Tim Vergenz
Charles Isbell
CS 4641: Machine Learning
March 15, 2015

Assignment 2: Randomized Optimization

Algorithms

Randomized Hill-Climbing

There are a few variations (hacks) on basic hill-climbing that can be applied to improve performance, but for the purposes of this project two were chosen to be applied:

1. Rather than terminating the climb when the first neighbor examined is of lower value, randomly choose another neighbor up to $k - 1$ times, and terminate only if all k neighbors were of lower fitness.
2. After terminating the ascent of a particular hill, restart the process n times, keeping the best of the $n + 1$ attempts.

Simulated Annealing

In order to be comparable to our implementation of Randomized Hill-Climbing, the Simulated Annealing implementation was modified to restart similarly to RHC, keeping the maximum solution found. The temperature was decreased linearly from T_0 down to 0. Neighbors of a higher fitness were kept with probability 1, and neighbors of a lower fitness were kept with probability $e^{(F(n) - F(c)) / T}$, where F is the fitness function, n is the neighboring solution, c is the current solution, and T is the current temperature.

Genetic Algorithm

A standard genetic algorithm was employed using a population size of 100, generating 50 random offspring and 10 mutations on every iteration. Offspring were chosen by interweaving two parents randomly: for each index i, a coin toss determined whether c_i came from p_1 or p_2. Mutation involved choosing a random index and flipping the bit at that index.

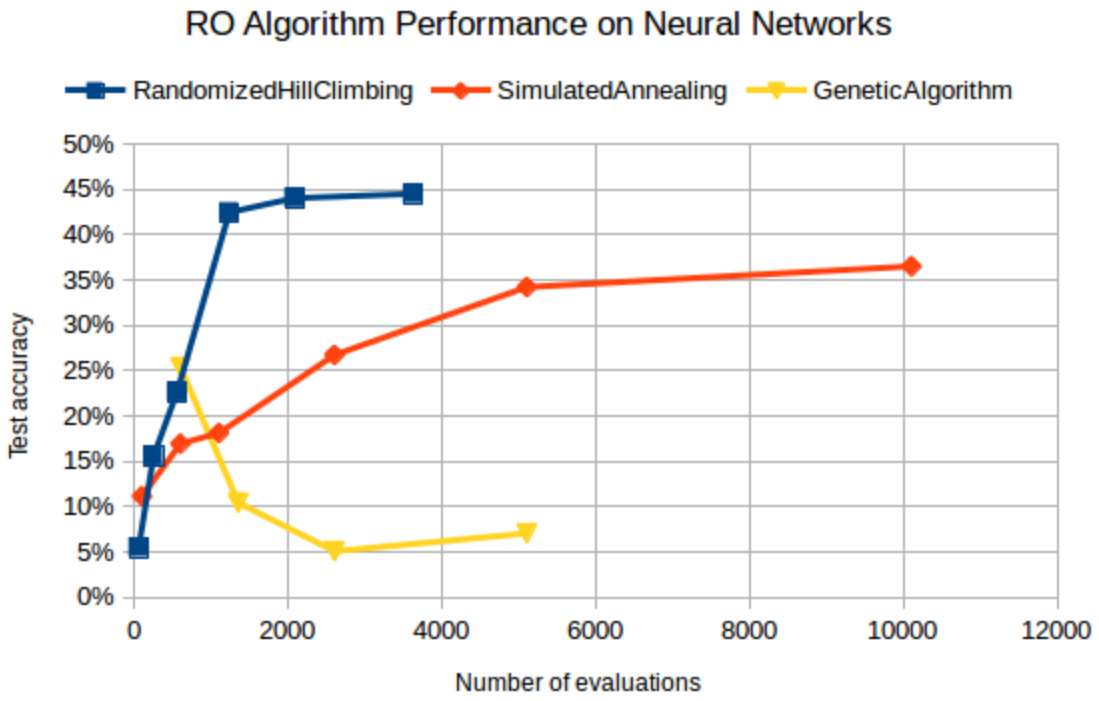
MIMIC

An implementation of MIMIC was used with the population size set to 100, where the top 50 samples were kept each iteration. The number of iterations was varied.

Neural Network Weight Optimization

For the first part of the assignment three of the four randomized optimization algorithms were applied to the task of choosing weights for an artificial neural network. The data being optimized over was from the Citibike dataset. (As a reminder, this was a join of hourly weather data from New York City with the number of trips taken by members of the city's bike share program that hour.)

The figure below shows the accuracy of each of the algorithms on this problem as a function of the number of evaluations required to reach that accuracy. (In all test cases, test accuracy was within 1% of training accuracy, so for simplicity only test accuracy is shown. Note the log scale.)



Each datapoint in the graph represents a single parameter configuration, and data for each parameter configuration was gathered from 10 trials and averaged. For hill-climbing and simulated annealing, the number of restarts was varied. For the genetic algorithm the number of iterations was varied.

Upon observation, a number of details stand out:

1. **Randomized hill-climbing performed the best** of all algorithms tested. This was unexpected, though does make sense in retrospect. Hill-climbing works most effectively in a monotonic space with relatively few non-optimal local maxima, and the space of

neural network weights does intuitively seem to fit that description. (Consider that randomized hill-climbing on negative error is essentially a lower-resolution version of the standard neural network training algorithm, gradient descent.)

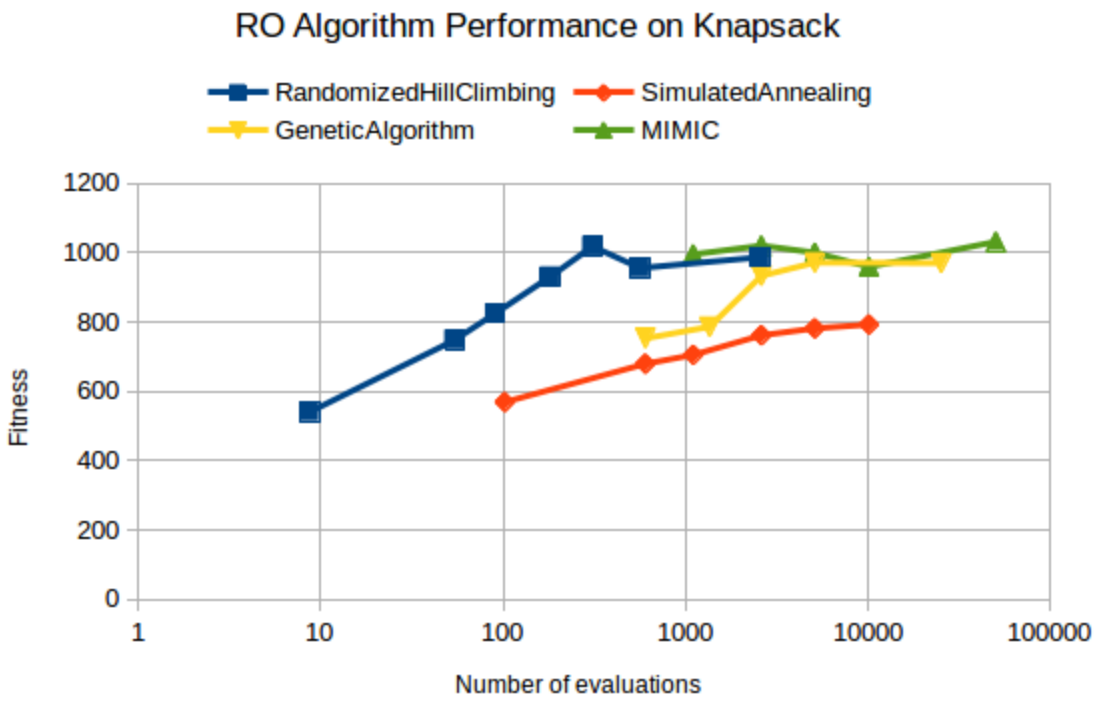
2. **The genetic algorithm performed terribly**. Runs with more iterations--which one would expect to perform better--only did worse. (There was one more run not included on the graph for visual purposes that ran five times as long with nearly identical results.) This abject failure was most likely because success of a given solution was highly dependent on the interactions and relationships between the weights, so crossover between two otherwise unrelated solutions was counterproductive.
3. **Simulated annealing performed marginally worse than plain hill-climbing**. This is consistent with our deductions from point (1): since local optima do not appear to be much of a problem in this space, SA's acceptance of less-optimal neighbors only served to slow down its progress.
4. **Training and testing accuracy were in all cases within 1% of each other**. This was the most interesting result. The only explanation I can think of is that since relatively little information from the actual data was used to guide each step in the decision process, there was little opportunity for overfitting to come into play. The final weights retained lots of randomness, so nuances in the noisy training landscape were lost and only general trends remained.

Overall the performance of randomized optimization algorithms on neural networks was still significantly worse than that of standard backpropagation and gradient descent (compare hill-climbing's 44% test accuracy with backprop/gradient descent's 63%), with only minor savings in training time.

Knapsack Problem

The problem is a fairly straightforward one: given a set of items with a value, volume and inventory count for each, as well as a maximum volume that is available for packing, determine the number of each item to pack to maximize value.

Below are the results of an experiment on this problem:



Again, each datapoint represents the average of 10 trials with a single parameter configuration. Number of restarts was varied for hill-climbing and simulated annealing, and the number of iterations was varied for the genetic algorithm and MIMIC.

A few observations:

1. **Hill-climbing performed surprisingly well**. For being an NP-complete problem I expected it to perform poorly, but I suppose if you randomly restart enough times you're bound to find some sort of reasonable solution.
2. **MIMIC performed better than the genetic algorithm sooner**. It took fewer function evaluations for MIMIC to reach its high performance before it leveled off than it took for the genetic algorithm. This makes sense, since genetic algorithms take a more chaotic, less goal-oriented approach to finding solutions, while MIMIC attempts to directly model the probability distribution.

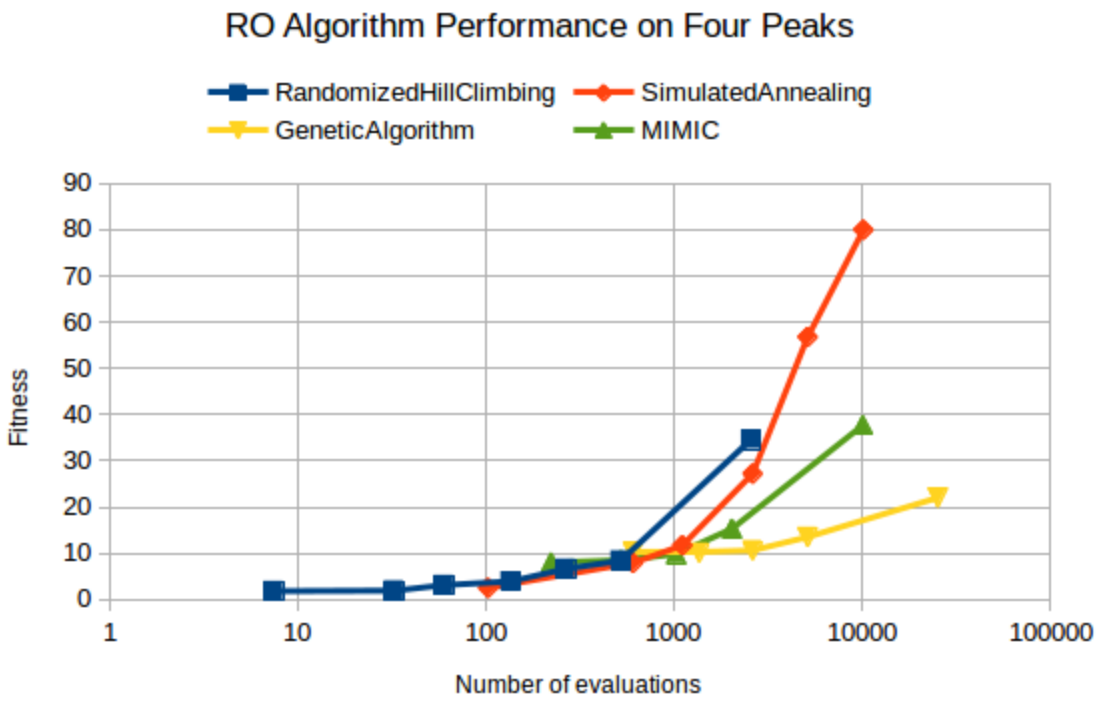
It is unclear how close the algorithms came to globally optimal solutions due to the nature of the problem, but the comparative trends are interesting nonetheless.

Four Peaks

The four peaks problem is a simple discrete problem with two global maxima, and two suboptimal local maxima. It is defined as $f(X,T) = max[tail(0,X), head(1,X)] + R(X,T)$,

where *tail* and *head* refer the longest prefix and suffix of that digit in X, and $R(X, T)$ is N if $tail(0, X) > T$ and $head(1, X) > T$, and 0 otherwise.

Tests for this problem were instantiated with $N = 80$ and $T = 20$.

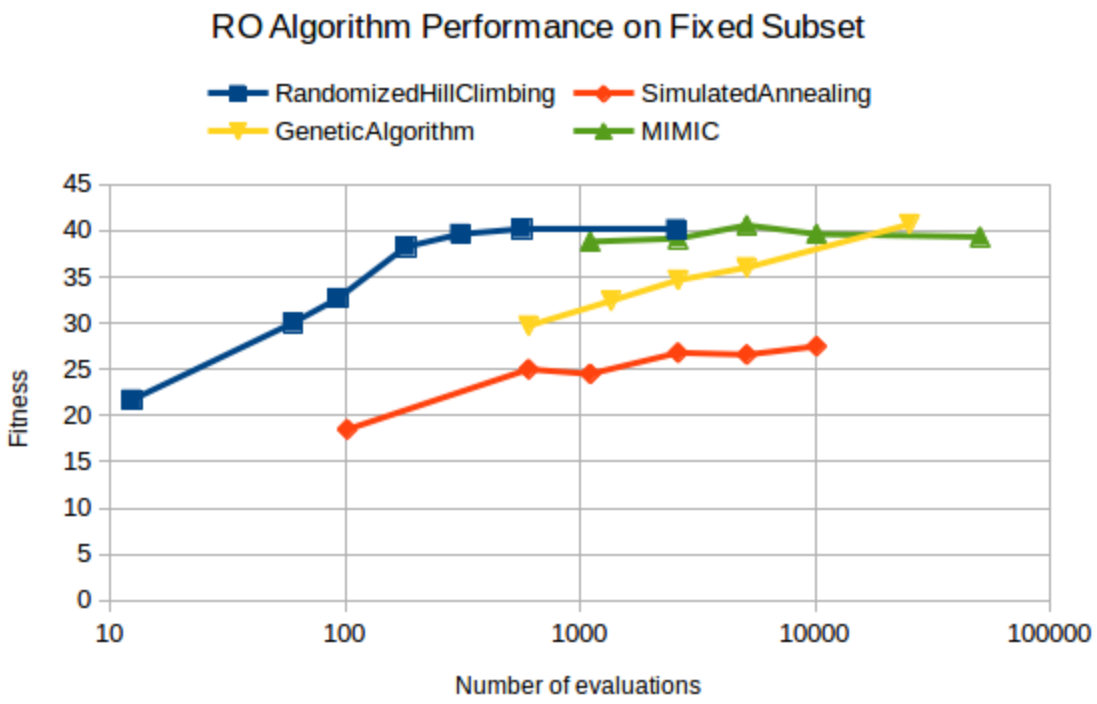


1. In the majority of cases tested, **MIMIC did not find the global optimum**. Since the problem structure has such a large basin of attraction for the suboptimal local maxima, perhaps MIMIC was only able to consistently train based on results from within that basin. A reasonably accurate probability distribution for samples within that basin is not likely to closely resemble an accurate probability distribution for the global maxima, so it would be difficult to break out of that model once there. (Note: It is possible that this did happen in some trials, but it did not happen frequently enough to significantly impact the average result.) It is also possible that MIMIC would more consistently find the optimum given more time.
2. **Simulated Annealing was the clear winner on this problem**. I believe the start temperature and cooling schedule were tuned well enough for this problem that the algorithm was able to consistently get out of the local maximum basin and find the global optimum. With enough repetitions simulated annealing consistently more than doubled the fitness of other algorithms for the same number of fitness function evaluations.

Fixed Subset Problem

The Fixed Subset problem was a simple discrete optimization problem consisting of the following: for a given instance of the problem with solution size N, a certain subset of the indices from 1 to N were selected and assigned fixed values. (Each index was independently decided with some probability p of being fixed, and if fixed the value was chosen uniformly randomly.) Candidate solutions were then assigned a fitness based on how many fixed bit positions matched their assigned values.

All test cases had $N = 80$ and $p = 0.5$, yielding an expected maximum fitness of 40.



1. **MIMIC consistently and quickly reach the optimal solution**. Since it was examining probability distributions for each bit individually, it was quickly able to recognize patterns in fitness level with respect to the fixed bits and converge to the maximum.
2. **The genetic algorithm found the maximum**, and technically performed the best for a certain number of evaluations, though this could have been due to random variations in the problem instances.
3. **Similar to neural networks, simulated annealing performed poorly due to the monotonicity of the fitness function**. There were no local optima in this problem. (For every non-optimal solution, there was some bit that was not set to its fixed value that

could be flipped to attain a higher score.) Therefore, accepting inferior solutions with any non-zero probability will reduce performance.

Discussions

Running Time

The genetic algorithm and MIMIC took by far the most amount of real-world time to reach their solutions, but that depended largely on the problem at hand. For instance, the neural network experiments were the only experiments that took on the order of hours to run, largely because the fitness function was very expensive to compute. (After all, at each evaluation, all 5802 training instances had to be pushed through all 55 nodes in the network, and error had to be computed for each.) In all other problems the experiments completed quickly, in orders of magnitude less time.

Improvements

The most important improvements to be made in the algorithms on the problems tested would be further effort in tuning the parameters. Investigation into the effects of modifying parameters (such as population sizes; cooling parameters and function families, i.e. whether the temperature should decrease linearly or exponentially; mutation and crossover rates; and mutation, crossover, and neighborhood functions) would have likely had significant impact on the algorithms' performance. Some work was done in this area, but previous results are not presented.

Conclusions

Overall, different algorithms presented strengths and weaknesses in different circumstances:

Algorithm...	worked *most* effectively when...	worked *least* effectively when...
Randomized Hill-Climbing	● there were no or few non-optimal local maxima	● the solution surface was rough or had many local optima
Simulated Annealing	● good solutions were "near" bad solutions (i.e. random walk from bad could land on good)	● there were no or few non-optimal local maxima
Genetic Algorithms	● fitness did not depend extensively on relationships within solution ● multiple subproblems within the space	● fitness depended on complex interplay between attributes ● structure was not adequately described by crossover operation
MIMIC	● there was complicated structure in the solution space ● fitness computation was expensive	● fitness computation was cheap