A Comparative Analysis of Randomized Optimization Algorithms

Michael Walton
Department of Computer Science, Georgia Institute of Technology

October 20, 2015

Abstract

In this work we compare the dynamics and performance of four randomized optimization algorithms. The capabilities and limitations of random hill climbing (RHC), simulated annealing (SA), genetic algorithms (GA) and Mutual Information Maximization for Input Clustering (MIMIC) are explored. Local optimization methods are applied to finding optimal weights for neural network with respect to a digit recognition task. All four algorithms are then applied to three well known optimization problems. {k-colors, four peaks, N-Queens} We conclude with general analysis comparing the strengths and weaknesses of these methods.

1 Introduction

Optimization is a diverse research domain in which a plethora of different algorithms have been explored to resolve problems with unique complexities and applications. The area with which this paper is primarily concerned is local random search methods; these algorithms tend to be targeted at problems characterized by non-convex optimization surfaces in which there may be many local minima with unknown distribution in the space. This difficulty leads to the possibility that local search methods will discover a local optimum, but may get 'stuck' before reaching a global optimum (such as a naive implementation of hill climbing in which the area around the current optimum is sampled and steps are taken in the direction of maximal increase until a convergence criteria). Different methods handle this issue in different ways, but generally the high-level concept is to use iterative stochastic sampling to cover a larger volume of the parameter space.

The randomized optimization algorithms considered in the present study are random hill climbing (RHC), simulated annealing (SA), genetic algorithms (GA) and Mutual Information Maximization for Input Clustering (MIMIC). Implementations are based on the ABAGAIL java library [5] The

python modules `scipy, numpy,` and `scikit-learn` [3] [4] are used extensively throughout the implementation for experiment scripting and analysis.

2 Methods

In the sections that follow, we will provide a brief overview of the digit classification dataset and the three optimization problems used in this study. In addition, we will briefly discuss the implementation details for the considered algorithms.

2.1 Digit classification Dataset

The first dataset considered in this study is a collection of 8 x 8p gray-scale images of handwritten digits. Each input instance $x \in X$ is projected, or 'flattened' from $x \in \mathbb{R}^{8\times8}$ into a 64-dimensional vector $\hat{x} \in \mathbb{R}^{64}$ the new feature array $\hat{X} \in \mathbb{R}^{64\times m}$ where $m = 1796$ is the number of instances in the dataset; target labels valued $\{0, 1...9\}$ are one-hot encoded as an array of bit strings $\boldsymbol{y}$ where, given a target label n the n^{th} bit of the corresponding string is 1 and 0 for all other bit positions. Our objective is to derive a hypothesis $h \in H, h : x \in \mathbb{R}^{64} \mapsto y \in \{0, 1...9\}$ approximating the target concept $c(x)$. We define the preferred hypothesis as the h which minimizes $error_{\mathbb{D}}(h)$ on the actual distribution $\mathbb{D}$ by learning on the training set $S \subset D \sim \mathbb{D}$ where D is the available dataset of (instance, label) pairs $\{(x_i, y_i)\}$ The objective is some tbd. 'best' function which partitions the instance space by mapping each example $x \in X$ to a corresponding bitstring $y \in \boldsymbol{y}$ where the n^{th} bit of y indicates the target label index.

In the previous study we used two Parkinson's disease dysphonia datasets to illustrate the capabilities and limitations of several supervised learning algorithms. While this was demonstrative of a possible real-world application of these methods on a non-trivial dataset, the study suffered from a lack of understanding of the problem domain and input feature space. We have elected to use the digit classification problem as an alternative because of its straightforward specification and extensive documentation. It is our hope that this will afford more opportunity to highlight the algorithms and significance of the data with respect to machine learning rather than placing the emphasis on the complexity of the problem domain.

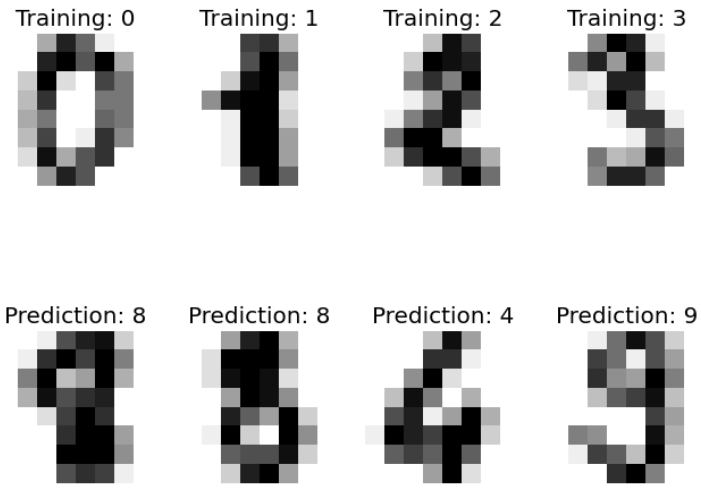


Figure 1: Hand Written Digits Example [4]

2.2 Optimization Problems

The four peaks problem is predefined in ABAGAIL as `FourPeaksEvaluationFunction` and is adapted from [1]. The problem is defined in terms of bitstrings of length 100 where a parameter T defines a threshold for a reward term of the fitness function. The goal is to maximize $z(x)$, the longest run of zeros starting from position 100 and moving backwards in the bitstring and $o(x)$ the longest run of 1s starting from position 1 in the bitstring. The reward term increments the fitness value by some large number if both objectives exceed the threshold T, that is if $o(x) > T \wedge z(x) > T$. The problem has two local maxima and two global maxima, hence the name 'four peaks'.

The k-coloring optimization problem is widely studied in computer science, in our implementation we use the `MaxKColorFitnessFunction` from ABAGAIL. Given a graph G with k possible colors for each of its verticies $V \in G$, the k-coloring of G is a color assignment st. no two adjacent V have the same color. The n-queens problem is also widely explored in computer science and seeks to find a placement of n queens in a standard chess board st. no queen may capture another in a single turn. We implement this scenario in ABAGAIL using `NQueensFitnessFunction` and `NQueensBoardGame`. The traveling salesman problem is yet another extensively explored and well defined optimization problem, our experiments use ABAGAIL's `TravelingSalesmanRouteEvaluationFunction`. In the traveling salesman problem, we consider the traversal of a graph G where verticies and edges $G =$

(V, E) are thought of as cities and distances respectively. The objective is to find a traversal st. every city is visited, the initial and destination cities are the same vertex, and the length of the traversal path is minimized.

2.3 Algorithms and Evaluation

Here we will discuss the application of RHC, SA and GA to the neural network optimization problem. In the case of the other optimization tasks, the reader should consider the general fitness function f of the corresponding problem however in the context of the digit classifier ANN, we discuss the algorithms in the context of minimizing the sum squared error E. The three optimization algorithms that were applied to finding weights for a neural network operated on a three layer model with 20 hidden units, 64 input units, 10 output units (corresponding to the dimensionality of the image instances and output labels respectively). Simulated annealing was run with initial temperature parameter $T = 10^{11}$ cooling rate $c = .95$. The genetic algorithm was run with the `StandardGeneticAlgorithm` ABAGAIL class with a population size $p = 200$ mating rate $m = 100$ which is the number of individuals which mate per iteration and mutation rate $\mu = 10$ which gives the number of individuals which mutate per iteration.

In randomized hill climbing, the weights of the network are initialized to a set of small random values. It is helpful to think of the network as having a particular state on each iteration which is a vector of weights ω in the high dimensional weight space. The network computes the local gradient of the error with respect to the weight vector $\frac{\partial E}{\partial \omega}$ and updates ω along the path of steepest descent (in the case of error minimization). This can be thought of as a greedy search through the weight space of the network. Simulated annealing conducts a randomized version of this search in a similar way, however in the case where on an iteration $t, \neg\exists\omega_{t+1}$ st. $E_{t+1} < E_t$ an ω_{t+1} local to ω_t is selected with probability proportional to the temperature term T. On each iteration, $T \leftarrow T - c$ where c is the cooling rate of the algorithm.

Genetic algorithms conduct a similar stochastic search through the parameter space of ω (and equivalently the hypothesis space H of possible neural networks given the constrained topology). However the implementation and representational structure differ from SA and RHC. The GA represents unique weight vectors ω as individuals in a randomly generated population Ω. On each iteration, the fitness (in this context taken to be the sum of squared error) of each individual $E(\omega)$ is evaluated and the m individuals with the least error are selected for breeding via a crossover procedure on the bits corresponding to each weight parameter in ω. μ of the individuals in the new population are

mutated by bit-flipping a single weight according to ABAGAIL's `DiscreteChangeOneMutation`.

The Mutual Information Maximization for Input Clustering (MIMIC) [2] algorithm is applied along with the aforementioned three to the other optimization problems. MIMIC estimates probability densities over the input space and iteratively samples from the top N-percentile of the estimate to optimize the fitness measure. On each iteration, MIMIC updates its estimation p^θ and decrements $\theta \leftarrow \theta - \epsilon$ st. an increasing smaller region of the parameter space is sampled. Good estimates $\hat{p}_\pi$ of the true probability distribution P minimize the Kullback-Leibler divergence $D_{KL}(P||\hat{p}_\pi)$. In the case where the true distribution P is not known, we may estimate P using a dependency tree which maximizes the mutual information between each feature node and its parent. Therefore, on a particular iteration t, a best estimate of the θ^{th} percentile of the data $P^\theta_{t+1}(x)$ is given by the maximum spanning dependency tree with respect to the mutual information I over the input features of the data.

In the neural network digit classification optimization problem, the 'fitness' measure is defined in ABAGAIL as `SumOfSquaresError`. In the context of our neural network, given training data $D, |D| = n$, the fitness function is simply the mean squared error of the network output. Therefore the full optimization problem is given by:

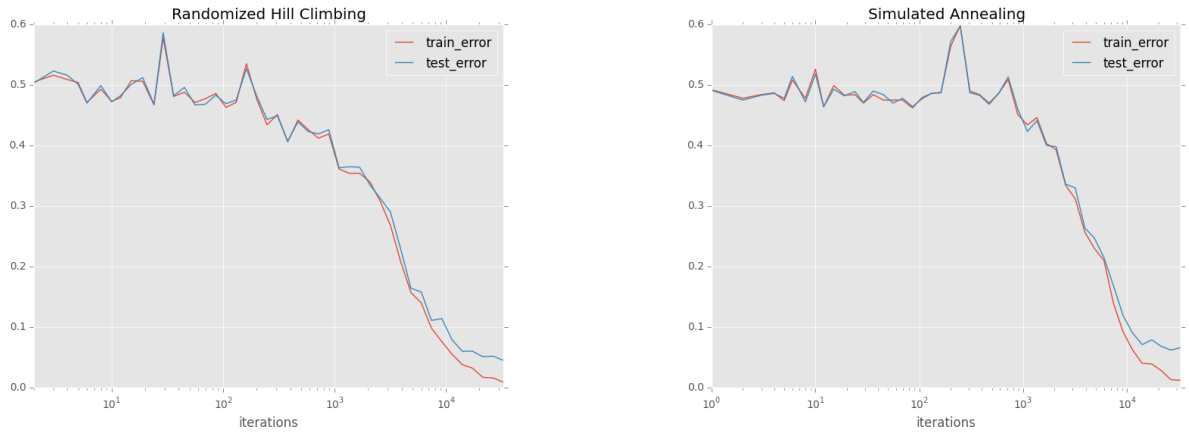
$$\underset{h \in H}{\operatorname{argmin}} \left[\frac{1}{n} \sum_{i=1}^{n} (\hat{Y}_i^h - Y_i)^2 \right] \tag{1}$$

Definition for the four peaks, k-colors, n-queens and traveling salesman problems are given in the Optimization Problems sub-section of this paper.

3 Results

Broadly defined, our analysis considers 3 factors in comparing the relative strengths and weaknesses of the optimization algorithms previously defined: Clear convergence to some solution, quantitative optimal of that solution, approximate time complexity exhibited by the algorithm as a function of number of iterations and / or fitness of solution. Taking these methods as a jumping-off point, a series of experiments were defined in which I varied the parameters of the optimization algorithms and the relative complexity of the optimization problems themselves in order to glean insight into their capabilities.

In the experiments applied to the problem of finding optimal weights for a neural network both RHC and SA converged to an optimum and show an obvious trend as a function of number of



(a) Randomized Hill Climbing

(b) Simulated Annealing

Figure 2: Neural Network Optimization Algorithm Comparison

iterations (figure 2). Wall-clock runtimes illustrated that though all algorithms are approximately $O(n^k)$ where n is the number of iterations, k_{GA} was some large factor greater than k_{SA}, k_{RHC}. The GA also did not converge in the (several hour) run of the maximal number of iterations considered. This is likely due to the representational bias of the GA which encodes the population of possible solutions as bit strings. The weight parameters of the neural network previously described are continuous values; as such the search conducted by the SA and RHC is traversal in $\mathtt{R}^{I \times H \times O}$ where the weights of the neural network are dimensions of the space. Intuitions about gradient descent processes and the error surface w.r.t the weight space hold in the context of SA and RHC. In the case of GA, we consider the binary encoding of these values, in a scenario where crossovers and point mutations may lead to orders of magnitude difference in the corresponding floating point evaluation of the bits.

In the application of the optimization algorithms to the K-coloring problem, all three performance measures were taken into consideration. Difficulty of the problem was varied by changing the number of colors k while keeping the structure of the underlying graph fixed. RHC performed the worst in terms of wall-clock runtime and could not find a solution for $k > 2$. Ability to find a solution and wall-clock times for SA and GA followed with convergence for $k < 5$ and $k < 6$ respectively. MIMIC not only showed fewer evaluations to reach a solution and generally shorter wall-clock times, it was also the only algorithm to find a solution for more than 6 colors. This result clearly illustrates MIMIC's advantage on this task; the relative performance of the algorithms indicates that the k-colors problem's fitness function is very complex and may have many local minima. It is also possible that MIMIC and GA have a representational advantage over the other algorithms as the graph may be stored as an adjacency list with colors encoded as a small bit string of the appropriate precision

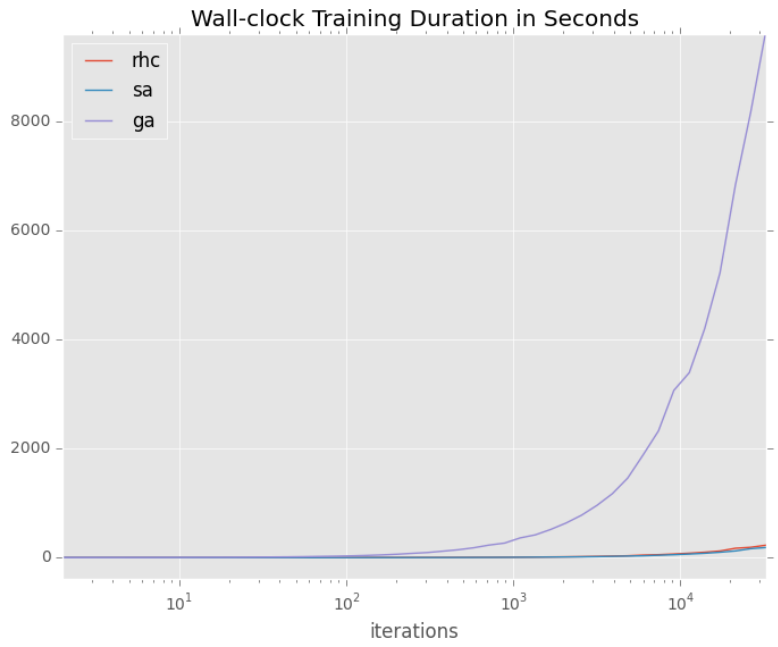
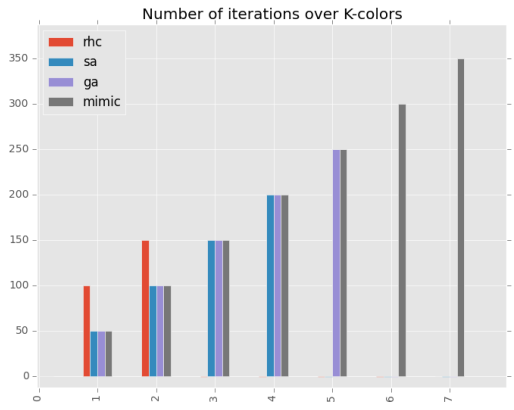


Figure 3: Neural Network Wall-clock Algorithm Runtimes

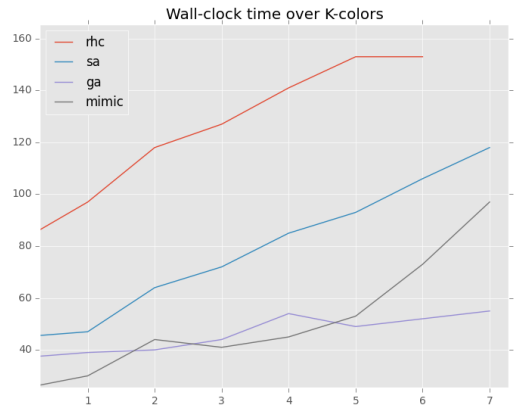
(in this case of length at most 3).

In the four peaks problem, the fitness of the optimized solution for all algorithms is roughly equivalent for $N < 25$. After this point SA and RHC rapidly diverge from GA and MIMIC as they loose the ability to converge to good solutions. Ultimately, MIMIC exhibits the most robustness to handle the four peaks problem for large values of N. As noted in [1], this particular problem is specifically designed to be solvable by GA. The locality of bits is precisely the feature of the problem which methods such as GA and MIMIC exploit. This is in some ways the inverse of the scenario seen in the neural network weight optimization problem, in which the continuous valued weights proved problematic for the GA.

In the N-Queens problem, though some variation in the number of solutions discovered by each algorithm as a function of N showed some potentially promising trends; the most indicative factor representing performance was the wall-clock runtime. MIMIC exhibited roughly $O(log(N))$ wall-clock time complexity, the GA showed roughly $O(N)$. The most surprising was in the case of SA and RHC which not only discovered (generally) more solutions to the N-Queens problems, they also appear to have $O(1)$ time complexity on this particular problem! This result was somewhat surprising in that it would seem that a good representation of the chess board would be a binary $2d$ array in which queen locations could be denoted by 1's and empty cells with 0's. This would seem to be and ideal problem for solution by GA or MIMIC; however it appears that in this particular instance, gradient methods

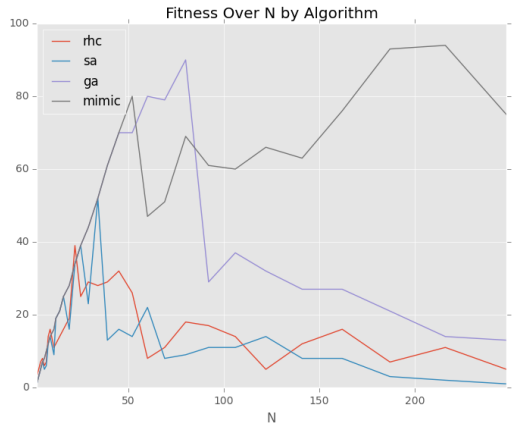


(a) Iterations to Find k-coloring

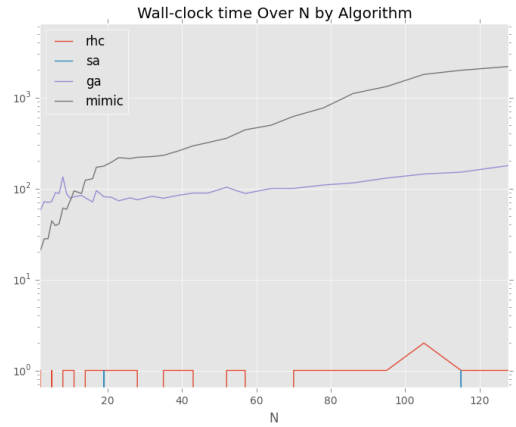


(b) Wall-clock Runtime in MS

Figure 4: K-Colors Algorithm Comparison



(a) Four Peaks Fitness by Algorithm



(b) N-Queens Runtime by Algorithm

Figure 5: Four Peaks and N-Queens

prevail. It could be that I am making some mis-representation of the underlying implementation of N-Queens in ABAGAIL or perhaps the board state in the space of possible board configurations follows the geometric assumptions which are typically exploited in RHC and SA.

4 Conclusion

In this study we have explored the application of four randomized optimization algorithms to four unique problem domains. Given the problem of finding optimal weights for a neural network classifying hand-written digits, we reported evidence that local gradient-based methods such as random hill climbing and its stochastic, thermodynamics-inspired variant simulated annealing are well suited

methods. The genetic algorithm used in these experiments failed to converge to a solution in the considered time and iteration intervals; further its time complexity is several orders greater than that of random hill climbing and simulated annealing. In the case of the k-coloring problem, MIMIC and GA have clear advantages over other methods both in terms of wall-clock time and ability to find a solution. MIMIC also exhibits the ability to solve the problem for $k \leq 7$. In four peaks, we discussed the performance advantage of MIMIC and GA for large values of N. However as we noted, this is a 'toy' problem which has been specifically engineered to be solvable by these methods. In the N-Queens problem, we observed shorter wall-clock times and slightly improved sets of board solutions using simulated annealing. Over all, we conclude that the representational structure of the particular optimization problems is a primary determinant of the best-suited method for finding a solution.

References

[1] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. Technical report, Pittsburgh, PA, USA, 1995.

[2] Jeremy S. De Bonet, Charles L. Isbell, Jr., and Paul Viola. Mimic: Finding optima by estimating probability densities. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, page 424. The MIT Press, 1996.

[3] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2015-09-15].

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Pushkar. Abagail, 2015.