Comparison of Randomized Optimization Methods

Jenny Hung

[jehung@gatech.edu](mailto:jehung@gatech.edu) (alias: jhung34@gatech.edu)

1. Optimization problems

I have chosen 3 diﬀerent optimization problems to demonstrate the various strengths of each algorithm. I picked the continuous peak problem, the Travel-Salesman-Problem (TSP) and the Flip Flop problem. Each of them and the motivation behind them will be explained in the following. Please note, that the first two functions are minimization problems, which can be easily transformed to maximization problems by changing the sign of the cost or the fitness function, so this is not a restriction.

1.1 Continuous Peaks

The Continuous Peaks problem is a variation based on the four-peaks problem, which was originally presented in [Baluja and Caruana, 1995][[1]](#footnote-1). The original four-peaks problem is defined as, given an input vector X, which is composed of N binary elements, maximize the following: Fitness is maximized if a string is able to get both the REWARD of 100 and if the length of one of head(1,X) or tail(0,X) is as large as possible. The four peaks problems also have two suboptimal local optima with fitnesses of N (independent of T). One of these is at tail(0,X)=N, head(1,X)=0 and the other is at tail(0,X)=0, head(1,X)=N. Hill-climbing will quickly get trapped in these local optima. For hill-climbing to work well here, it must repeatedly make “correct” decisions while searching large plateaus; this is extremely unlikely in practice. By increasing T, the basins of attraction surrounding the inferior local optima increase in size exponentially while the basins around the global optima decrease at the same rate.

In the Continuous Peaks version, rather than forcing 0’s and 1’s to be at opposite ends of the solution string, they are allowed to form anywhere in the string. For this problem, a reward is given when there are greater than T contiguous bits set to 0, and greater than T contiguous bits set to 1.

In solving for the Continuous Peaks problem, the existing cost function of the JAVA library, ABAGAIL, was used. Each algorithm was run using 20000 iterations to observe how quickly the algorithms converge on the optima. We observed that different algorithms have wildly different time complexity. For example, MIMIC is highly computationally intensive, primarily because the additional time it takes to build structure while learning enables the algorithm to learn with fewer iterations.

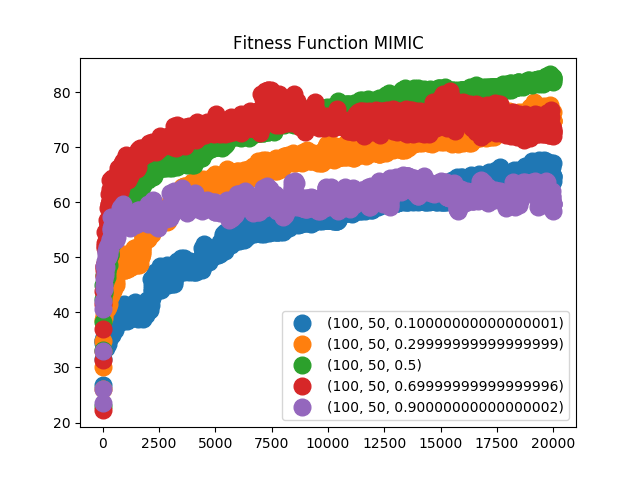
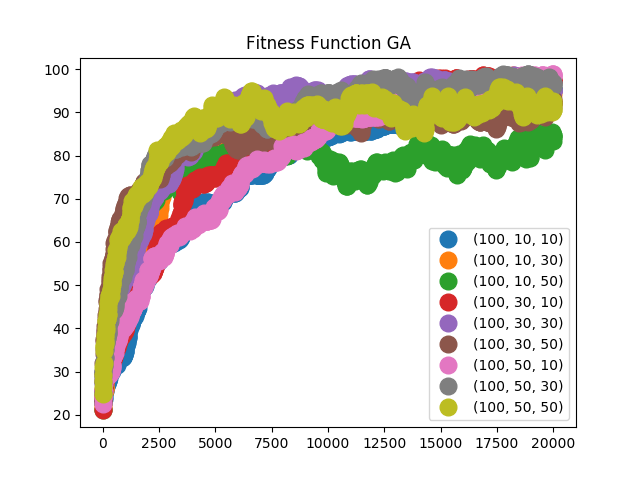
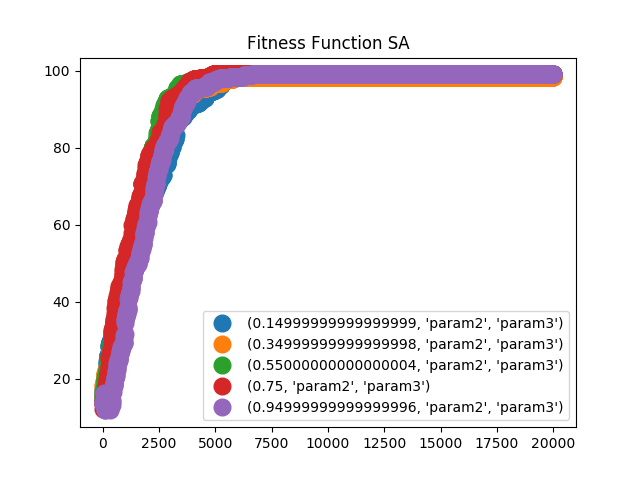
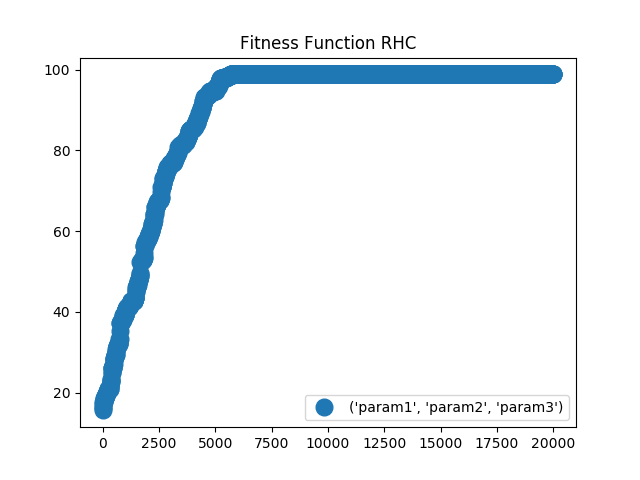
Each of these tests were run 5 trials and the average function value is reported. The fitness score and running time were reported every 10 iterations.

The default parameters in ABAGAIL were as follows:

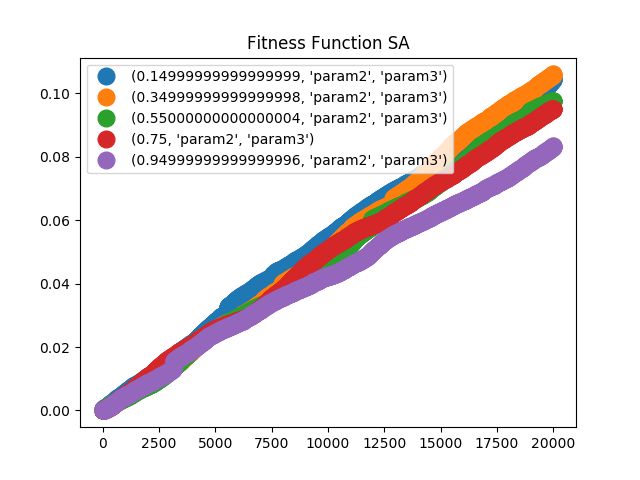
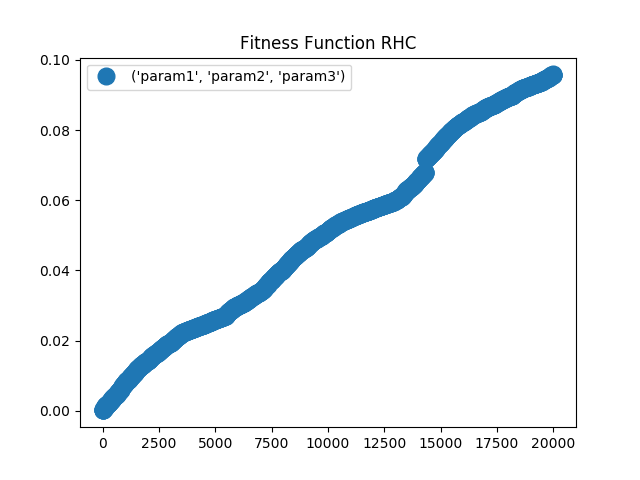
* Simulated Annealing: temperature = 1E11, cooling rate = [0.15, 0.35, 0.55, 0.75, 0.95]
* Genetic algorithm: population size = 100, toMate (varying value) = [50, 30, 10], toMutate (varying value) = [50, 30, 10]
* MIMIC: sample size = 100, keep = 50, m (varying value) = [0.1, 0.3, 0.5, 0,7, 0.9]

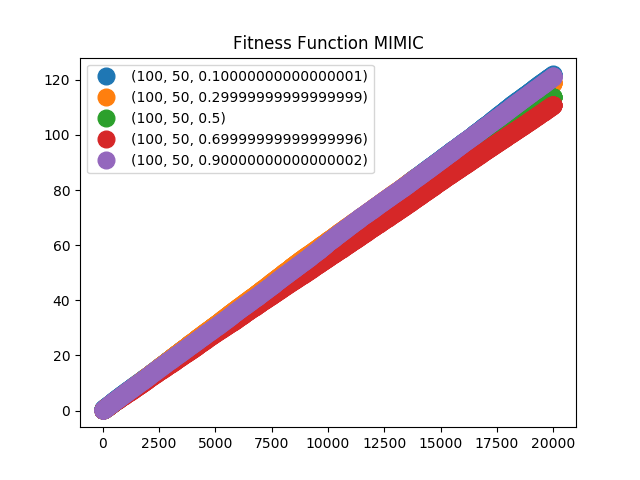
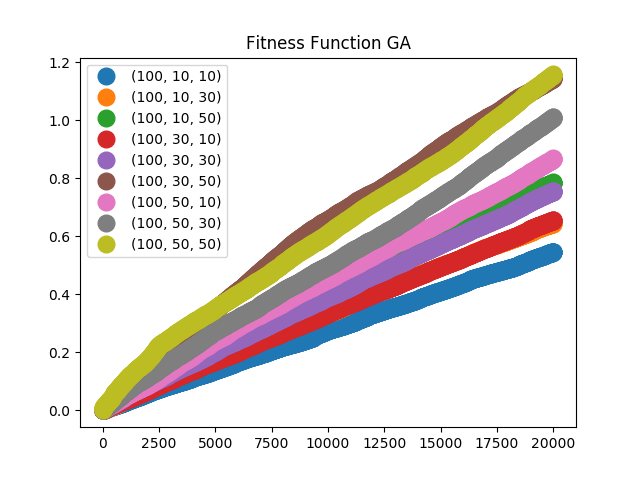
This problem set contains many local optima in a 1D space, similar to the example in lectures about determining the elevation of many peaks. Although this problem set was chosen for its simplicity, it highlights differences between random optimization algorithms well and applies to other examples like topography and optimization of surfaces.

The outcomes of fitness functions were plotted against number of iterations below.



Running time were plotted against number of iterations below.





Clearly SA and RHC outperform the other algorithms, both achieved nearly no error – SA had achieved this with the least number of iterations (approx. 3000), whereas RHC achieved this at approx. 5000 iterations. Further, GA also performs better than MIMIC, which gives the lowest fitness across all trials. MIMIC is highly computationally intensive – but what is interesting about MIMIC is that, it tends to achieve higher fitness much earlier on, compared to all other algorithms considered. This is primarily because it takes the time to build the probability structure, which enables the algorithm to learn faster, given the additional information (that does not exist in other algorithms).

Similarly, SA and RHC have very similar wall time when it comes to algorithm execution. Depending on the toMate and toMutate parameters, GA’s running time varies – it could be in the range of SA/RHC but it did progressive get worse when the number of computations gets larger given the parameters, which is consistent with our understanding. Of all the algorithms, MIMIC took the longest and its running time in terms of wall time were an order of magnitude larger compared to others. MIMIC is highly computationally intensive, primarily because the additional time it takes to build structure while learning enables the algorithm to learn with fewer iterations.

* 1. TSP

The well-known Traveling Salesman Problem is proven to be NP-hard. The goal is to find the shortest round-trip between N cities while visiting each city just once. TSP problems also occur in everyday life. Planning optimal routes between cities is a crucial task for logistic companies or a backpacker who wants to visit some landmarks. Although in this cases N may be rather small. Other applications may include factory scheduling, wiring looms and circuit board drilling. In general, apart from the fact that the number of possible routes grow exponentially with respect to the number of cities, the TSP problem is not ’discreet continuous’[2](#page2). That means that small changes in the configuration of the points can lead to completely diﬀerent optimal routes. That makes it hard to define a good search algorithm, and a greedy algorithm is very unlikely to find the optimal route.

Each of these tests were run 5 trials and the average function value is reported. The fitness score and running time were reported every 10 iterations.

In solving the Traveling Salesman’s problem, the existing cost function of the JAVA library, ABAGAIL, was used. Each algorithm was run using 20000 iterations to observe how quickly the algorithms converge on the optima.

the The default parameters in ABAGAIL were as follows:

* Simulated Annealing: temperature = 1E11, cooling rate = [0.15, 0.35, 0.55, 0.75, 0.95]
* Genetic algorithm: population size = 100, toMate (varying value) = [50, 30, 10], toMutate (varying value) = [50, 30, 10]
* MIMIC: sample size = 100, keep = 50, m (varying value) = [0.1, 0.3, 0.5, 0,7, 0.9]

1.3 Flip Flops

Flip-flops can be either simple (transparent or asynchronous) or clocked (synchronous). The simple ones are commonly described as *latches*,[[1]](https://en.wikipedia.org/wiki/Flip-flop_(electronics)#cite_note-pedroni-1) while the clocked ones are described as *flip-flops*.[[2]](https://en.wikipedia.org/wiki/Flip-flop_(electronics)#cite_note-ee42-2)

Simple flip-flops can be built around a single pair of cross-coupled inverting elements: [vacuum tubes](https://en.wikipedia.org/wiki/Vacuum_tube), [bipolar transistors](https://en.wikipedia.org/wiki/Bipolar_transistor), [field effect transistors](https://en.wikipedia.org/wiki/Field_effect_transistor), [inverters](https://en.wikipedia.org/wiki/Inverter_(logic_gate)), and inverting [logic gates](https://en.wikipedia.org/wiki/Logic_gate) have all been used in practical circuits.

Clocked devices are specially designed for synchronous systems; such devices ignore their inputs except at the transition of a dedicated clock signal (known as clocking, pulsing, or strobing). Clocking causes the flip-flop either to change or to retain its output signal based upon the values of the input signals at the transition. Some flip-flops change output on the rising [edge](https://en.wikipedia.org/wiki/Signal_edge) of the clock, others on the falling edge.

Since the elementary amplifying stages are inverting, two stages can be connected in succession (as a cascade) to form the needed non-inverting amplifier. In this configuration, each amplifier may be considered as an active inverting feedback network for the other inverting amplifier. Thus the two stages are connected in a non-inverting loop although the circuit diagram is usually drawn as a symmetric cross-coupled pair (both the [drawings](https://en.wikipedia.org/wiki/File:Eccles-Jordan_trigger_circuit_flip-flip_drawings.png) are initially introduced in the Eccles–Jordan patent).

2 Optimization Algorithms

The randomized optimization algorithms under comparison are:

* Randomized Hill-Climbing (RHC): locates local optima by moving towards more optimal neighbors until it reaches a peak. With random restarts, RHC randomizes its starting position to locate other local optima, and selects the value with the highest value as the global optimum. RHC was performed using RandomizedHillClimbing() in ABAGAIL on Java.
* Genetic Algorithm (GA): inspired by biology in which the population evolves by iteratively mating and mutating parts to crossover the best traits and to eliminate irrelevant traits. A significant disadvantage of GA is that it does not handle a large hypothesis space, which is dictated exponentially by the number of attributes. SA was performed using StandardGeneticAlgorithm() in ABAGAIL on Java
* Simulated annealing (SA): originates from metallurgy where the ductility in metals are improved by heating to a higher temperature (below melting, above the annealing temperature where residual structural stresses are relieved) and then slowly cooled to maintain its structure. The algorithm, a function of initial temperature and cooling rate, strikes a balance between exploring new points and exploiting nearby neighbors in search of local optima. Initially, at high temperatures, the algorithm explores by randomly seeking new points and as it cools, it proceeds to evaluate neighbors for local peaks. SA was performed using SimulatedAnnealing() in ABAGAIL on Java

Simulated annealing was found to be sensitive to the initial temperature and the perturbation functions used. An exhaustive search for the best parameters like for SVMs could be used, but would take a significant amount of time and was considered to be beyond the scope of the assignment. The values were adapted by try and error. It has been found that not the absolute values for initial temperature and the amount of perturbation were significant but their ratio.

* MIMIC: MIMIC algorithm, as opposed to most optimization algorithms, “remembers” previous iterations and uses probability densities to build structure of the solution space and find optima. MIMIC was performed using MIMIC in ABAGAIL on Java.

Because all algorithms are randomly initialized, running an algorithm once on the data and try to derive conclusions about the nature of the algorithm is not suited. Instead all algorithms were run several times and the mean was used for comparison. The specific iterations are given in the results.

3 Results

This section gives details about the parameters used for optimization as well as the results of the diﬀerent algorithms.

3.2 TSP

All algorithms were run 10 times, except MIMIC which was run 5 times. In addition the GA was run for varying population sizes varying from 5 to 50. The average results are given in in table [1.](#page5) The error is the distance to the true minimal function value given by the optimal tour. A tour was represented by the permutation that one obtain when sorting a vector. This enables to use ordinary addition and cross-over operators, while calculating the neighbor or the cross-over of two permutations is not trivial. One could argue, that this may lead to bad results, but it has found in a separate test, that substituting the neighbor operation by swapping two cities does not improve the optimization results. One can think of many definitions of a neighbor of a permutation and they are all equally well. Clearly all algorithms perform not very good, the route they compute is in average 50 % too long. Like with Rastrigin’s function GA improves with increasing population size, therefore the charts are omitted. Unlike with Rastrigin’s function the TSP problem is not smooth and hard to analyze. It seems, that due to its ’discontinuities’ [4](#page6) MIMIC and GA perform bad. Interpolation among them may fool the algorithms. It should be noted, that there exists TSP-optimized variants that implement more sophisticated cross-over and mutation operations. In my representation a mutation might not change the route at all. RHC and SA perform equally well, although this time the complete poll strategy for RHC leads to a 1000 mi better route in average. I think both algorithms perform pretty good, cause they explore a huge search space and are likely to find good routes. The best obtained route for SA and GA are given in figure [6.](#page7) Although RHC performs equally well to SA, it is way slower. Also RHC expands its search radius, it stays in its current neighborhood and searches within it. The restart from a diﬀerent random point may change this. SA evaluates not only points in the current neighborhood but also other points. That might be why it finds a ’good’ route faster than RHC.

3.3 One-Max

The optimization of One-Max was done with the ABIGAIL package and the average results over 100 runs (MIMIC 10 runs) are given in table ??. n is set to 80, so the maximum should be obtained at 80.

In this case SA performs worst, although it evaluates many points. The reason is that one-max relies on a search scheme that interpolates evaluated points, that is why MIMIC excels all other optimization algorithms, which only check neighbors. As can be seen in figure [3](#page3) this will often lead to local minima or worse configurations. I think MIMIC most probably performs better that the other algorithms, cause One-Max is ’discrete continuous’, so an interpolation will seriously guide the search towards the optimum value.

4. The neural network

I used the neural network from assignment one to recognize cancer cells from features extracted from microscopic images. I showed that a network with 1 hidden layer of 7 units can achieve classification rates of 97.8 %. I used the standard minimization of the sum of squared distances (SSD) of the error with regularization term to prevent the solution from blow-up, that might be occur during RHC. The regularization term is just the addition of the norm of weights of the units. With a 20 dimensional input, that leads to a 20 ∗ 7 + 7 ∗ 1 = 147 dimensional problem. The weights were initialized randomly and each optimization algorithm was run 10 times. The mean results are given in table ??. The recognition rate was obtained by applying the network to a separate test set. There are some interesting results to notice. First the function value we try to minimize does not lead necessarily to a good recognition rate. This may have two reasons. First the introduction of the regularization term and second overfitting may occur, when try to minimize the SSD without any cross-validation set. It is also interesting that the GA performs best in this case. It might be that mutation and especially cross over model structural changes of connections of the layers and therefore lead to very good results. As usual RHC needs the most function evaluation, followed by GA and SA.

8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm | optim. time s | fun-evals | average fun-value | best value | best recogn. rate |
|  |  |  |  |  |  |
| RHC - first poll | 1290.96 | 5034.2 | 7.3685 | 6.6078 | 0.76842 |
| GA | 916.02 | 3420 | 14.7817 | 10.6755 | 0.87895 |
| SA | 498.84 | 3958 | 10.5211 | 9.3258 | 0.71053 |
|  |  |  |  |  |  |

Table 4: Neural Network Training Results

5. Conclusion

It could be shown that smooth problems can be solved best with SA (Rastrigin’s function and neural network training), while ’discreet smooth’ problems like One-Max are best suited for MIMIC. RHC is an exhaustive search which performs very well on smooth problems, but has serious diﬃculties with discontinuities. All 4 algorithms does not perform well on problems like TSPs, where special variants of them are needed to obtain good results. If we have a smooth problem with a global minima, like in the case of minimizing the SSD in case of the NN, none of the algorithms can outperform an ordinary Gradient descent method.

9

1. Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space [↑](#footnote-ref-1)