmatrix} \quad \mu_2 = \begin{pmatrix} 4.70 \\ 143.12 \\ 74.70 \\ 32.98 \\ 35.82 \\ 0.62 \\ 36.41 \end{pmatrix}$$

Visualization is impossible on dimension higher than 3. However similar means and highly variated features in both classes makes us believe that both classes are not linearly separable. Intersection between classes seems relatively high.

2.2 Weights optimization

In this assignments, we used the ABIGAIL library to compute our algorithms. The errors/performance metrics used are the Mean Square Error (MSE) on training and testing data set (70/30 ratio) and the accuracy (percentage of correctly labeled samples, normalized between 0 and 1). Note that, because of the randomness and variance of those algorithms, we can not use a mean to do results aggregation since it will not corresponds to 'real' local/global maximas of the function we are tring to maximize. Instead we will iterate 5 times each and take the best.

Back-propagation as a reference: Before comparing the randomized optimization algorithms (ROAs), we started by tuning the NNets parameters : It's skeleton (number of layers and nodes) to define the space for our algorithms to explore and it's way of computing (number of iterations and activation function) . To do so, we used the classical back-propagation algorithm.

The optimal parameters were, 7 inputs layers, 2 hidden layers of size 60, with a 'relu' activation function and 1150 iterations. The best accuracy, given this skeleton on back-propagation was **0.748 in 9.02 secs**, which we will use in this first part as a reference (global maximum).

Note that these parameters were defined as 'optimal' for the only use of back-propagation, but they might limit the performance of other algorithms. We will see that later in this section.

A first comparison: Before trying to optimize the space and parameters of each algorithm, we first started to compare their raw performances with initial parameters, during a fixed number of iteration (1150) : A GA with an initial pop size of 50 and 20 cross-overs and 10 mutate / iterations. A SA with a cooling factor of x0.15. Note that each iteration takes very different times depending on the algorithms.

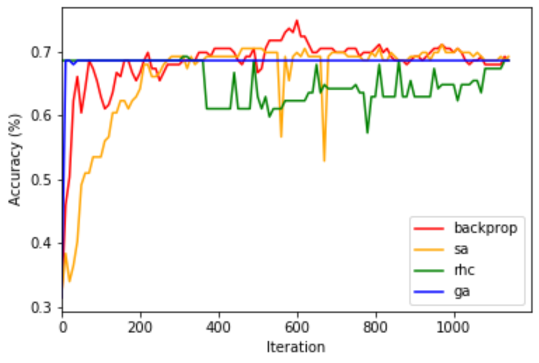

Figure 1: Comparison of each accuracy score on the test datasets by the 4 algorithms

Analysis fig 1 :

From this picture, we can notice that a local-maximum of 0.68 accuracy has attracted all the three algorithms tested. The Genetic Algorithm almost directly converged, the SA explored several other lower local maximas before converging and cooling. The RHC explored the space, finding different local maximas along the way, then it reached the same local maxima after 1000 iterations (accuracy of 0.68). **None of the ROAs did reach the 'global maximum' of 0.748.**

There are several reasons for that. The Space of Research is here relatively big due to the number of layers and nodes that we fixed and the number of iterations is small, because, in our case, the back-propagation did require a smaller amount of iteration for exploring the space and finding the best solution. Also, the initial parameters that we fixed for our ROAs restrained the exploration phase, so the algorithms couldn't find the global maximum.

A solution for improving our performance here, is to shrink the exploration space or increase the flexibility in the exploration of our algorithms. To shrink the space we could reduce the amount of layers / nodes in our NNET skeleton. Nevertheless, in this assignment, to emphasize the importance and characteristics of each parameters in our algorithms, we will keep the same NNet structure and only tweak the algorithms themselves.

Hence, the goal is to find the way to 'give more space for exploring' to our algorithms, in a fixed exploration space. According to the algorithms chosen, this 'exploration stretching' can be achieved by using different techniques.

Genetic Algorithm The Genetic Algorithm uses an initial population of random samples and iterate by creating new population using cross-overs and mutations over the previous population. Hence, the parameters here are of big importance in the exploration phase. To extend the number of possibilities, a solution is **to increase the population size**. Also, to avoid a too quick convergence a solution is to **set a higher number of mutations** to try new solution samples in the space of research at each iteration. Note that, because an iteration of GA is expensive but not much iterations are required before convergence, we will reduce the number of iteration by more than 2 (400).

To illustrate we tried different range of set-ups for population size (100 to 500) and mutation numbers (10 to 50). Then we plotted the 3 bests (cf. Fig 2)

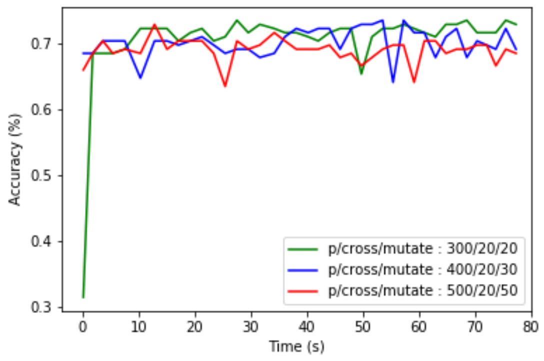

Figure 2: Accuracy of 3 GAs with different population size (p), cross-overs (cross) and mutations (mutate) on test dataset

Analysis fig 2 :

We can see that, regardless of the parameters the GA tends to quickly attain a 'plateau' at a relatively good solution (accuracies vary between 0.63 and 0.75) after 2 secs. Then the solutions are variated and do not properly converge to a maximum. The best result was obtained by the GA 400/20/30 with an accuracy of **0.735 in 53.92 seconds**. There is a trade-off to accept here : the more space you give for your exploration, the more probable GA will find great solutions, but it obviously takes more time.

Random Hill Climbing: The RHC explores the space by randomly drawing new samples and testing its neighbors. A solution to give the RHC more flexibility is to **increase the number of iteration** so it can test more local maximum and eventually reach the global maximum.

To illustrate it, we increased the number of iterations by a factor of 10 (10000).

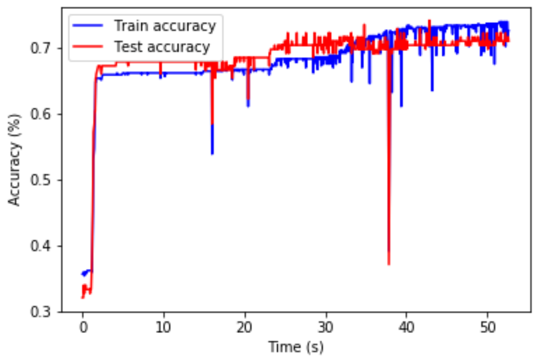

Figure 3: Accuracy of RHC function of time on test dataset

Analysis fig 3 :

Although the results are highly variated due to the random reinitialization, a great accuracy of **0.742 was reached by the RHC after 42.883 sec**. The RHC requires time to expend its space exploration. If one would accept an infinite run time, the RHC would converge to the global maximum.

Stimulated Annealing: The SA explores the space by testing a sample's neighbors and moving forward regarding a 'temperature' parameter. If the temperature is hot (high), then the SA will explore the space as a random walk. When the temperature is cold (low), the SA will act similarly to a hill climbing, by only moving to a neighbor providing a better fitness score. A solution to give the SA more flexibility is to play **on the cooling factor and number of iteration**. Indeed, the exploration space is limited to the space explored while the temperature was hot. Hence, if the SA 'cools' slowly, then the space of exploration will be less limited.

To illustrate it, we tried a range of cooling factors [x0.15, x0.35, x0.55, x0.70, x0.95] and multiplied the number of iterations by a factor of almost 3 (3300). Note that we used here the mean of 5 of each SA to give a general overview. Hence the results are lowered.

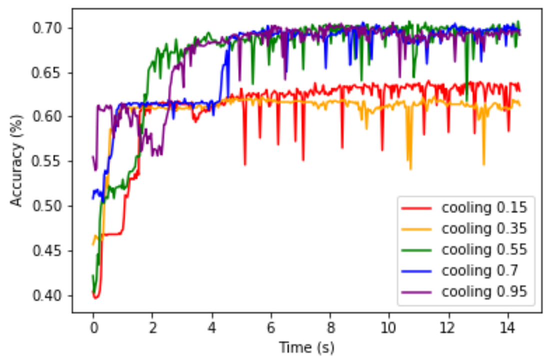

Figure 4: Accuracy comparison of SA with different cooling factor on test dataset

Analysis fig 4 :

It is clear that best results are obtained for SA with slower cooling process (from x0.55 to x0.95). The cooling factor of x0.55 seems to propose a good trade off between accuracy and run-time. It reaches a mean of 0.706 accuracy in 10.648 sec. An individual SA with x0.55 cooling factor reached an accuracy of **0.748 in 7.05 sec**.

Comparisons: In this Pima case, the Genetic algorithm proposes interesting solutions in a very small amount of iterations and hence time (0.697 after 2 seconds). However, because each iteration has a computational time which increase with the size of the population and operations (mutations /cross-overs), it can be really long to reach the best solutions if one wants to maximize the space exploration (0.735 at 53.92 seconds) .

The Random Hill Climbing is interesting in a sense that its exploration space is very large (random reinitialization). Because of this criteria, it can reach very great solutions in tiny or wide range of time (0.681 after 2 sec / 0.742 at 42.883 sec).

The Stimulated Annealing appears here to offer the best trade-off between time and accuracy. On the opposite of the RHC and GA, the SA requires a cooling phase so it can't offer almost-direct great solutions (0.53 after 2 seconds). However, by exploring the space with a relative cooling factor (x0.55 here) it offers great confidence in converging towards **a probably best solution in a limited amount of time (0.748 in 7.05 sec)**.

3 Optimization Problems

In this section, we will illustrate the 4 randomized optimization algorithms in 3 different contexts, corresponding to 3 optimization problems. Namely, N-Queen, the Traveling Salesman and the K-color.

3.1 Problem 1 - 10 Queens

The 10 queens puzzle is the problem of placing 10 chess queens on an 10×10 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

More formally, we will try to maximize the fitness evaluation function : number of non-attacking pairs. Note that, we can thus know the global maxima of this function : for n queen : (n-1)*n / 2. So here, for a n fixed to 10, the **global maxima is 45**. An 'instance' solution of our problem will be here an array (10,1) with the positions(rows) of each queen in their respective columns. Note that the iterations are multiplied by a factor of 10^3 for visualization purpose.

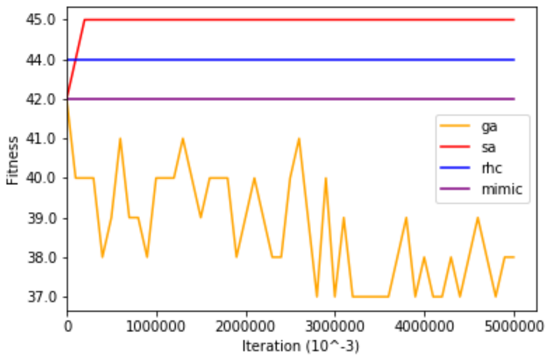

Figure 5: Fitness function of iterations - 10queen

Analysis fig 5 :

First, we can see that the **SA reached the global maximum** in very few iterations. The RHC stayed stuck in a local maxima of 44 (and it was the case for the 5 trials of this algorithms). The MIMIC acts relatively good since it reached a fitness value of 42 on the first iterations. However, the GA acted horribly and even decreased its finess value with the iterations. Why ? This illustrates a particular weakness of the GAs for certain problems where the the fitness score is very dependent of the feature ordering in the sample solutions. Indeed, the GA uses cross-overs to create offsprings, using parts of the parents. In the N-QUEEN problem, two parents with great score can give offspring with terrible scores because the position of the queens in each columns are highly dependent (the queen can attack in diagonal). Ex with 4 queens : two instance with a fitness score of 3 out of 4, [0,3,1,2] and [2,1,3,0]. A cross over using left and right parents parts would result in an offspring [0,3,3,0] with a fitness score of 0 !

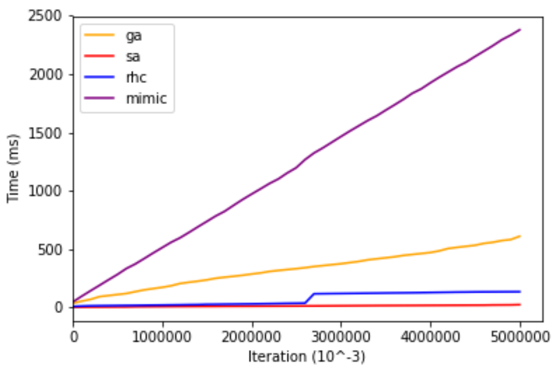

Figure 6: Times function of iterations-10queen

Analysis fig 6 :

The SA and RHC are very quick to iterate because their iteration do not depend on population size and operations over this population. Although the MIMIC proposed interesting results, the increasing computational time of each of its iterations makes it a bad concurrent. Finally the SA remains the best algorithm to solve the 10-queen problems.

3.2 Problem 2 - The Traveling Salesman

The traveling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"

More formally, we will try to maximize the fitness evaluation function : 1/distance total of the route. Note that, this problem has $\frac{1}{2}(n-1)!$ different solutions. Here we use n = 100. An 'instance' solution of our problem will be here an array of the cities corresponding to the route. Note that we multiplied the fitness score by 10^3 for visualization purpose.

Pre-analysis: On the opposite of the NQUEEN, in this Problem, the Genetic Algorithm could be an interesting candidate. Indeed, the cross-overs on two parents with short paths in their total route, could lead to global shorter routes once reunited.

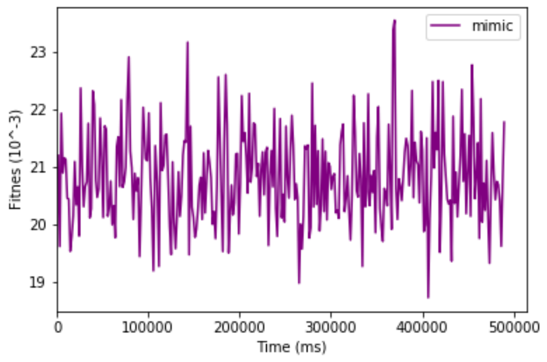

Figure 7: Mimic fitness function of times - TSP

Also, the MIMIC here offered terrible results (cf fig. 7). Hence, for visualization purpose, we won't plot the mimic results for the TSP, so we can focus on the parts of the curves where the RHC, GA and SA were performing well. Here the best score was reached by the GA with 30 cross overs and 30 mutations.

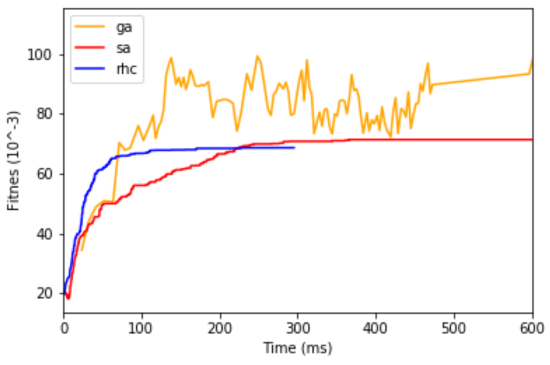

Figure 8: Fitness function of times - TSP

Post Analysis fig 8 :

Both RHC and SA did converge to a local maxima with a fitness value around 0.067. **The GA, here demonstrates very great results**. Indeed, after 100 milli secs, the GA is already better than the others and increases until almost 0.1 in fitness value after 150 milli secs. This result from GA fits the pre-analysis above.

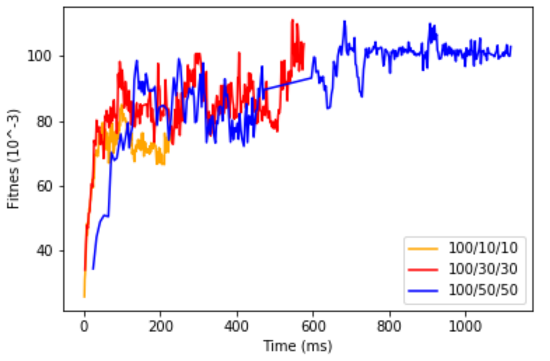

Figure 9: Different GAs function of times (p/cross/mutate)

Post Analysis fig 9 :

We can see that because of the great size of space exploration, GAs with a **higher mutation number per iteration** perform better.

3.3 Problem 3 - Two colors

The two colors problem consists in finding a list of n instance of values/color (0 or 1) such that a maximum of instances i in the list are surrounded (i-1, i+1) by an instance of different value. More formally, lets define that a solution is a list of instance with value 0 or 1. An instance with value v will be considered correct, if its left and right neighbors are of value v' (with v' different of v). We will try to maximize the fitness evaluation function : count the number of **correct instance**. Note that with n = 100 instance, we know that the global maximum is set to 100.

Pre-analysis: There is an intrinsic structure in the data of this problem. Indeed, each instance are dependent of both their left and right neighbors. It is a dependency tree that an algorithm such as MIMIC could take advantage of. The Mimic algorithm generates new population of samples by using probability distribution. It then evaluates the samples and retain only the 'tokeep' best solutions, generate new distribution and reiterate. Mimic has 2 main parameters that we can tune, the number of samples generated and the number of samples to keep after their evaluation, at each iteration.

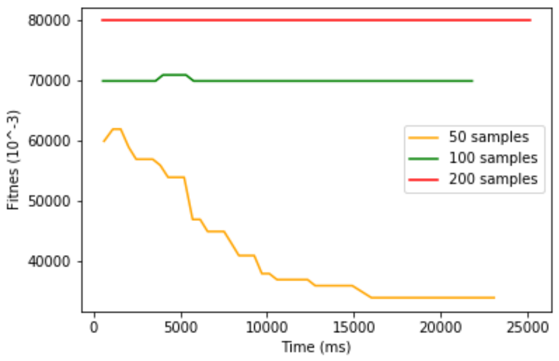

Figure 10: Comparisons of Mimic with different number of samples , function of times

Post Analysis fig 10 :

We can see that, the number of samples generated at each iteration of Mimic has a strong influence on the result. Indeed this help exploring a higher number of possibilities. The best score was given here for a mimic generating 200 samples.

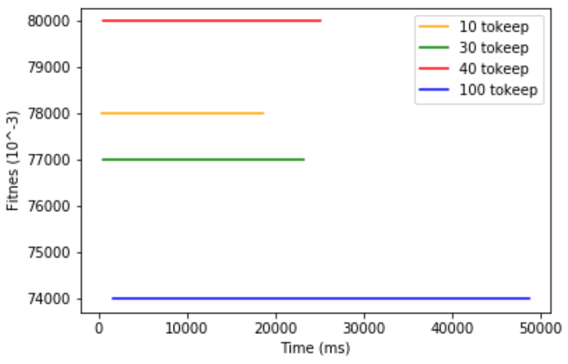

Figure 11: Comparisons of Mimic with different number of tokeep samples , function of times

Post Analysis fig 11 :

We can see that, the number of samples to keep at each iteration of Mimic also has a strong influence on the result. Strangely, it is not the mimic with the lowest 'tokeep' samples that acts the best. Indeed, the best fitness value was given for a mimic keeping 40 samples.

Using these parameters we can now compare the MIMIC with the 3 others algorithms.

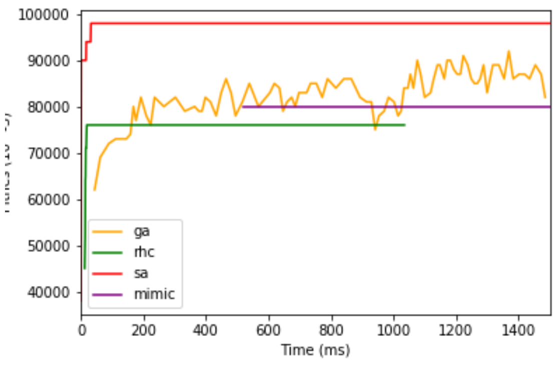

Figure 12: Fitness function of time - 2colors

Post Analysis fig 12 :

Although the MIMIC proposes interesting results because of the raisons given above, the best algorithm remains the SA. The SA almost reached the global maximum of 100 in only 20 ms. Surprisingly, the RHC performed badly. The RHC was stuck in a local maxima around 78. A reason is that it didn't manage to reinitialize out of it.

3.4 Conclusion

To conclude, this assignement emphasized the strengths and weaknesses of 4 randomized optimization algorithms.

The RHC usually performs well and quickly because it accesses the whole space thanks to its random reinitialization. However it can also get stuck in local maximas for a random range of time (two-colors).

The SA depends on cooling-factor. We have seen that, in most of the cases : it was better to use slower cooling factor so the SA could explore more before exploiting. The SA had very interesting performances in computational time because iterations are very lightweight to compute.

The GA and Mimic are algorithms which depend a lot on the structure and/or the nature of the solution.

Mimic can take advantage on structured data to propose great performances (NQUEEN, TWOCOLOR). The cost of each iteration of MIMIC is high but since this algorithm doesn't require a lot of evaluation functions and iterations, it could be useful for problems with evaluation functions that are expensive to compute.

Finally the GA depends on the solution nature. Mutations and cross-overs can be very efficient in some cases (TravelingSalesman) because few iterations can lead the GA to reach near-optimal maximas. However, such operations can also destroy links and relations that were used by the fitness function (NQUEEN).

As an opening subject of research, it could be interesting to mix some of those algorithms to keep their pros and avoid their cons. For example a mix of a GA that uses different SAs for each iteration.