Assignment 2

Sai Chen

**Section 1. Introduction**

In this analysis, I used three optimization problems to illustrate the relative strength and drawbacks of four random-search based optimization algorithms: random hill climbing, simulated annealing, genetic algorithm and Mutual-Information-Maximum Input Clustering (MIMIC). Also, I used the first three algorithms to train the neural network for classifying handwritten digits from assignment 1 and compared the results from each algorithm as well as to the results from the backpropagation algorithm. Below I will describe and discuss my findings. In section 1, I will introduce the datasets used in this analysis and some technical details about the implementations of each algorithm. In section 2 I will evaluate the performance of each algorithm using three different optimization problems. The relevant conclusion will be presented and discussed. In section3 I will show the results of applying the algorithms to train the neural network.

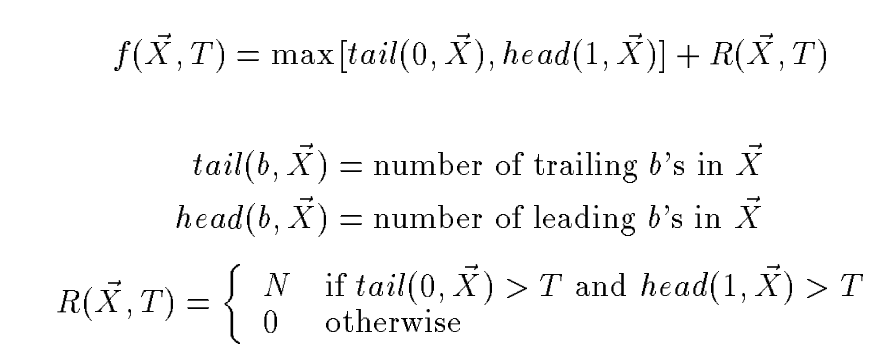
***Section 1.1. Datasets: optimization problems***

I chose three different optimization datasets to compare the four different optimization algorithms: Count-One problem, Four-Peaks problem and Knapsack problem. In general, the four different algorithms have various performance based on the nature of the target problem.

**Random hill climbing** and **Simulated Annealing** are better suited for problems with a single global optimum. Also, they perform better if the underlying optimization problem has no dependent structures among the optimizing variables. By having a well-designed cooling scheme, simulated annealing can sometime escape from the local minimum and have better performance than simple random hill climbing. Also, both algorithms are simple to implement, and the time cost is trivial compared to the other two. In this analysis, I implement the simplest possible algorithm for both algorithms. Multiple random initial states or greedy (steepest) neighbor searching are not implemented. The Count-One problem is borrowed from the ABAGAIL1 documentation. The target function is the count the number of 1’s in a binary array. For an array with length N, the single best solution should be obviously N. There is no dependent structure among the variables for this problem, so I expect that random hill climbing and simulated annealing would do better for this problem.

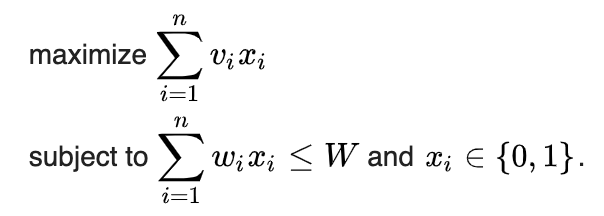
**Genetic Algorithm** and **MIMIC** utilizes a population of potential solutions, mutation operator and crossover operator to work around the weakness of the first two random search algorithm: there is no look back along the search path so that it is very easy for both algorithms to be stuck at local minimum. In another word, single-neighbor-based methods can only do well in ‘exploitation’ while population-based methods can also search for candidate solutions to refine, namely ‘exploration’. Genetic algorithm and MIMIC differ in how they use the population to explore the structure of the problem. Genetic algorithm makes use of the mutation and crossover operators to recombine the refined population in each generation, hoping that the global optimum is attainable by reorganizing several less-optimum solutions. On the other hand, MIMIC has a clearer probabilistic framework in which the algorithm learns an approximate representation of the underlying distribution of a smaller (with better performances) region in the solution space and uses that to sample the population in each generation, hoping that the best answer could sweep through the population in later runs. MIMIC might have better performance than simple genetic algorithm in problems with strong dependent structures for the final solution.

The Four Peaks problem is a constructed optimization problem2 useful for evaluating performance of algorithms. The target function is described below. There



are two global maximums and two local maximums. I expect that random hill algorithm will stuck in the two local maximums most of the time because of the large attraction basins around them. Simulated annealing might be able to escape in small-sized problems but won’t get luck in large-sized ones. It is not clear to me which of the two population-based algorithms can do better for this problem. However, it is clear that with some careful designs genetic algorithm runs much faster than MIMIC. That could an advantage of the genetic algorithm.

The Knapsack problem is a well-known NP-hard combinatory optimization problem. Given a list of items with values and weights, the 0-1 Knapsack problem tries to maximum the total value under a constraint of the total weight. The target function is illustrated below3. In this analysis, I implement the simplest Knapsack problem with



integer values and weights. This problem could be exactly solved by dynamic algorithm. This recursive nature along the sub-problems makes me think that MIMIC could do better by utilizing this structure. Thus, I expect MIMIC will be able to give the best answer with more and more complicated problems.

***Section 1.2.*** ***Digits recognition problem***

Digits recognition problem is directly imported from sklearn library. Each training sample is an 8x8 image representing one digit from 0~9. There are 1797 samples in the dataset, ~180 per class. The pixel values range from 0 to 16 in integer form.

***Section 1.3.*** ***Implementation details***

All algorithms are implemented by me without using other libraries (python (>3.6.8) and numpy (>1.11)). This might explain the slow speed of my analysis.

For random hill climbing and simulated annealing, neighbor is searched by randomly flipping one of the bits in the bit array for discrete problems. For training neural network, neighbor is searched by randomly adding uniform distributed noises on all parameters independently and simultaneously. Cooling function in simulated annealing is in the simplest form in which temperature is decreased by repeatedly multiplying with a cooling rate.

For genetic algorithm, mutation operator is implemented very similar to the neighbor function described above, with an additional parameter controlling the chance of whether the mutation operation occurs. Crossover operation is implemented to occur uniformly along the chromosome, with a parameter controlling how often the crossover event occurs on a single spot. A tournament process is also implemented so that the two parents and the two children after a crossover event are all kept and compete with each other to select the best two individuals that can go into the next generation.

MIMIC algorithm is implemented as in the original paper4. The implementation is only suitable for bit-string based problems. I have tried more complicated combinatory optimization problems, such as Traveling Salesman problem and K-color problem. However, it is not easy to code them as bit string problem and the time cost for the general discrete MIMIC algorithm is much larger than simple binary case. This is one of the reasons I chose Knapsack problem in the final list.

**Section 2. Comparisons of the algorithms for the 3 bit-array based problems**

In this section I will compare the performance of the 4 algorithms from three perspectives: final solution found, number of iterations and time cost. Due to the limited computational resources I can have access to, I have restricted the number of iterations for the first three algorithms to 10,000 runs, while only 1,000 runs for MIMICs in some of the problems. Although it might be an unfair comparison, from the results I can see some clear patterns that are suitable to illustrate the basic ideas. I will discuss the effects of this restricted run time in each piece of the analysis.

***Section 2.1. Count-One problem***

Before applying the four algorithms to the Count-Ones problem, I have to decide what hyper-parameter values to use for each algorithm. As described in the beginning of this section, evaluating some algorithms is extremely expensive in computation. The importance of selecting the right parameters for each algorithm will be illustrated and discussed in the next subsection. However, I could not do parameter-tuning for even the simplest problem with reasonable replicate numbers to reduce random noise. Thus, I chose parameter values from prior experience and guidelines. This might make the comparison a little unfair, but I think the basic conclusion could still be illustrated.

The parameters used in this section is described in the table below. The size of the problem is the length of the bit array.

|  |  |
| --- | --- |
| Algorithm | Relevant parameters |
| Hill Climbing | None |
| Simulated Annealing | T = 1, cooling rate = 0.95 |
| Genetic Algorithm | mutation = 0.4, crossover = 0.6,  popsize = 4\*size of the problem |
| MIMIC | popsize = 4\*size of the problem  percentile = 0.75 |

|  |  |  |
| --- | --- | --- |
|  |  |  |

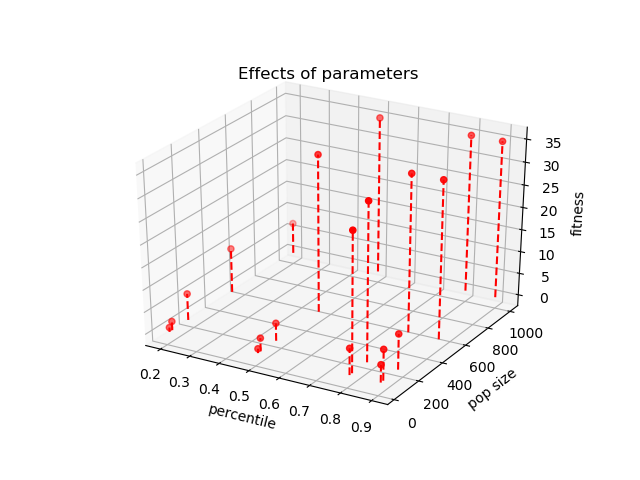
The above figures show the results with varying sizes of the Count-Ones problem (20, 30, 40, 50, 60 respectively). From left to right are final fitness, number of iterations and total time cost v.s. problem size. Confidence regions are calculated based on one standard deviation from the mean over 5 runs of each algorithm. As we can see, hill climbing and simulate annealing (SA in short below) can both reach the optimum in each of the 5 parallel runs. More importantly, the number of iterations they used are trivial compared to other two algorithms, as well as the time cost.

The MIMIC algorithm is doing fine for small problems but cannot reach the optimum for larger ones. I notice that the time cost for MIMIC is much larger than the other three algorithms. Even though I put a lot of efforts in optimizing the numerical performance of the algorithm, it is still possible that this drastic difference in time cost is due to my implementation. However, the time complexity of the MIMIC algorithm is O(n2) or O(mn), where n is the dimension of the data and m the sample size. The result, in some level, illustrates the large time cost of MIMIC if it cannot find the optimum faster than other algorithms. It should be noted that I only allowed 1,000 iterations for the MIMIC algorithm so that it is possible for MIMIC to find the best answer after more iterations.

On the other hand, genetic algorithm (GA in short below). Has a poor performance. After 10,000 iterations it still cannot reach the optimum for almost all problems with varying sizes.

This problem best illustrates that when the problem is structureless, GA and MIMIC cannot perform very well over simple random search methods. The path towards the optimum in this problem is very simple structured: one just needs to flip each 0 to 1. There is no need to look back. Thus, simple random search can do much better in this simple case.

***Section 2.2. Four-Peaks problem***

There are two parameters in the Four-Peaks problem itself: the threshold (T) and the dimension (N). What I noticed during analysis is that there is a tradeoff in choosing the ratio of these two values. If N/T is too large, then SA can easily get to one of the global optimum due to its annealing process. However, when N/T is too small, the time cost for MIMIC increases drastically. For example, to find the global optimum for N =30 and T = 10, it takes a whole day to run (over 20,000 calls) MIMIC. After some tweaking, I decided to use N/T = 5 so that the final results can best illustrate the concept while keep the running time reasonable.

Not only the problem itself affects the outcome, the hyper-parameters are also important in evaluating this problem. Figure on the left shows the result of running MIMIC over a problem with N = 20 (remember N/T = 5) with different parameter combinations. The two parameters I varied are percentage of points to exclude before building the distribution and the size of the sample generated for each iteration. All runs are stopped after 500 iterations if the right answer (in this case, 35) has not been reached. We can see that the final fitness value can be largely improved by generating more samples in each iteration. However, building the distribution by better fit samples might not always lead to better results, as can be shown by the bell-shaped fitness distribution along the population size of 160. Even though keeping more samples makes convergence much slower, throw away too many samples might trap the algorithm in a local minimum by randomness, and this is especially true with small sample size. To the most extreme case, if we only keep the best sample in each run, then the algorithm will be stuck at the point forever. Based on this, I took a trial-and-error approach to find reasonable parameters for the further analysis. And the table below show the parameters used in this analysis. I did not change the parameter values based on problem size to reduce the computational cost for larger problems.

|  |  |
| --- | --- |
| Algorithm | Relevant parameters |
| Hill Climbing | None |
| Simulated Annealing | T = 10, cooling rate = 0.95 |
| Genetic Algorithm | mutation = 0.4, crossover = 0.6,  popsize = 500 |
| MIMIC | popsize = 1000, percentile = 0.8 |

|  |  |  |
| --- | --- | --- |
|  |  |  |

Figures above show the result. GA and MIMIC can always reach the global optimum. Due to its annealing process, SA can reach the optimum for smaller problems but is stuck at local minimum most of the time for larger problems. As expected, simple random hill climbing is always stuck at the local minimum (all zeros or all ones) unless it gets luck when the initial state is close to one of the global optimums.

Surprisingly, it takes MIMIC almost no time to reach the global optimum. On the other hand, though the time cost of GA is lower than MIMIC for smaller problems, it performs worse than MIMIC for the largest problem with regard to the time cost.

Overall, when the problem requires large potential of exploration over the solution space, GA is superior over random search methods. MIMIC has a relatively similar performance to GA in this analysis. However, as I mentioned before, when the N/T ratio is too small, MIMIC takes a much longer time to converge than GA. In summary, GA has a good balance between rate of convergence and finding the right optimum for this problem.

One thing to note is that this conclusion is different from the original MIMIC paper. Besides my ‘cheating’ choice of problem sets and computational resources, the major difference is that I used a slightly greedy GA implementation in which I kept the best two individuals in each crossover event. This is a well-known technique to speed up the convergence rate.

***Section 2.3. Knapsack problem***

In this problem, I randomly generated item values and weights in integers as well as the weight constraint. The number of items varied, similar to the previous two subsections. The exact solution is found by a dynamic algorithm. The first three algorithms are given 10,000 maximum iterations and this number is 1,000 for MIMIC. Once a single run reaches the optimum the run will be stopped, otherwise it will be stopped after the maximum iterations. Table below shows the parameters I used in this analysis. The reason for lack of parameter tuning has been discussed above.

|  |  |
| --- | --- |
| Algorithm | Relevant parameters |
| Hill Climbing | None |
| Simulated Annealing | T = 10, cooling rate = 0.95 |
| Genetic Algorithm | mutation = 0.4, crossover = 0.6,  popsize = 4 \* #items |
| MIMIC | popsize = 4 \* #items, percentile = 0.75 |

|  |  |  |
| --- | --- | --- |
|  |  |  |

Figures above show the results. In this analysis, none of the methods can find the best answer in the given number of iterations due to the limited computational power. However, it is clear to see that on average, MIMIC and GA perform better than random search. The large variation in sample size 50 for MIMIC might be caused by my ill-behaved fitness function. The fitness value is zero if the total weight is above the weight constraint. It is highly possible that with a small sample size and large problem size, the population is mainly composed of samples with 0 fitness, if the randomly generated weight constraint is too small. This will affect the MIMIC algorithm, since in the current implementation, no noise will be added to the samples for building the distribution besides the randomness in the samples themselves.

One should note that in this analysis, MIMIC was only allowed to run 1,000 iterations. It is highly possible that with longer runs, the MIMIC can find the best answer.

In summary, when the problem has recursive structure in the solution space (in this case, the fact of using dynamic programming to solve Knapsack problem), MIMIC can do better than other algorithms by explicitly model the dependent structure among variables.

**Section 3. Neural Network training for Digits Recognition**

In this section I will show the result of training neural network using first 3 optimization algorithms. I will compare the results to that of backpropagation.

In assignment 1 I used SGD to search for the optimal number of hidden units for a one-hidden-layer perceptron. I found out that 15 to 20 hidden units were enough for a good classification performance over validation sets. I this analysis I used 15 nodes in the network training.

|  |  |
| --- | --- |
|  |  |
|  |  |

Figure above show the results. In each plot, training errors and validation errors are plotted against number of iterations for each algorithm. Validation sets are generated by 4-fold CV approach. Each algorithm used the same 4-fold validation sets across the training. The confidence regions are calculated as one standard deviation from the mean using the 4-fold datasets. For each algorithm, the maximum number of iterations is 1,000,000, and the error values are recorded by 10 iterations. I did not use any batch method in backpropagation. I used the negative cross entropy loss as the fitness function and plotted the classification accuracy based on the maximum logit values from the classifier.

From the figures we can see that gradient based methods are much better than random search methods with limited number of runs. Considering the large dimension of the parameter space (8\*8\*15 + 15\*10 + 15 + 10), this result is not surprising. Among the three random search algorithms, GA on average performs the best in 1,000,000 runs. However, it should be noted that random search algorithms might be able to do better when given longer time. This is especially clear for SA since the validation accuracy is still climbing up for validation sets.

The other important result is that backpropagation converges much faster than the other three algorithms. Guided by the gradient, the SGD optimizer can move towards a local optimum more efficiently than other algorithms. However, it is not guaranteed that gradient based methods can reach the global optimum. In those cases, GA might be able to do better due to its strong exploration potential.

The poor performance of the three random search algorithms might be due to the particular implementations I used in this analysis. To search for a neighbor or mutate a particular individual, a random noise is added to the current individual by sampling from a scaled uniform distribution (scale \* [-1, 1]). Since I used the same scale over the entire searching process, it is possible that random search algorithms might be stuck at some ‘good’ solutions if the scale is too small or skip the best solution if the scale is too large. It is then possible to design a scheme in which the scale parameter can be adjusted based on the progress of the training.

Also, I did not perform any parameter tuning for those algorithms. For SA, it is suggested that a careful choice of cooling schedule should be performed in continuous problems. With a better tuning one might be able to get better results.

In summary, my analysis suggests that gradient based method might perform better in high-dimensional parameter searching than random-search based algorithm.

**Section 4. Conclusion**

In this analysis,

Reference

1. https://github.com/pushkar/ABAGAIL
2. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. *Technical report*, Carnegie Mellon University, May 1995.
3. <https://en.wikipedia.org/wiki/Knapsack_problem>
4. Bonet, Jeremy S. De, Isbell, Charles L., Viola, Paul. MIMIC: Finding Optima by Estimating Probability Densities". *Advances in Neural Information Processing Systems*: 424. January 1996