Randomized Optimization

Peter Lucia

October 13, 2019

1 Abstract

In this work, I aim to compare the effectiveness of four local random search algorithms, random hill climbing (RHC), genetic algorithm (GA), simulated annealing algorithm (SA), and MIMIC, against three optimization problems with discrete-valued parameter spaces. I expect the knapsack problem to highlight the advantages of the genetic algorithm because it is suited to highly complex problems and it is resilient to being trapped in suboptimal local maxima [1]. I expect the traveling salesman problem to highlight the advantages of the simulated annealing algorithm because simulated annealing is a resilient heuristic optimization technique that incorporates gradual shifts from exploration to pure hill climbing, which fit naturally within the context of the travelling salesman problem [10]. I expect the four peaks problem to illustrate the advantages of the MIMIC algorithm because it uses second order statistics to discover more information about the underlying structure of a problem, which is a more practical approach against a structurally based optimization problem such as four peaks [5]. I do not expect randomized hill climbing to be the most effective algorithm against any of the three problems because of its tendency to fall into the basin of attraction of suboptimal local maxima. I will also compare the effectiveness of the RHC, GA, SA algorithms for the task of finding optimal weights in place of backpropogation in a neural network used to classify handwritten digits from images.

2 Experiments

2.1 Knapsack

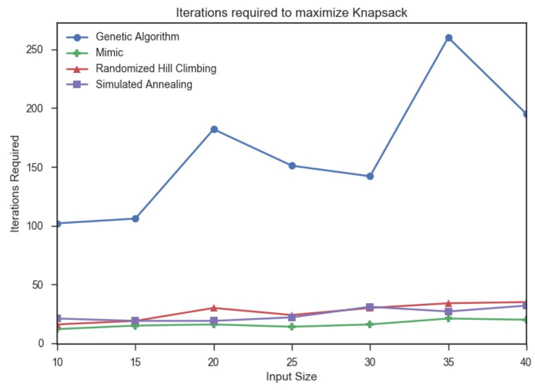
The knapsack problem is taken from [6] and [7]. The knapsack decision problem is NP-Complete, so there is no known algorithm that is both correct and polynomial-time fast in all cases [6]. Also, some instances of the knapsack problem are much easier to solve than others, thus coming up with a difficult knapsack problem is a computationally difficulty problem in and of itself [6]. These properties of the knapsack problem make it an interesting and effective tool to compare different optimization strategies. Given a list of items with some value and weight, the goal of the knapsack problem is to determine the number of each item to include in a collection so that the total value is as large as possible and the total weight is within the limit. This is summarized with the following equations.
We want to maximize

$$Fitness(x) = \sum_{i=0}^{n-1} v_i x_i$$

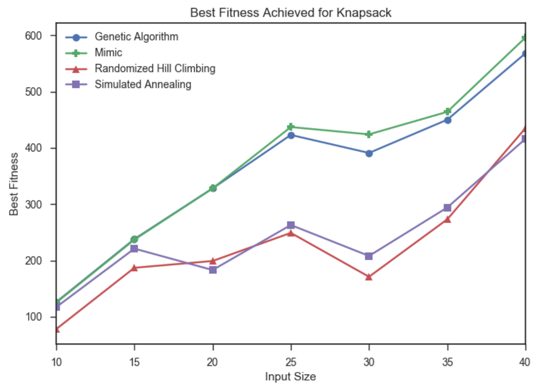
within the constraints of the weight limit. The number of copies of each item x_i is unbounded provided the weight limit is met.

$$\sum_{i=0}^{n-1} w_i x_i \leq W \text{ with } x_i \geq 0$$

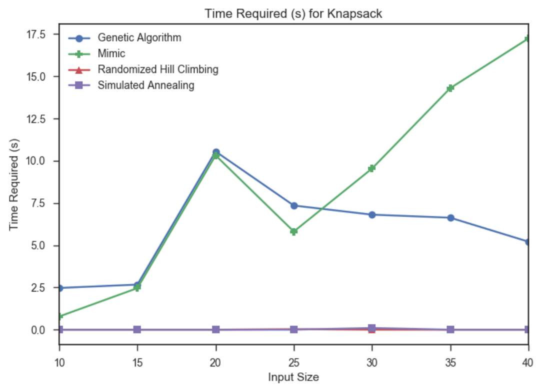
As is shown in Figure 1b, the genetic algorithm and MIMIC algorithms achieved the top two best fitness scores across all input sizes. While it required more iterations as the input size increased than the other algorithms, the overall time required to reach best fitness for the genetic algorithm did not increase with input size like it did for MIMIC, as is shown in Figures 1a and 1c. Since MIMIC required fewer iterations, but took the longest out of any algorithm to reach the best fitness, and because we assign greater importance to total time to reach best fitness instead of iterations required, the genetic algorithm would be the best choice among these algorithms against the Knapsack problem. The genetic algorithm is especially suited to tasks in which the hypotheses are complex, or tasks where the optimization goal is an indirect function of the hypothesis [1]. Moreover, in the knapsack problem, forming a set of acquired rules that can fill the knapsack to its maximum value builds a hypothesis, while keeping total weight within the limit is the indirect function of the hypothesis. Analogous to biological evolution by utilizing crossover operations to produce new offspring in a randomized, hill-climbing search, the genetic algorithm's search can sometimes transition more abruptly while creating offspring that are more unlike each parent, but the positive consequence of this against optimization problems like knapsack is that it is also less likely to fall into suboptimal local maxima. [1]. In this experiment, the best results were obtained by setting the GA population size to 200, the mutation probability to 0.1, maximum attempts to 100, and the maximum iterations to 1000. In future iterations of this experiment, it would be worth investigating in more detail how best fitness, iterations required, or time required across input sizes behaves as population size and mutation probability vary.



(a) Iterations required to maximize Knapsack



(b) Best Fitness against Knapsack

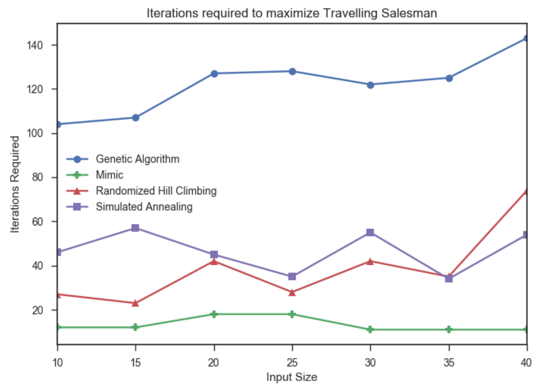


(c) Time Required for Knapsack

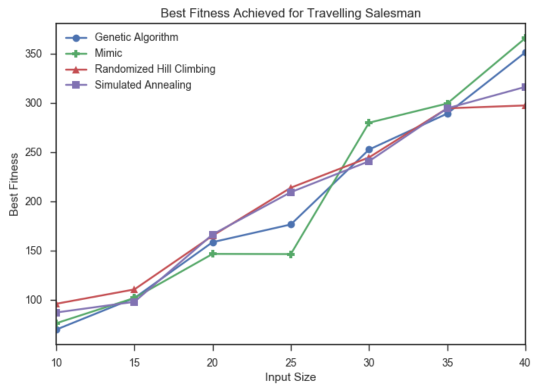
Figure 1: Algorithm comparison for the Knapsack problem

2.2 Traveling Salesman

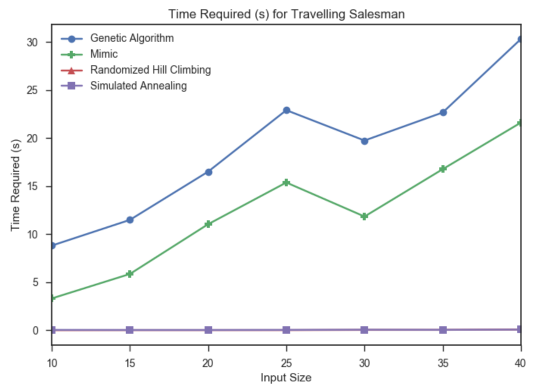
The traveling salesman problem is taken from [8]. Given a list of of cities (coordinates), the goal is to find the shortest possible route that visits each city, and finally returns to the starting point. The traveling salesman problem is NP-Complete, and therefore any polynomial algorithm that solves it would solve all problems in NP [10, 8, 9]. This difficulty is what makes the traveling salesman problem an interesting problem for our weights optimization method comparison. In this experiment, the city coordinates and distances are randomly generated for each input size. Figure 2 shows the results of each algorithm against this problem. No algorithm particularly stands out when comparing the best fitness achieved across all input sizes in Figure 2b. With regards to the iterations required, the genetic algorithm did not perform well. The MIMIC algorithm required the fewest iterations across all input sizes, but if we compare that to the time required in Figure 2c, the time required to reach best fitness increased with input size. In this experiment, the simulated annealing algorithm is the best choice against the traveling salesman problem because it reached similar best fitness scores as the other algorithms while not needing an increasing number of iterations or time to do so. It also tied with the random hill climbing algorithm for the lowest time required across all input sizes. Simulated annealing requires a slow increase in effort with increasing input size, and a higher temperature encourages exploration, even in the downward direction, preventing the algorithm from falling into suboptimal local maxima, while the smaller changes occur at lower temperatures. In [10], the computational effort as input size increases should scale as N or as a small power of N, while both the iterations required and time required with increasing input size were monotonic. In future iterations of this experiment, it would be worth investigating whether other permutations of the traveling salesman problem (e.g. patterned city arrangements and distances that are more representative of the real world) would require more computational effort on the part of the simulated annealing algorithm for increasing input size, as we should expect to see some increase in computation cost with increasing N according to [10]. In summary, simulated annealing is a widely applicable and resilient heuristic optimization technique that fits naturally within the context of the travelling salesman problem [10].



(a) Iterations required to maximize Traveling Salesman



(b) Best Fitness against Traveling Salesman



(c) Time Required for Traveling Salesman

Figure 2: Algorithm comparison for the Traveling Salesman problem

2.3 Four Peaks

The four peaks problem consists of two global maxima and two suboptimal local maxima and it is taken from [5]. The four peaks evaluation function is defined as

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

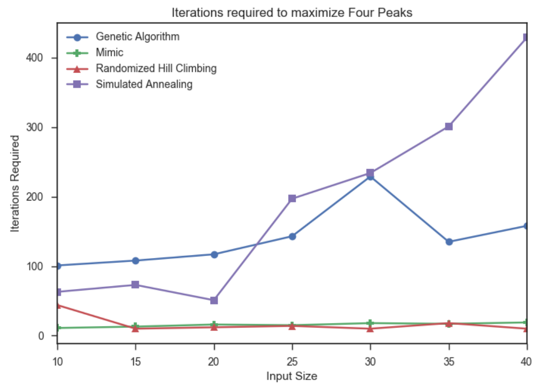
where we have

$$tail(b,\vec{X}) = \text{the \# of trailing b's in } \vec{X}$$

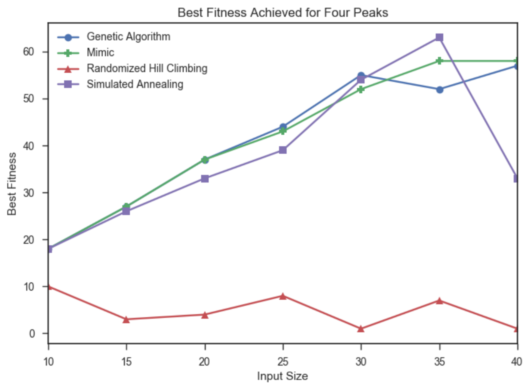
$$head(b,\vec{X}) = \text{the \# of leading b's in } \vec{X}$$

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

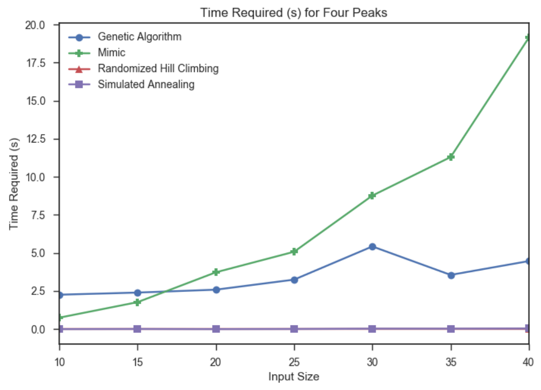
In these equations, the n-dimensional vector, $\vec{X}$, holds state information, and for large values of T, the basin of attraction of the two suboptimal local maxima becomes larger, making the problem increasingly more difficult. While simple, the structure of four peaks can trap some algorithms in local optima, which makes it an interesting and useful problem with which we can compare various randomized optimization methods. As is shown from Figures 3a and 3b, while the randomized hill climbing algorithm required fewer iterations to converge, the best fitness it achieved for all input sizes was much lower than the other algorithms, likely because it could not escape the basin of attraction for the suboptimal local maxima. The MIMIC algorithm on the other hand performed best if we consider only iterations required and the best fitness results, as it required less than 20 iterations across all input sizes to reach fitness scores similar to the genetic and simulated annealing algorithms. The genetic algorithm required over 100 iterations across all input sizes but achieved similar fitness scores as MIMIC. The simulated annealing algorithm achieved fitness scores on par with MIMIC and simulated annealing algorithms, but the number of required iterations did not scale well to larger input sizes. MIMIC uses second-order statistics to generate a random population of samples from the input space, utilizing a process of successive optimization that discovers more information about the underlying structure [5]. Therefore, the underlying structure of the four peaks problem is predisposed to an algorithm like MIMIC because the structure is relatively straightforward for it to model compared to the other problems discussed in this work. Genetic algorithms and randomized hill climbing approaches on the other hand don't do as well with structure [11]. MIMIC does come with tradeoffs, as you get a lot more information per iteration with MIMIC at the cost of the time it takes to estimate the probability distribution [11]. This is why we see the time required increase with input size in Figure 3c, and why it required less than 20 iterations across all input sizes in Figure 3a. MIMIC tends to be most useful when the when the cost of evaluating your fitness function is high, and so if we care more about requiring fewer iterations for the algorithm to complete instead of the time it takes between iterations, then MIMIC would be the best choice against this problem. On the other hand, if we care more about the time required than iterations required, the results of this experiment suggest that the genetic algorithm would be the best choice, since neither iterations required or time required increased much at all with input size while best fitness scores were always among the top two or three performers.



(a) Iterations required to maximize Four Peaks



(b) Best Fitness against Four Peaks



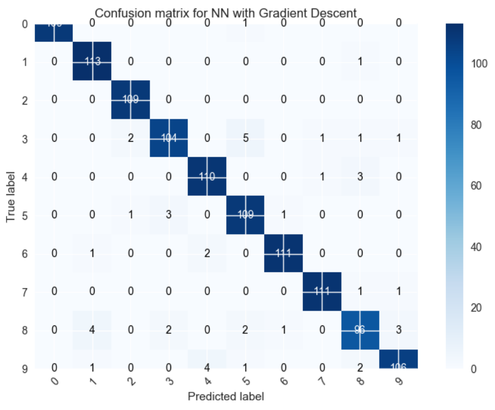
(c) Time Required for Four Peaks

Figure 3: Algorithm comparison for the Four Peaks problem

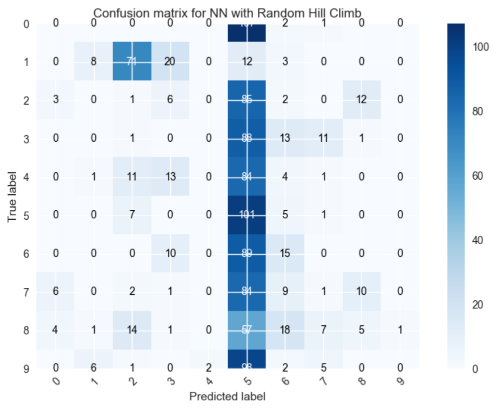
2.4 Neural Network Weight Optimization

The purpose of this weight optimization experiment is to compare the effectiveness of the random hill climbing, genetic, and simulated annealing algorithms against the task of finding optimal weights in a neural network that will classify handwritten digits from images and to see whether any of the three randomized optimization methods offer advantages against the baseline gradient descent method previously used in assignment 1.

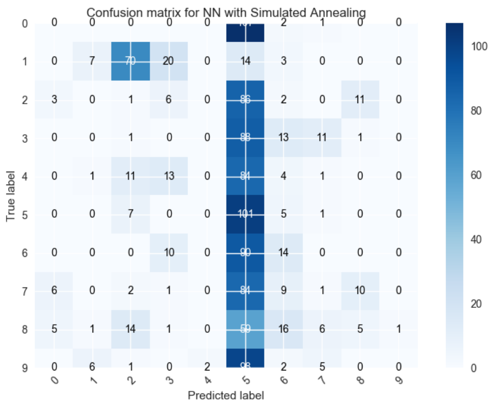
All networks in this experiment are trained using the same hyper-parameters used in assignment 1. For example, results for the previous assignment showed that a learning rate of 0.001 in combination with the relu activation function yielded the highest scores, and so those hyper-parameters are used again in this experiment. The mlrose library is used in combination with sklearn to build the neural network architectures. To check whether any differences could exist in the underlying library implementations, a second baseline experiment was performed. The confusion matrix for the gradient descent optimization method is shown in Figure 4a. After replacing gradient descent with each of the other three optimization methods, we begin to see why gradient descent is an optimal randomized optimization method for training the neural network. The confusion matrices help us understand in greater detail how well the neural network classifier performed with each of the different weight optimization methods. When classifying well, we expect to see the predicted labels match the true labels, as is shown in Figure 4a, which shows the confusion matrix with gradient descent. Both neural networks with random hill climbing and simulated annealing mistakenly predicted the label '5' too often. The neural network with the genetic algorithm predicted the label '4' too often, but showed more accuracy for digits such as 6, 3, 2, and 7. Genetic algorithms, which often succeed in finding an object with high fitness, have been applied both to function-approximation problems and learning neural network topologies [1]. This helps to explain why the genetic algorithm performed marginally better than the other optimization methods, however gradient descent clearly is the best choice. Gradient descent based methods are commonly used and proven to work empirically. [1]. For classification, gradient descent puts no restriction on the linear separability of the data, and for optimization processes such as finding proper weights in the neural network, at each step it updates the weight vectors in the direction with steepest descent along the error surface until the global minimum error (maximum fitness), is reached [1]. The genetic algorithm, which employs a randomized beam search method to find the best hypothesis, tends to move more abruptly from one hypothesis to another, sometimes creating an offspring that is radically different from the parent. The gradient descent search in backpropogation, on the other hand, moves smoothly from one hypothesis to another. Another risk with using the genetic algorithm for optimizing weights in the neural network is the problem of crowding. Crowding is a phenomenon where the diversity of a hypothesis population is reduced and progress slows due to the quick reproduction of a highly fit individual [1]. Since we need strong diversity and smooth transitions from one hypothesis to another, the genetic algorithm is not as well suited for the task here, and the gradient descent method proves to be superior.



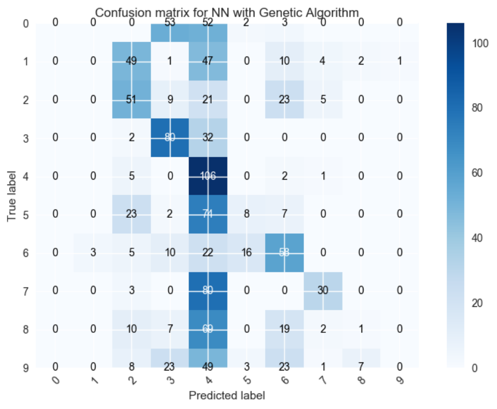
(a) Confusion Matrix with Gradient Descent



(b) Confusion Matrix with Random Hill Climbing



(c) Confusion Matrix with Simulated Annealing



(d) Confusion Matrix with Genetic Algorithm

Figure 4: Confusion Matrices for each weight optimization algorithm

In the baseline experiment, the gradient descent algorithm maintained a test accuracy of 96%, similar to its performance in assignment 1. The other optimization algorithms exhibited both training and testing accuracies of 30% or less in this experiment as seen in Figure 5. The training query time was considerably long for the genetic algorithm. The gradient descent train query time was subject to normal variation, taking several seconds longer than in the previous experiment of assignment 1. The random hill climbing algorithm exhibited the lowest training query time, but had very low test accuracy of 12%.

classifier	training time (s)	train query (s)	test query (s)	train accuracy	test accuracy
NN with Genetic Algorithm	2027.384	0.003	0.001	0.3	0.3
NN with Gradient Descent	23.736	0.003	0.001	0.99	0.96
NN with Random Hill Climb	13.702	0.007	0.002	0.12	0.12
NN with Simulated Annealing	20.279	0.003	0.001	0.12	0.11

Figure 5: Neural Network Performance with Various Optimization Methods

3 Conclusion

As expected, each of the GA, SA, and MIMIC algorithms each showed their respective strengths against the knapsack, travelling salesman, and four peaks problems, while RHC was not a strong performer against any problem. Against knapsack, the genetic algorithm achieved similar fitnesses as MIMIC but required less time to converge with increasing input size. The genetic algorithm's resilience to falling into local maxima and its affinity for problems where the optimization goal is an indirect function of the hypothesis explain its top performance against the knapsack problem.

Against the traveling salesman problem, the Simulated Annealing algorithm achieved similar best fitness scores as the other algorithms, while requiring the least amount of time to converge and less than 60 iterations to converge per input size overall. Because of its heuristic approach, the simulated annealing algorithm is a more natural fit against the traveling salesman problem. Compared to its performance against the other problems, the random hill climbing approach performed best against this problem, but did not achieve high enough best fitness scores as input size grew. It is suggested that future experiments of the traveling salesman problem model distances between cities that are more representative of the real world, and to see how the results differ if they are arranged within clusters.

Against four peaks, the MIMIC algorithm performed best if iterations required is given more importance than time required because it had among the highest best fitness scores while requiring very few iterations. MIMIC can perform well against this problem because it excels at highly structural problems by repeatedly applying a threshold against samples generated from probability distributions until the global optima are reached. If time required is given more importance against the four peaks problem, then according to these results the genetic algorithm would be a better choice than MIMIC because its time required remained relatively constant with increasing input size.

The RHC, GA, and SA algorithms did not perform as well as gradient descent for the task of neural network weight optimization used in the classification of handwritten digits. The confusion matrices of Figure 4 indicate that many labels were misclassified for all algorithms except gradient descent, and their poorer performance is confirmed by the low test accuracies shown in Figure 5. It is recommended for future iterations of this experiment to more fully explore how changing parameters affect each algorithm's performance against the three optimization problems, and how additional hyperparameter tuning might affect time required, iterations required, or best fitness achievable. In summary, the results of this experiment demonstrate that the effectiveness of each algorithm depends heavily on the problem. As a consequence, the results of this work provide empirical evidence for the no free lunch theorem, as no optimization strategy excelled against all the problems presented [12]. Finally, the results confirm that the gradient descent method is superior to RHC, GA, and SA algorithms for the task of optimizing weights in a neural network trained to classify handwritten digits from images.

References

[1] Mitchell, Tom M. "Machine learning." (1997).

[2] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12.Oct (2011): 2825-2830.

[3] Buitinck, Lars, et al. "API design for machine learning software: experiences from the scikit-learn project." arXiv preprint arXiv:1309.0238 (2013).

[4] https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

[5] De Bonet, Jeremy S., Charles Lee Isbell Jr, and Paul A. Viola. "MIMIC: Finding optima by estimating probability densities." Advances in neural information processing systems. 1997.

[6] https://en.wikipedia.org/wiki/Knapsack_problem

[7] Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. https://github.com/gkhayes/mlrose. Accessed: 5 Oct 2019.

[8] https://en.wikipedia.org/wiki/Travelling_salesman_problem

[9] https://en.wikipedia.org/wiki/NP-hardness

[10] Kirkpatrick, Scott, C. Daniel Gelatt, and Mario P. Vecchi. "Optimization by simulated annealing." science 220.4598 (1983): 671-680.

[11] https://classroom.udacity.com/courses/ud262

[12] Ho, Yu-Chi, and David L. Pepyne. "Simple explanation of the no-free-lunch theorem and its implications." Journal of optimization theory and applications 115.3 (2002): 549-570.