# An Exploration of Randomized Optimization

Stephen Shepherd | sshepherd35@gatech.edu | 03/5/2022

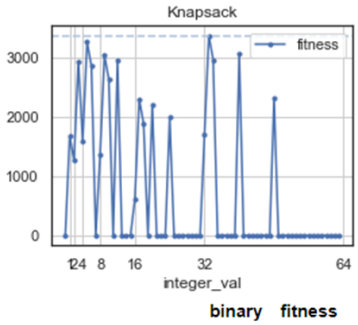
## Introduction

I will explore 4 randomized optimization algorithms (Random-Restart Hill Climbing, Simulated Annealing, Genetic Algorithm, MIMIC) as applied to 3 "toy" problem spaces. I will also use the first 3 algorithms to optimize the weights of a Neural Network for a classification problem.

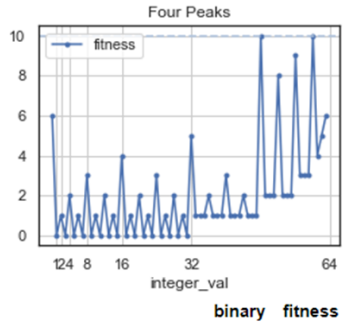
## The "Toy" Learning Problems

I chose 3 different "toy" problems of discrete valued parameter spaces to compare and contrast the algorithms. The parameter spaces are represented as binary bit strings. The functions representing these problems are described below.

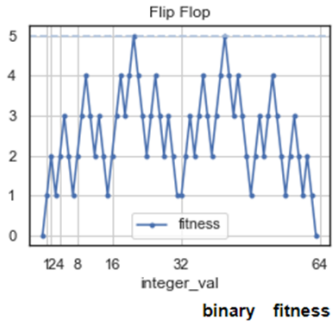
- **Knapsack problem**: given a set of $n$ items each with a weight $w$ and a value $v$, maximize SUM($v_i$) by either including or discarding item $x_i$ subject to SUM($w_i$) < $W$ (max weight).
- **Four Peaks problem**: $f$ (x) = MAX ( $o$(x) , $z$(x) ) + $R(x,T)$, where $R(x,T) = n$ if $o$(x) > T and $z$(x) > T and 0 otherwise, and $z$(x) = number of contiguous 0's ending at the last position and $o$(x) = number of contiguous 1's starting at the first position, and $T = t_{pct} * n$.
- **Flip Flop problem**: count of paris of consecutive digits where $x_i$ != $x_{i+1}$.




| integer_val | binary | fitness |
|---|---|---|
| 33 | [1, 0, 0, 0, 0, 1] | 3365.0 |
| 5 | [0, 0, 0, 1, 0, 1] | 3261.0 |
| 40 | [1, 0, 1, 0, 0, 0] | 3056.0 |




| integer_val | binary | fitness |
|---|---|---|
| 48 | [1, 1, 0, 0, 0, 0] | 10 |
| 60 | [1, 1, 1, 1, 0, 0] | 10 |
| 56 | [1, 1, 1, 0, 0, 0] | 9 |




| integer_val | binary | fitness |
|---|---|---|
| 21 | [0, 1, 0, 1, 0, 1] | 5 |
| 42 | [1, 0, 1, 0, 1, 0] | 5 |
| 53 | [1, 1, 0, 1, 0, 1] | 4 |

***Figure 1**: The three toy learning problems, demonstrated with input size of 6, with fitness plotted as a function of integer value of the bitstring. The 3 fittest bit strings are displayed below each plot.*

These three toy problems provide diversity with regard to problem structure and usefulness of direct neighbors as compasses to optima.

- The **Knapsack problem highlights the strengths of the Genetic Algorithm** and presents a challenging random structure determined by the weights and values of the items. High value items with lower weight are commonly present in the top candidates, but a neighbor candidate can have a drastically different fit based on just one item.
- The **Four Peaks problem highlights the strengths of Simulated Annealing (as well as GA)** and has a large region designed to trap hill climbing algorithms with two suboptimal local optima, and a smaller region with the two true optima. This provides a two-region structure where direct neighbors serve as a noisy compass pointing toward suboptimal local optima, making broader exploration necessary.
- The **Flip Flop problem highlights the strengths of MIMIC** and provides a symmetric but oscillating structure, similar in appearance to a Galaga starship. Only bit strings consisting of a sequence of alternating bits are optima here: each bit has a dependency on the preceding bit.

## Setup

The mlrose_hiive library was used, modified slightly to stop each algorithm if the true global maximum is reached. An input size of 100 was used for each problem. For summary tables, 10 runs with different random states were examined and metrics are reported as an aggregate (eg. % of runs reaching global maximum, average # of fitness evaluations, etc.).

## Highlighted Results

% of runs reaching maximum

| problem<br>algorithm | knapsack | four_peaks | flip_flop |
|---|---|---|---|
| genetic | 0.90 | 1.00 | 0.00 |
| simulated_annealing | 0.00 | 0.90 | 1.00 |
| mimic | 0.00 | 0.00 | 1.00 |
| randomized_hill_climbing | 0.00 | 0.20 | 0.00 |

avg run seconds

| problem<br>algorithm | knapsack | four_peaks | flip_flop |
|---|---|---|---|
| genetic | 26.04 | 25.53 | 189.25 |
| simulated_annealing | 0.03 | 9.72 | 2799.34 |
| mimic | 99.28 | 299.64 | 53.86 |
| randomized_hill_climbing | 3.47 | 26.38 | 54.30 |

***Figure 2**: Problems highlighting each algorithm. In cases where fitness results are similar, the highlighted algorithm is more efficient in terms of wall clock time.*

## Remarks on the Three Toy Problems

We see three primary themes impacting performance on the 3 toy problems:

1. **Exploration vs. Exploitation:** RHC is unable to scale to larger input sizes as its ability to explore these spaces depends on randomly falling into a good region. Simulated Annealing is more flexible about exploration and scales to larger inputs.
2. **"Neighbor" Candidate Evaluation**: Genetic and MIMIC are able to beat RHC and Simulated Annealing for Knapsack and Flip Flop by generating candidates either

randomly or using dependency trees. This is helpful when fitter candidates may not be in the immediate neighborhood RHC or Simulated Annealing in each iteration.

3. **Structure**: MIMIC is able analyze the Flip Flop structure to find ideal bit string sequences, whereas Genetic is able to randomly try Crossover and Mutation in Knapsack where the structure is more random.

# Random-Restart Hill Climbing ("RHC")

**RHC was only able to find the global optima for the Four Peaks problem, and was only able to do so in 2 of 10 runs**. It reached a fitness of ~92% of the global optima for the Flip Flop problem and ~86% for the Knapsack problem.

| problem | maximum | avg_best_fitness | avg_best_fitness_pct | pct_reached_maximum | avg_state_neighbor_attempts | avg_fitness_evaluations | avg_seconds |
|---|---|---|---|---|---|---|---|
| knapsack | 60121 | 51803.40 | 0.86 | 0.00 | 499.18 | 829432.70 | 2.73 |
| four_peaks | 189 | 127.20 | 0.67 | 0.20 | 374.36 | 6734167.90 | 25.32 |
| flip_flop | 99 | 90.60 | 0.92 | 0.00 | 497.81 | 1319588.80 | 51.57 |

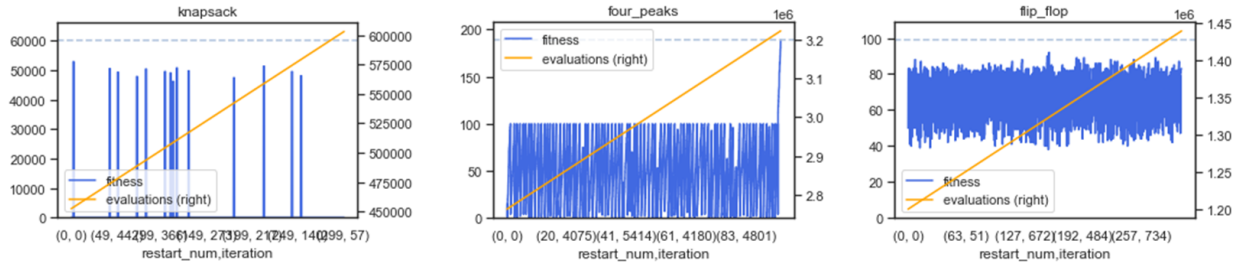



***Figure 3**: RHC performance on all 3 problems using maximum # of 500 neighbor search attempts and 300 random restarts. The bottom 3 plots show fitness by restart number and iteration number for the best performing run.*

It's clear from the Four Peaks iterations chart that the large region containing the local optimas traps most of the search, but it eventually is able to randomly stumble into the region containing the global optima if given enough restarts, and then climb to the top quickly. This would not scale well to larger problem sizes as demonstrated later. For Knapsack and Four Peaks, in the *Figure 3* table we see the average # of neighbor searches for each state is close to the maximum allowed: **for most states, RHC can't find neighbors that are a better fit**. This is intuitive as the Knapsack problem contains many points where fitness is 0 due to item weight surpassing the knapsack weight capacity, so adjacent neighbors may also be 0. Flip Flop suffers a similar scenario: flipping 1 bit may cause it to differ with the bit to the left, but may then match the bit to the right, causing net zero improvement. So, immediate neighbors aren't guaranteed to be helpful in this scenario.

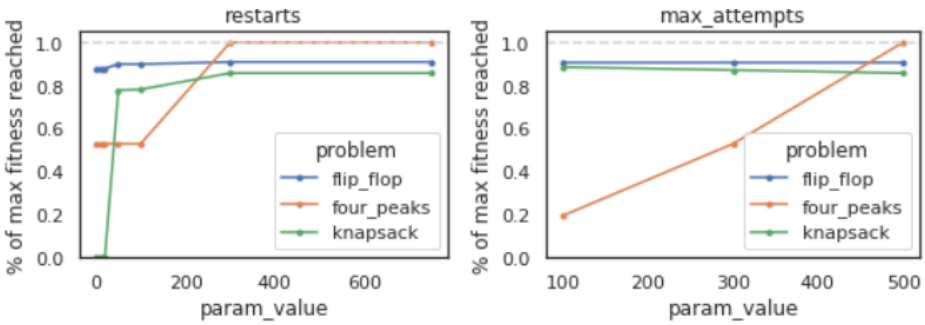



***Figure 4**: RHC performance on the Four Peaks problem. Results shown by # of random restarts allowed and # of random neighbor search attempts allowed.*

Examining hyperparameters, we see that more restarts and neighbor search attempts allows the algorithm to explore. We could increase these values, but given the lengthy runtimes and only 20% success rate on Four Peaks, I'd rather investigate other approaches. To improve the RHC approach, I'd consider an implementation that remembered results from regions already explored to avoid resampling those regions on future random restarts.

# Simulated Annealing

**Simulated Annealing's exploration abilities improved performance for Four Peaks significantly over RHC**. 8/10 runs reached the global optima and did so in 1/5 the time as RHC. Simulated Annealing can solve the exploration challenges we encountered in RHC by being more flexible in how the algorithm selects neighbors. We are using the Four Peaks problem to highlight this algorithm, so I did a parameter search for initial temperature and exponential decay and found an initial temperature of 10 with a slow decay of .000005 to .00001 to be the sweet spot. This high initial temperature allows the algorithm to explore the space more at first, jumping to neighbors that may be slightly less fit, before cooling down and being more selective about neighbors. **Notice the more structured search in the Four Peaks iteration chart in *Figure 5***: there is oscillation early on exploring neighbors of flexible fitness, but later on the algorithm hones in on the region containing the optima and stays there as the temperature cools until it finds the peak. Instead of reaching one of the suboptimal peaks near the optima and randomly restarting, Simulated Annealing is able to gradually explore the region until finding the true optima.

| problem | maximum | avg_best_fitness | avg_best_fitness_pct | pct_reached_maximum | avg_state_neighbor_attempts | avg_fitness_evaluations | avg_seconds |
|---|---|---|---|---|---|---|---|
| knapsack | 60121 | 48595.00 | 0.81 | 0.00 | 252.73 | 1093.60 | 0.02 |
| four_peaks | 189 | 171.20 | 0.91 | 0.80 | 13.19 | 407886.00 | 5.08 |
| flip_flop | 99 | 98.90 | 1.00 | 0.90 | 21.21 | 640037.30 | 142.44 |

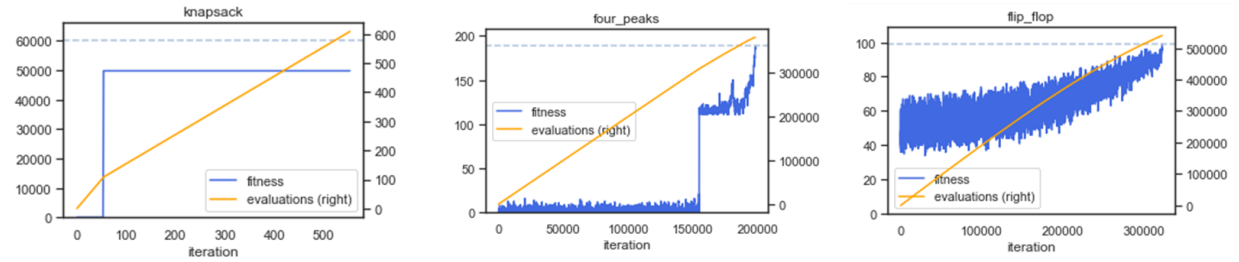



***Figure 5**: Simulated Annealing performance on all 3 problems using maximum # of 500 neighbor search attempts, initial temperature of 10 with exponential decay of 0.000005. The bottom 3 plots show fitness by iteration number for the best performing run.*

We can see from the parameter exploration in *Figure 6* that slower decay rates generally lead to better results.. This emphasizes the ability of Simulated Annealing to balance exploration with exploitation, and more consistently find the Four Peaks optima with less effort.

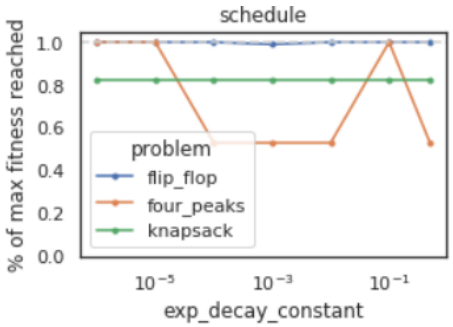



***Figure 6**: Simulated Annealing performance by exponential decay rate.*

Notice that Simulated Annealing also performs well on the Flip Flop problem, but averages over 2 minutes to find the maximum. It fails to find a solution for the Knapsack problem. The optima are more randomly distributed in Knapsack with lots of values with fitness of 0: once Annealing finds a local optima, neighbors of fitness 0 are so unfit even with the more flexible selection that they aren't explored.

# Genetic Algorithm ("GA")

**The Genetic Algorithm is the only one that finds the true optimum of the Knapsack problem.** 9/10 runs find the optimum in ~25 seconds. Notice that the iteration curves for GA are monotonically increasing with dramatic advances. Instead of exploring direct neighbors by

sacrificing fitness constraints like Simulated Annealing, the GA splices together traits of the best candidates it has found so far and tests small random mutations. **It is able to draw the best traits from distant neighborhoods and combine them to find the optimal value**. This makes it especially strong for the Knapsack problem where combinations of key item selections are crucial, and where the optimal value may be an outlier in its direct neighborhood.

| problem | maximum | avg_best_fitness | avg_best_fitness_pct | pct_reached_maximum | avg_state_neighbor_attempts | avg_fitness_evaluations | avg_seconds |
|---|---|---|---|---|---|---|---|
| knapsack | 60121 | 60119.00 | 1.00 | 0.90 | 3.96 | 409069.70 | 25.61 |
| four_peaks | 189 | 189.00 | 1.00 | 1.00 | 3.22 | 358955.60 | 26.50 |
| flip_flop | 99 | 93.60 | 0.95 | 0.00 | 15.03 | 668337.00 | 170.52 |

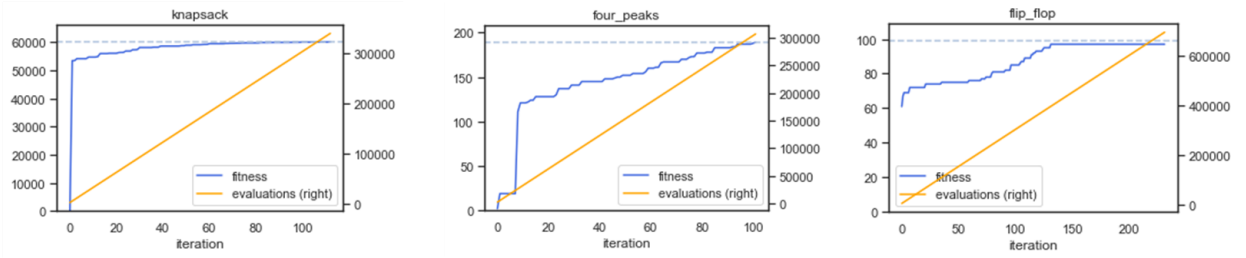



***Figure 7**: Genetic Algorithm performance on all 3 problems using maximum # of 100 attempts and a population size of 3,000. The bottom 3 plots show fitness by iteration number for the best performing run.*

*Figure 8* demonstrates the efficiency with which the Genetic Algorithm handles the Knapsack problem compared to Simulated Annealing. It is able to radically combine mutations from disparate neighborhoods that contain high value items while satisfying the weight constraint.

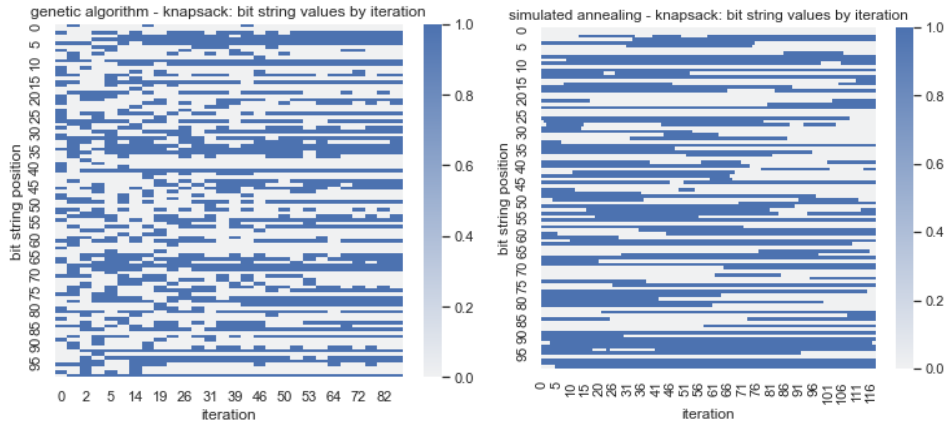



***Figure 8**: Best candidate bit string values by iteration, Genetic Algorithm vs. Simulated Annealing. Simulated Annealing tests close neighbors while Genetic Algorithm performs more radical crossover and mutation.*

The Four Peaks problem was designed to be "GA-Friendly"[1] so the GA unsurprisingly performs well on it as an alternative to Simulated Annealing, although it takes 5x the time. It is easy to see that long stretches of trailing 1's or 0's, which maximize Four Peaks, can be obtained by exploring possibilities via Crossover.

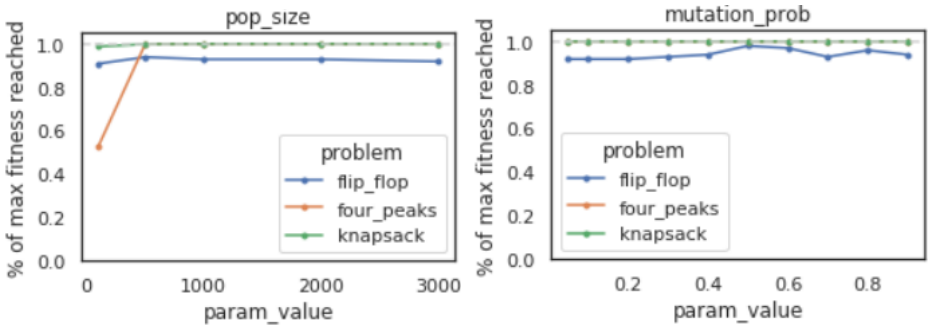



***Figure 9**: Genetic Algorithm performance by Population Size and random Mutation Probability.*

Examining the iteration curve in *Figure 7* for Flip Flop we see that GA reaches a fitness that is 95% of the true maximum value and stops improving. This could be a result of "crowding." To improve the Genetic Algorithm for Flip Flop, I would explore approaches to reduce crowding, where the population becomes less diverse as a handful of similar but imperfect candidates replicate in the population. Mitchell describes some approaches to maintain diversity in the population, such as Fitness Sharing, where "fitness of an individual is reduced by the presence of other, similar individuals in the population."[2] *Figure 9* shows that higher Mutation Rate ~.5 helped diversity in the population for Flip Flop, but still was not able to find the maximum.

# Mutual-Information-Maximizing Input Clustering ("MIMIC")

Unlike the Knapsack problem that has a random structure, the Flip Flop problem has a symmetrical structure where the optima rely on long sequences dependent on the preceding bits. **MIMIC is able to take advantage of Flip Flop's dependent structure by forming dependency trees that generate candidates through structural analysis**. Here is an excerpt from Professor Isbell's MIMIC paper that describes searching for a permutation $\pi$ that is optimal that helps MIMIC detect alternating bit patterns:

$$J_\pi(X) = h(X_{i_1}|X_{i_2}) + h(X_{i_2}|X_{i_3}) + \ldots + h(X_{i_{n-1}}|X_{i_n}) + h(X_{i_n}).$$

> The optimal $\pi$ is the one that produces the lowest pairwise entropy with respect to the true distribution. By searching over all $n!$ permutations, it is possible to determine the optimal $\pi$.
>
> [3]

[1] Baluja, Shumeet, and Caruana, Rich. 1995. *Removing the Genetics from the Standard Genetic Algorithm*. Page 2.

[2] Mitchell, Tom. *Machine Learning*. McGraw-Hill, 1997. Page 259.

[3] Bonet, Jeremy S., Isbell, Charles L Jr.., Viola, Paul. *MIMIC: Finding Optima by Estimating Probability Densities.* Advances in Neural Information Processing Systems 9. 1996

**Accordingly, we see optimal fitness for Flip Flop found in 10/10 runs for MIMIC with few function evaluations and small runtime**. MIMIC is able to locate the optima in ~32 seconds and 30k fitness evaluations, compared to ~142 seconds and 640k fitness evaluations that Simulated Annealing requires.

| problem | maximum | avg_best_fitness | avg_best_fitness_pct | pct_reached_maximum | avg_state_neighbor_attempts | avg_fitness_evaluations | avg_seconds |
|---|---|---|---|---|---|---|---|
| knapsack | 60121 | 53056.90 | 0.88 | 0.00 | 6.06 | 45916.80 | 70.30 |
| four_peaks | 189 | 172.40 | 0.91 | 0.00 | 2.10 | 167181.30 | 244.31 |
| flip_flop | 99 | 99.00 | 1.00 | 1.00 | 1.10 | 30017.20 | 32.62 |

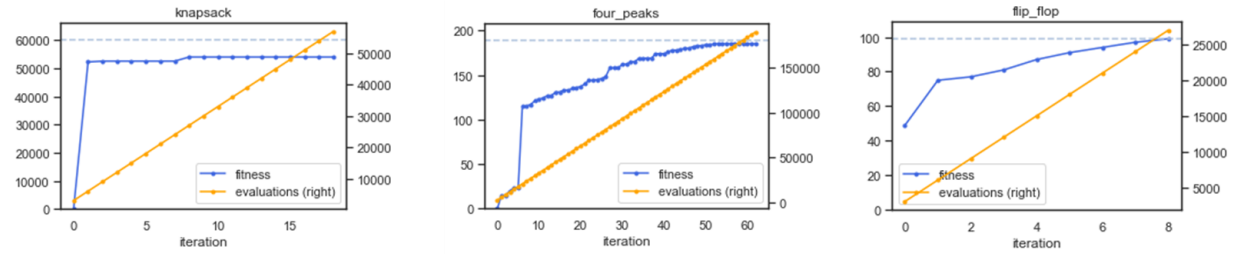



***Figure 10***: *MIMIC performance on all 3 problems using a population size of 3,000, a keep % of 20%, and 10 maximum candidate attempts. The bottom 3 plots show fitness by iteration number for the best performing run.*

Notice in *Figure 11* the efficiency at which MIMIC hones in on sequences of alternating bits. This informed approach is much more likely to quickly find the long optimal bit sequences vs. the random crossover and mutation approaches that GA performs.

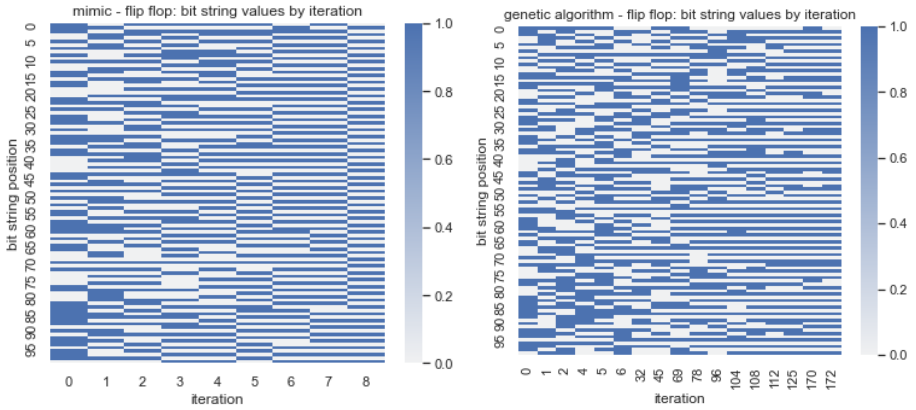



***Figure 11***: *Best candidate bit string values by iteration, MIMIC vs. Simulated Annealing. MIMIC detects the alternating structure of the optimal bitstring with much less effort.*

Examining results by Population Size and % of Top Candidates to Keep, we see that larger population size is generally helpful. % of Candidates showed higher performance at lower quantities for Four Peaks - being more selective in this problem turned out to be useful.

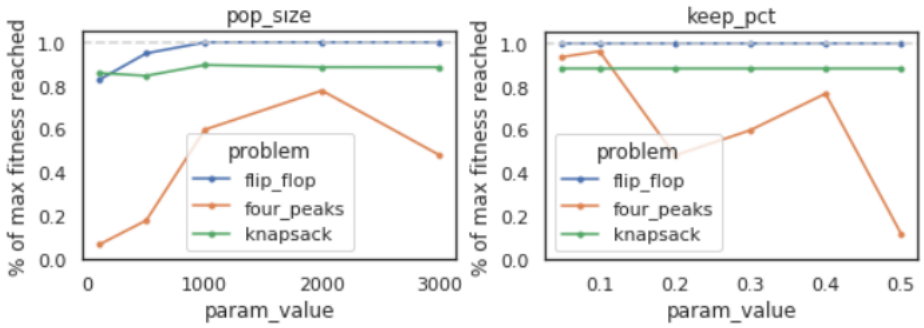



***Figure 12**: Performance by Population Size and % of Top Candidates to Keep.*

## Scaling Input Size

Trying various input sizes, we notice some interesting trends. # of neighbor attempts, restarts, and population size were scaled linearly with input size.

- RHC's performance varied the most alongside problem size in general and will be more difficult to scale compared to the other algorithms.
- All algorithms were able to nearly reach the global optima for Four Peaks for smaller input size. This problem gets more difficult as it scales and GA ends up beating SA.
- Knapsack and Flip Flop were difficult problems regardless of input size. In general, Genetic should be used for Knapsack and MIMIC for Flip Flop.

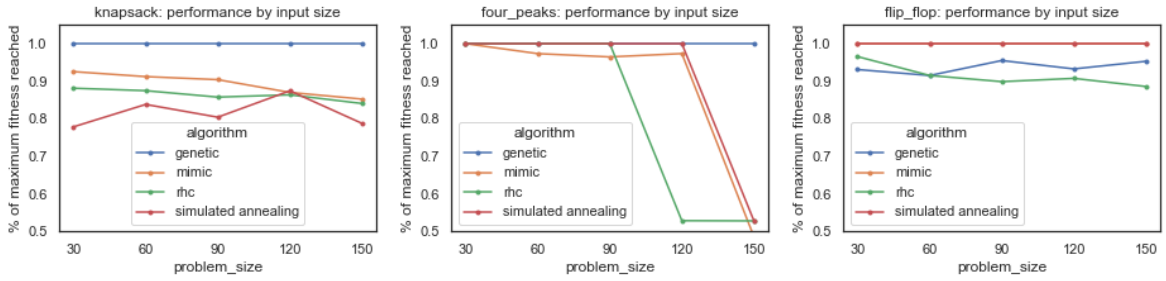



***Figure 13**: Best fitness reached as a ratio of maximum fitness for varying input sizes.*

## Neural Network Weight Optimization

I then applied the Randomized approaches to optimize the weights for a Neural Network with a single layer of 50 hidden units using a RELU activation function. This was the same configuration as the Neural Network I used in Assignment 1. The Neural Net attempts to classify hand-written digits 0-9. As a baseline, Stochastic Gradient Descent ("SGD") produced Train Set accuracy of 100% and Validation Set accuracy of 97% in 0.14 seconds, so this is a tough benchmark to beat. Unlike the discrete valued toy problems of 100 input size, this is a continuous value optimization for 3,750 weights (64 features * 50 units + 50 bias + 50 units * 10 outputs), so it's a larger and more complex space to search. Simulated Annealing and Genetic

were able to produce Train Set accuracy >= 90%, but failed to generalize as well and were much more expensive to train. RHC was prohibitively expensive to train with a linear loss curve, and was unable to achieve strong performance in < 1 hour.

| Optimization | Parameters | Train Accuracy | Validation Accuracy | Clock Minutes |
|---|---|---|---|---|
| Stochastic Gradient Descent | NA | 100% | 97% | 0.002 |
| Random Restart Hill Climbing | 100 Restarts, 100 Neighbor Attempts, 5k Iters | 42% | 41% | 58.833 |
| Simulated Annealing | ExpDecay, InitTemp = 2, ExpConst = .1 | 97% | 88% | 22.650 |
| Genetic Algorithm | PopSize = 200, MutationProb = .01 | 90% | 88% | 63.687 |

***Figure 14**: Neural Network classification performance by weight optimization approach.*

SGD is able to run a smoother analysis using derivatives and is designed to avoid suboptimal local optimas, whereas the Randomized approaches are more prone to settling into them. The Randomized approaches appear to settle, with the best Train Set accuracy settling at 97% vs. SGD's 100%. A local optima for 97% accuracy on the Training Set for this problem seems to correctly classify a majority of the digits and leave two or three of the more noisy digits like 1, 8 and 9 with lackluster performance, whereas a smoother SGD search for the true optimum achieves a more uniform fit for all of the digits. *Figure 15* shows this occurring for those more noisy digits, especially the digit 8. For an analysis of why these digits are more difficult, see my previous Assignment 1 submission.

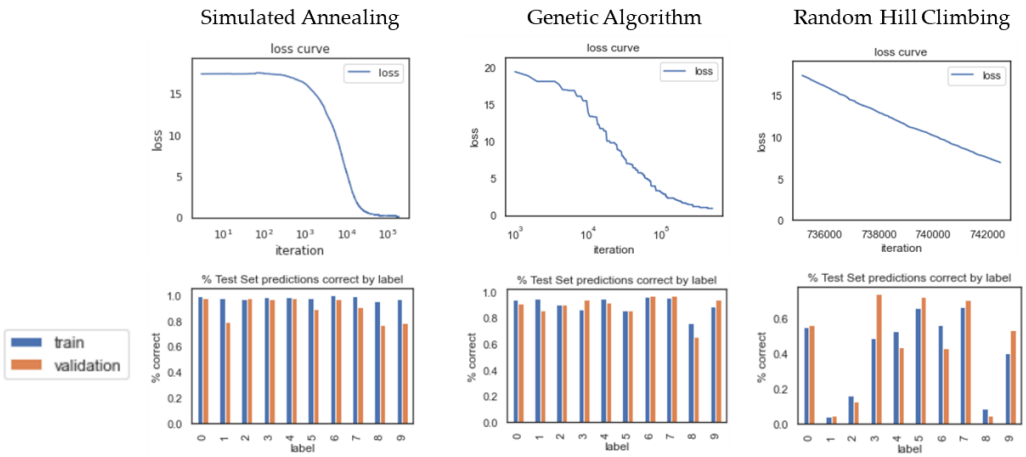



***Figure 15**: Neural Network training loss curve and accuracy by digit for each algorithm.*

Optimal parameters for these algorithms differed greatly from those of the Toy Problems, given the continuous value search and larger space. Notably, the exponential decay was much faster for Simulated Annealing and the Mutation Rate for Genetic Algorithm was much lower. Based on these results, I would prefer to use Gradient Descent to train a complex Neural Network instead of Randomized Optimization approaches.