1) Knapsack Problem

Abstract

The first problem under analysis is a variation of the knapsack problem. In the scenario at hand, there are 50 items under consideration to be included in the knapsack, with a weight capacity of 50 units. The weights of each item were developed by randomly sampling over a discrete uniform distribution of the integers {1, 2, 3…10}. The values were developed similarly, except the sampling range was done over the integers {1, 2, 3…15}. The goal is to maximize the total value of the items in the knapsack, while remaining within the weight capacity constraint of 50 units. One reason why this problem is interesting is that since the bit string has a length of 50, meaning that the total number of possible bit strings is 2^50—so the actual global optimum (or optima) cannot be directly computed without a more powerful computer. Another note of interest is that the problem involves the interplay between which bits are turned on or off (1 or 0) with the associated weights.

Note: In the analysis, GA refers to genetic algorithm, SA refers to simulated annealing, and RHC refers to randomized hill climbing. When discussing hill climbing, "search depth" means the number of neighbors examined at each iterative step in the algorithm.

Overall

The below table shows the maximum fitness achieved by each algorithm, along with the associated clock time (in seconds) and number of algorithm iterations it took to find those fitness levels.

Algorithm	Max Fitness	Clock Time	Number of Iterations
Genetic Algorithm	**182**	**111.74**	**249**
Simulated Annealing	**171**	**2082.72**	**100,000**
MIMIC	**160**	**2816.53**	**10**
RHC	**137**	**372.04**	**1000**

Overall, the genetic algorithm implementation showed the best performance. Notice that not only did it achieve a higher max fitness value than SA and RHC, but it did so with far fewer iterations and clock time. The greedy nature of RHC hinders its performance as this leads to it getting stuck in the local optima appearing the knapsack problem. After a much larger number of iterations, the SA algorithm is able to get close to the GA algorithm, but still faces the weakness that only one state is considered at a time in each iteration (hence the much larger of iterations needed to get closer to the GA result). The MIMIC algorithm shows a strength here by achieving a relatively high fitness within a very small number of iterations—only 10; however, each iteration is more computationally intensive, as can be deduced from its clock time. This could partially be due to inefficiencies in our implementation of the MIMIC algorithm (using Python in this case).

Genetic Algorithm Approach

First, we analyze how the size of the population affects this algorithm's performance for the problem.

Size of Population	Fitness	Total Number of Iterations
100	154	208
200	179	635
300	171	261
400	172	335
500	182	280
600	179	246
700	175	301
800	181	417

900	181	259
1000	181	305

Next we examine adjusting the number of "best fitness" individuals to survive each generation. The results are in the table below:

Number of individuals	Max Value Achieved	Total Number of Iterations
10	174	245
30	180	266
50	181	285
75	180	255
100	181	267

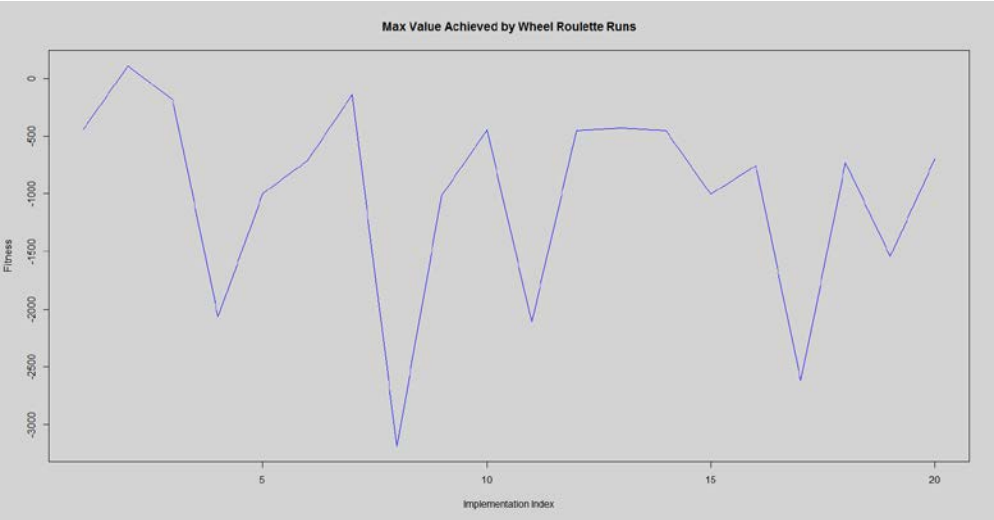
Looking at two different crossover functions, and using the other typical parameters, the following results are achieved:

Crossover Function	Max Value Achieved	Total Number of Iterations
Single-Point	181	255
Uniform	182	249

Next we vary the selection function to see the results:

Selection Operator	Max Value Achieved	Total Number of Iterations
Linear-Rank	182	249
Nonlinear-Rank	170	204
Roulette Wheel	-1796	213
Unbiased Tournament	182	243

Give that the Roulette Wheel selection adds in an extra probabilistic component in its selection of "fit" individuals, we run twenty implementations of the algorithm to see if the results can be improved:



As can be seen in the above exhibit, even after twenty implementations, the Roulette Wheel selection operator consistently leads to much poorer results than the other selection functions. Because it adds a probabilistically-weighted approach to selecting fit individuals, the Roulette Wheel method allows for a greater possibility of allowing less fit individuals to pass through generations than a straight-cutoff method, like Linear Rank shown in the table above. This could be leading to its poor results.

Simulated Annealing Approach

First, we test algorithm performance by varying the temperature parameter:

Temperature	Fitness Value
0.01	-11303
0.1	153
1	153
100	149
500	156
1000	150
2000	152
3000	150
5000	165
7500	150
10000	156

Next, we test varying the number of iterations, using a fixed initial temperature of 5000:

Number of Iterations	Fitness Value
1	-11103
10	95
100	95
1000	116
10,000	164
100,000	171

MIMIC Approach

In this method, we tested varying several parameters. First, we examine varying the percentage of samples (percentile) sent to the next distribution:

Percentile	Fitness
10%	153
30%	157
50%	137
70%	134
90%	106

As can be seen in the above table, the fitness is strongest when the top 10-30% of samples are kept after each iteration, and decreases as the percentile goes beyond that range. This is most likely due to the higher percentile including samples are not as fit, and thus having a weaker population in each iteration.

Next, keeping 30% as the fixed percentile, but varying the number of initial samples, we get the following:

Number of Samples	Fitness
100	112
500	130
1000	143
3000	157
5000	160

At the 3000-5000 range, the MIMIC algorithm gets closer to the result achieved by the GA.

Randomized Hill Climbing Approach

This approach, overall, shows less success than the other algorithms.

First we look at the results of this algorithm by varying depth search levels:

Search Depth	Fitness Value
100	146
300	133
500	128
1000	133

Next, we examine varying the number of restarts, using a search depth of 1000:

Number of Restarts	Fitness Value
10	109
100	139
300	135
500	133
1000	137

So overall, the RHC algorithm's best results are poorer than the best results of each of the other algorithms. In our problem, the structure of the domain bit string is important e.g. it matters if a bit with a high weight is turned on, as this could help push the overall weight of the bit string past capacity. This is a weakness of the RHC, as it is employing a greedy approach without much regard to the structure of the bit string and seems to be resulting in local optima in its implementations.

---

2) 4 Peaks Problem

Abstract

In this problem, our goal is to maximize the fitness function of the 4 Peaks Problem:

For N-dimensional vector X and for $T \leq N$,

$$f(X,T) = [\max(\text{tail}(0,X) , \text{head}(1,X)] + R(X,T)$$

where tail(0,X) = number of trailing 0's in X

head(1,X) = number of leading 1's in X

$$R(X,T) = \{ N \text{ if } \text{tail}(0,X) > T \text{ and } \text{head}(1,X) > T , 0 \text{ otherwise}$$

In this implementation, we use N = 150 and T = 15 (10% of N). From this, it can be derived that the global optimum is 284, achieved by T + 1 leading ones, followed by all zeros; also achievable by T + 1 trailing zeros preceded by all ones. This problem is interesting because it is easy for an algorithm to get stuck in the local maximum of all 1's or all 0's (resulting in a fitness of N). Also, the structure of the bit string is very important (as will be discussed later in the paper).

Overall

The below table shows a summary of the algorithms' performances; the details associated with each algorithm correspond with the run of the algorithm that showed the best fitness performance. The SA algorithm was able to reach a higher optimum than that shown, but this result was not consistently generated through different runs of the algorithm using the same key parameters (more explanation in the simulated annealing section below).

Algorithm	Max Fitness	Clock Time	Number of Iterations
Genetic Algorithm	**284**	**397.81**	**442**
Simulated Annealing	**150 / 284**	**1081.53**	**1 for 150 108 for 284 (See below for explanation)**
MIMIC	**150 / 284**	**45.34**	**1500**
RHC	**174**	**223.78**	**1000**

The MIMIC and genetic algorithm approaches showed the best results for this problem. Of all the three optimization problems analyzed in this paper, optimizing this one has the strongest dependency on the underlying domain structure. As simulated annealing and RHC have greedy approaches to optimization, they do not perform as well as GA or MIMIC. The genetic algorithm method has an advantage over the greedy approaches, as well, because it is able to keep up with a population of potential solution points simultaneously. As can be seen in the corresponding section below, reducing this advantage by using adequately small population levels also reduces the maximum fitness achieved by the algorithm (see below for more details).

The SA algorithm was only able to get a fitness of 150 for most the different parameter-variations tested. However, there were two instances where it was able to get of this local optimum and reach the global maximum of 284. Since these results could not be consistently reached (more details in the simulated annealing section below), 150 is considered to be the best fitness value the SA algorithm could consistently reach with the same given parameters.

Similarly, the MIMIC algorithm was able to hit 284 some of the time, but only reached the local maximum of 150 at other times. However, it was able to reach each optima with much faster clock time than the other algorithms.

Genetic Algorithm Approach

The initial run of the algorithm used a population of 1000 and was able to achieve the global maximum in 422 iterations. Fixing the maximum number of allowed iterations at 1000, but lowering the population levels to see the results yields:

Population	Fitness	Number of Iterations
100	123	1000
300	150	1000
500	284	556
1000	284	442

Thus, the algorithm was able to hit the global maximum with a population of only 500. Running more iterations at the lower population levels would most likely lead to better results for those levels. Even so, however, this demonstrates the need for an adequate population size to learn the optimal value in this situation using GA.

Simulated Annealing Approach

Simulated annealing had the poorest performance of each of the algorithms for this problem, usually only achieving a fitness of 150—apparently, a local optimum.

First, we test varying the initial temperature parameter and get the following results:

Temperature	Fitness Value
0.01	150
0.1	150
1	150
10	150
100	150
500	**284**
1000	150
2000	150
3000	150
5000	**284**
7500	150
10,000	150

As can be seen from the table above, only two of the temperature values tested were able to get the algorithm out from a local maximum of 150. This can most likely be explained by the fact that SA is only taking into account one potential solution point at a time, and does not take into account the structure of the domain. Because of this, the SA algorithm is susceptible to getting stuck in local optima, as appears to be the case here. Also of note here is that the temperature parameter adds a probabilistic component to the results. When the SA algorithm was run several more times, the global maximum fitness value was not always achieved by the temperature values of 500 and 5000, but they were in the initial run case above. This is made possible by the fact that the temperature parameter of the SA algorithm adds the probabilistic component of deciding whether or not the algorithm goes in a "exploitation" (the greedy approach) or "exploration" (risk searching through worse solutions to possibly find a better optimum) direction, and therefore, may get out of the local optimum in some of the runs of the algorithm, but not always.

Next, using a temperature parameter of 5000, we examine varying the maximum number of iterations for the algorithm using iteration values of 1, 10, 100, and 1000—all resulting in a fitness of 150. A local optimum is reached by the SA algorithm, but even with various temperature values and number of iterations, it is mostly unable to get "unstuck" from the local optimum.

MIMIC Approach

The MIMIC algorithm had mixed results in terms of maximum fitness achieved, but overall, was able to accomplish a much higher fitness value in faster clock time than the other algorithms. The initial run of the algorithm used 1500 iterations with 1000 samples, and the percentile of samples kept from each iterative distribution at 3%. This resulted in the algorithm finding the global optimum value of 284 some of time it was ran with these parameters. Running the algorithm multiple times with these parameters sometimes became stuck in the local optimum of 150.

This algorithm's key advantage over the others in this problem is its ability to have a greater understanding of the domain structure of the set of possible bit strings.

Testing lower sample levels shows the following results:

Number of Samples	Fitness Value
200	72
400	87
600	130
800	136
1000	284 / 150

Looking at the percentile number of samples kept from each iterative distribution yields:

Percentile	Fitness Value

1%	150
3%	150
5%	89
10%	54

This shows a rather extreme drop in fitness when the percentile is increased. This may be due to a higher percentage of less fit samples being included, and therefore, worse data points to pick from.

Randomized Hill Climbing

The RHC algorithm had less success here than the other methods attempted. This is most likely due to its greedy nature, and the fact that, unlike GA and MIMIC, only one potential solution point is considered at a time during each iteration of the algorithm. This is a fatal flaw, as the domain structure of the bit string is crucial to its optimization in this case, since the order of the bits in the string directly affects the fitness function value.

Looking at different depth search levels, with 500 random restarts, the following results are achieved:

Search Depth	Fitness Value
100	16
300	20
500	30
1000	169
3000	88

The depth of the neighborhood search has a significant effect of fitness value performance, as the above table shows—which the large jump between 500 and 1000 data points. However, the results are inconsistent, as the search depth of 3000 resulted in a poorer fitness value than the 1000 depth mark. This could be due to the randomization nature of RHC resulting in a "lucky" performance at the 1000 mark, and escaping a local optimum after a random restart; the 3000 mark search depth was not able to do this. These results could be different with varied number of restarts.

Thus, now we examine varying the number of random restarts using a search depth of 1000:

Number of Restarts	Fitness Value
10	31
100	39
500	169
1000	174
3000	174

The above results suggest a more consistently achieved result as the number of restarts grows larger. As this number is increased, the probability of escaping a local optimum increases.

---

3) Hamming Weight Problem

Abstract

The third problem analyzed is a variation of the Hamming Weight problem. In this scenario, the goal is to maximize the sum of a bit string of length N. Our variation uses an N value of 50. This problem is interesting because it is trivial that the answer is N, but a computer has to undergo more computational effort to figure this out.

Overall

The below table shows the summary results of the algorithms implemented.

Algorithm	Max Fitness	Computation Time	Number of Iterations
Genetic Algorithm	**50**	2.04	7
Simulated Annealing	**50**	27.78	1
MIMIC	**50**	227.4	10
RHC	**50**	0.299	10

Genetic Algorithm Approach

The GA approach was able to achieve the global optimum of 50 in multiple scenarios, but the number of iterations required to do so varied depending on its input parameters. For example, varying the population size yielded the following:

Population Size	Number of Iterations Required to Achieve Global Optimum
10	152
50	38
100	22
300	18
500	17
1000	7

Simulated Annealing

In this problem, SA showed the best performance by fewest number of iterations standards. SA was able to final the global optimum by the end of one iteration.

Testing out varying temperatures in the set {100, 500, 1000, 2000, 3000, 5000, 7500, 10000}, the algorithm was still able to reach the global optimum within the first iteration for each temperature choice. One potential reason for this is that two of the main disadvantages of SA are not as significant in this problem. First, that fact that SA only looks at one potential solution at a time is not much of a disadvantage here, since each potential data point has a neighbor that has a greater fitness value than itself (with the exception of the global optimum). Second, the fact that SA takes little account for the underlying domain structure does not play much of a role here for similar reasons.

MIMIC Approach

The initial run used a sample size of 200 with a percentile of 80% of individuals making the cutoff after each iteration. This resulted in a final fitness level of 45, below the global optimum of 50. To see if this result could be improved the following additional percentiles were tested, resulting in:

Percentile	Fitness Value
10%	47
30%	50
50%	50
70%	47
90%	40

One of the strengths of the MIMIC algorithm is that it takes into account knowledge of the structure of the problem space. In the Hamming Weight problem, this is not an advantage because the order of the sequence of the bits (e.g. the structure of the bits) does not matter; all that matters is the overall sum of the bit string. This is probably part of the reason why the algorithm is not as successful in some of the percentiles tested. However, it is still able to reach the global optimum when looking at percentiles between 30-50%.

Seeking to improve the fitness value of 40 for the 90%, we increased the number of iterations, with success:

Number of Iterations	Fitness Value
10	40
15	47
20	50
25	50

Randomized Hill Climbing

Like GA and SA, RHC was able to achieve the global optimum of 50. The initial runs of the algorithm used. As a greedy algorithm, RHC has a potential weakness of getting stuck in local optima; in the Hamming Weight problem, however, there are no local optima—50 is the global optimum achieved if and only if the bit string is all 1's. Thus, getting stuck in a local optimum is not an issue for this problem. Because of this, the advantage that SA, GA, and MIMIC have for having greater abilities to escape local optima are not as applicable in this situation.

Classification – Neural Network Implementation for Income Prediction

Abstract

The goal of this classification problem is to predict whether individuals have an annual salary above or below $50,000 (binary prediction). The train and test datasets are on an individual-record level, and contain various attribute information such as education level, hours worked per week, marital status, age etc.

In developing the model, we will train the weights of the neural network using three different algorithms—genetic algorithms (GA), simulated annealing (SA), and randomized hill climbing (RHC).

Note: Throughout the analysis for this problem, any mention of a "positive" label means an individual with greater than $50K as a salary; a "negative" label implies less than $50K as a salary.

Citation for dataset: Becker, Barry. (1996). UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/Adult] Irvine, CA: University of California, School of Information and Computer Science.

Overall

Algorithm	Testing Accuracy	Training Clock Time	Testing Clock Time	Number of Iterations
Genetic Algorithm	81.164%	11042.553	0.199	1000
Simulated Annealing	75.685%	403.854	0.420	1000
RHC	76.523%	306.122	0.408	1000

Overall, the GA approach to training the neural network weights showed the best testing accuracy, but at the cost of computational complexity, as can be seen from its training clock time above. The SA and RHC approaches showed little difference with the final number of iterations at 1000. However, as will be seen in the results below, the GA algorithm-trained neural network was able to learn the data at a greater accuracy faster (with fewer iterations) than SA or RHC. Investigating how the number of hidden layers affects the network's accuracy shows that the SA and RHC-trained networks are significantly worse when the number of layers gets larger (see below), but the GA-trained network performed roughly in the same 76-81% range when tested across different number of layers. Since the greater number of hidden layers drastically increases the complexity of the neural net, and the SA and RHC approaches are not as effective at training a larger number of weights, as they only keep up with one sample point at a time in their implementations—which could be leading to getting stuck in local optima. As the number of hidden layers, and therefore, weights increases, then this becomes more and more likely. The GA, on the other hand, is more robust to escaping local optima, due to the nature of the algorithm.

Also, the tables below show that increasing the number of iterations has a greater effect on increasing the accuracy of the SA and RHC-trained networks versus the GA approach, whose accuracy changes little when the iterations is increased. This is probably due to the fact the nature of the GA algorithm is that it keeps up with a population of potential solutions in each iteration, whereas the other two algorithms do not—so more knowledge is accomplished in a single iteration with GA, than with the other two approaches (at least in this scenario).

Genetic Algorithm Approach

Testing the GA method by varying the number of iterations, the following results are achieved:

Number of Iterations	Testing Accuracy	Training Clock Time	Testing Clock Time
10	75.881%	168.208	0.607
50	76.096%	656.933	0.322
100	77.896%	1678.062	0.229

Number of Hidden Layers	Testing Accuracy	Training Clock Time	Testing Clock Time
1	75.918%	311.116	0.235
5	81.164%	11042.553	0.199
10	77.082%	1183.938	0.371
30	80.267%	4548.147	0.8333

Simulated Annealing Approach

Testing the SA method by varying the number of iterations and hidden layers, the following results are achieved:

Number of Iterations	Testing Accuracy	Training Clock Time	Testing Clock Time
10	27.772%	3.553	0.289
50	47.632%	13.497	0.213
100	75.553%	29.790	0.231

Number of Hidden Layers	Testing Accuracy	Training Clock Time	Testing Clock Time
1	24.082%	11.494	0.317
5	75.685%	403.854	0.420
10	68.615%	23.548	0.397
30	30.387%	79.061	0.731

Randomized Hill Climbing Approach

Testing the RHC method by varying the number of iterations and hidden layers, the following results are achieved:

Number of Iterations	Testing Accuracy	Training Clock Time	Testing Clock Time
10	60.233%	4.200	0.537
50	75.071%	17.535	0.446
100	75.866%	28.033	0.336

Number of Hidden Layers	Testing Accuracy	Training Clock Time	Testing Clock Time
1	75.918%	10.337	0.528
5	76.523%	306.122	0.408
10	68.925%	22.938	0.502
30	36.744%	88.528	1.057