Assignment 2 Randomized Optimization

Lujia Zhang
CS7641
Lzhang341@gatech.edu

1. Introduction

The purpose of this assignment is to explore 3 discrete optimization problem

1. Traveling Salesman – Highlight GA
2. N-Queens – Highlight SA
3. One Max – Highlight MIMIC

implement 4 local random search algorithms

1. Randomized Hill Climbing
2. Simulated Annealing
3. Genetic Algorithm
4. MIMIC

Outside of exploring 3 discrete optimization problem, Randomized Hill Climbing, Simulated Annealing and Genetic Algorithm are used to find good weights for my assignment 1 neural network's implementation.

Implementation of the code can be found at: https://github.com/swagluke/CS7641-HW2

2. Algorithms

2.1. Randomized Hill Climbing (RHC)

Randomized Hill Climbing is a the most straightforward randomized heuristic search algorithm that helps to find the best possible solution to the optimization problem. [1]

It starts out with randomly guessing a starting point, then drifts to the best direction within a local neighborhood of that point until it reaches an optimum. This process will be repeated multiple times in order to avoid stuck in local optimum. There are 3 parameters for RHC.

Max_attempts: The max amount of attempts the algorithm runs to find the best neighbor at each step.

Max_iterations: The max amount of iterations the algorithm runs for the search.

Restarts: The number of the times the algorithm picks a new random point to start.

2.2. Simulated Annealing (SA)

Simulated Annealing is a probabilistic technique to find the optimal value of a function. Similar to randomized hill climbing but instead of always exploiting the best possible direction, simulated annealing does additional exploration with bad possible direction based on acceptance probability function. It simulates the nature behavior of heating and cooling solid. [2]. There are 3 parameters for Simulated Annealing.

Max_attempts: The max amount of attempts the algorithm runs to find the best neighbor at each step.

Max_iterations: The max amount of iterations the algorithm runs for the search.

Schedule: The function that uses to determine the value of the temperature parameter. There are 3 types of decay functions that we can choose from. *Arithmetic Decay*, *Geometric Decay*, and *Exponential Decay*. Since each decay function has different equation, thus the temperature change will be very different.

2.3. Genetic Algorithm (GA)

Genetic algorithm is random-based search heuristic algorithm. The whole idea came from Charles Darwin's theory of natural evolution, which is strongest survival. It believes that the strongest value will be passed down through multiple iterations in order to produce the best result. It starts with population of individuals, then local search around each group (mutation). Cross over from different population groups in order to produce best result. [3]. There are 4 parameters that I used for Genetic Algorithm.

Max_attempts: The max amount of attempts the algorithm runs to find the best neighbor at each step.

Max_iterations: The max amount of iterations the algorithm runs for the search.

Pop_size: The population size for each group.

Mutation_prob: The probability of mutation at each group.

2.4. MIMIC

Mutual-Information-Maximizing Input Clustering (MIMIC) is an optimization algorithm that passes the most useful information about cost function between iterations. It first generates sample from the current probability distribution then set the θ to be the nth percentile of the data set. Given only the sample for which f(x) greater than or equal to θ, it estimates the new probability distribution and repeat the process until some thresholding criteria is met. By repeating this process, it expects the best info gets pass down to produce the best results for the problem. [4] There are 4 parameters for MIMIC.

Max_attempts: The max amount of attempts the algorithm runs to find the best neighbor at each step.
Max_iterations: The max amount of iterations the algorithm runs for the search.
Pop_size: The population size for each group.
Keep_pct: The proportion of samples that MIMIC keeps after each iteration. This determines how much info gets to pass down to next iteration.

3. Hyper Parameter Tuning

For all experiments, I ran the same hyper parameter tuning from below range of values in order to find the best hyper parameter for each algorithm at each specific problem.
Max_attempts: All value between 10 to 1000 with each value as a multiple of 10.
Max_iterations: All value between 10 to 1000 with each value as a multiple of 10.
Restarts: All value between 0 to 5.
Schedule: Arithmetic Decay, Geometric Decay, and Exponential Decay
Pop_size: All value between 100 to 1000 with each value as a multiple of 100.
Mutation_prob: All value between 0.1 to 1.1 with each value as a multiple of 0.1.
Keep_pct: All value between 0.1 to 0.5 with each value as a multiple of 0.1.

4. Optimization Problems

4.1. Traveling Salesman – Highlight GA

Traveling Salesman is a well-known NP-hard problem of coming up with optimal traversal route between all points with minimum weight sum. It only allows one visit to each point as well and requires the starting point and ending point being the same point[5]

Why interesting?

Traveling Salesman problem is interesting because the most optimal traversal route isn't necessary the shortest route but the route with minimum weight sum. The one visit only rule also makes it hard to come up with the right route. Since it's NL-hard problem, there isn't any global optima but tons of local optima, which is great to test out all 4 optimization algorithms.

4. 1. 1 Set up

I decided to test out the algorithm with 8 unique points.

Local Optima

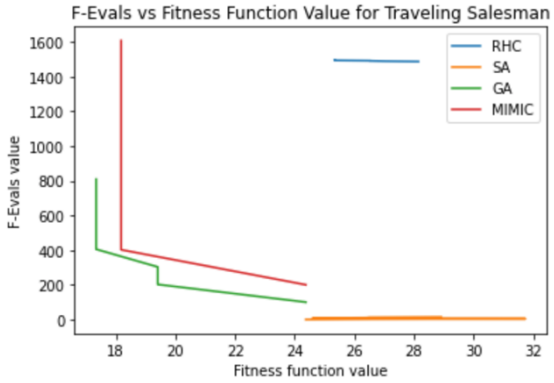
There are lots of local optima since there are tons of route, each with different weight sum.

Global Optima

No global optima due to traveling salesman as NP-hard problem.

4. 1. 2 Results

F-evals vs fitness

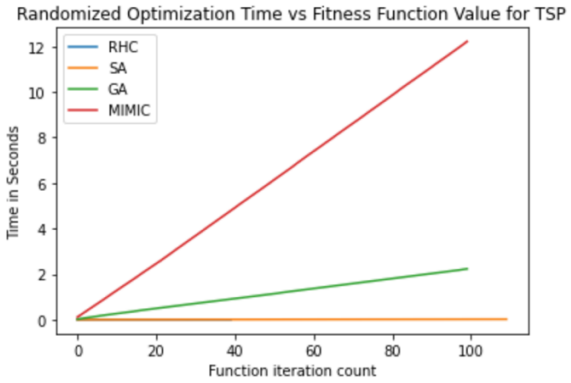


Iterations vs fitness

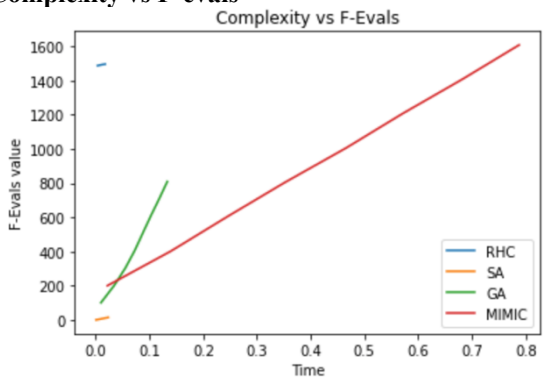
Best Hyper Parameter

Algorithm	Parameters	Value
RHC	Max Iterations, Max Attempts, Restarts	40, 40, 0
SA	Max Iterations, Max Attempts, Decay Function	110, 110, Exponential
GA	Max Iterations, Max Attempts, Pop Size, Mutation Prob	100, 100, 100, 0.1
MIMIC	Max Iterations, Max Attempts, Pop Size, Keep%	100, 100, 200, 0.1

Complexity vs Fitness



Complexity vs F-evals



Result Table

Algorithm	Best Fitness	Avg Fitness	Avg Time
RHC	-17.34	-19.34	0.0031
SA	-17.34	-20.46	0.0108
GA	-17.34	-17.57	1.1357
MIMIC	-18.18	-18.19	6.1066

4. 1. 3 Conclusion

Structure:

Traveling Salesman problem has a few constraints that make it really hard to solve. There are many different variation of routes go from point to point. And each of route can represent the local optima. Since this is a NP-Hard problem, such global optima doesn't exists. Remembering and storing useful information from each partial route/solution becomes helpful. And that's why we see good performance from MIMIC and GA. Those two algorithms benefit from the nature of the Traveling Salesman problem's structure by remembering and pass downing useful info using mutation, crossover, selecting samples and updating probability functions.

Hyperparameter Tuning

All 4 algorithms don't require crazy large or small values for their parameters.

For RHC, it' surprising to me that it doesn't even need any restart. I suspect since each partial route has lots of different variations, with enough max attempts, RHC doesn't get stuck in any local optima.

For SA, exponential decay function again worked out as the best parameter. A fast decay function helps SA to converge to the best possible fitness function the fastest.

For GA, only a small population and small mutation probability is required. Since each route provides a different fitness score, it doesn't require GA to do crazy mutation in a large group.

For MIMIC, it does require a little bigger population size than GA to make sure there are enough samples to choose from at each iteration.

Problem Size:

16 points - Traveling Salesman problem

Algorithm	Best Fitness	Avg Fitness	Avg Time
RHC	-35.21	-38.4	1.034
SA	-35.62	-40.1	1.561
GA	-34.24	-35.6	4.511
MIMIC	-34.24	-36.2	14.33

32 points - Traveling Salesman problem

Algorithm	Best Fitness	Avg Fitness	Avg Time
RHC	-69.24	-71.99	19.2453
SA	-68.12	-71.19	15.2552
GA	-63.89	-65.16	143.259
MIMIC	-63.98	-65.24	56.811

As problem size increase, we see that it takes much longer for each algorithm to run. GA and MIMIC's good performance are more obvious as problem size increase since the avg fitness and best fitness score are much better compare to RHC and SA. I choose 8 as the problem size since all algorithms were able to run in a reasonable time and achieve the close enough fitness score. It's much easier to compare model performance since we have to consider one less metrics (Best fitness). Also, quick run time allows researchers to compare result much easier since we don't have to sit around to wait.

Performance change with hyperparameter:

For all algorithms, having more max_iterations and bigger max_attempts will cost more time to run. With all algorithm already achieving the similar best fitness score with best hyperparameter setting for each, I don't think it's worth the effort to change max_iterations and max_attempts for traveling salesman problem with 8 points.

For RHC, increasing restarts will cost more time to run but potentially leads to better performance since it allows it to avoid stuck in local optima. As more the problem size increase, having restarts become useful.

For SA, with different decay function, it affects the model performance a lot. I still think Exponential decay is the best choice here since it provides the quickest and biggest decay among all 3 decay functions.

For GA, if the problem size gets bigger, bigger population size and bigger mutation probability will lead to longer model run time, but create potential better performance as it's able to mutate and pass down more info for other groups.

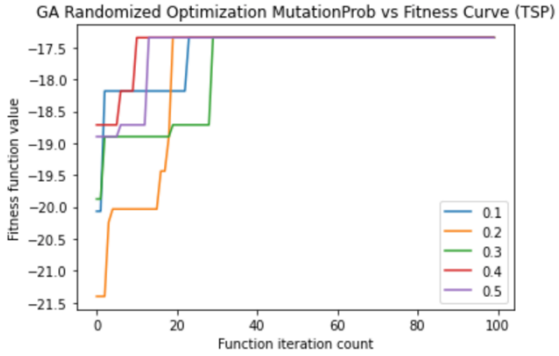
For MIMIC, having a bigger population size and bigger keep % will cost the model to run longer but again, it will possibly create better performance for bigger problem size.

Convergence Property:

For the best answer perspective, as we can see from all the graphs and table from the results' section, all 4 algorithms were able to achieve the very similar best fitness score. It took less than 40 iterations for each algorithm to get close to the best fitness score.

For the reasonable amount of run time perspective, from the complexity vs fitness graph, to no oner's surprise, GA and MIMIC took their "sweet" time to run due to the nature of their algorithm. RHC and SA again were fast to run. As problem size increases, we can see the run time dramatically increases for MIMIC and GA since there are lots of info to store and pass down. Also it becomes much harder to come up with the best route. Complexity vs f-evals graph shows that each algorithm's function evaluation score increases as time increase, but SA has a horrible f-eval score started out.

GA highlight:



As I stated in the previous section, GA worked really well for the traveling salesman problem due to it's ability to mutate among groups, store those info and crossover with other group. It helps to build a final route for those partial good routes. It was able to combine all good info from partial routes and come up with the best solution. Even though GA does require a long time to train. But the run time is still reasonable.

It's quite interesting to see how different mutation probability affects GA's performance. With bigger mutation probability, it seems like it was able to achieve the same best fitness score much faster. When we compare 0.1 and 0.5 as the mutation probability, it's interesting to me that both were able to achieve same best fitness score with a very fast convenience. But the hyperparameter tuning suggested to pick 0.1 instead of 0.5. I suspect this is due to the fact of small iteration and small max attempts, maybe the model doesn't need a huge mutation probability. This is indeed very hard to judge whether the hyperparameter choice is the best choice.

4. 2. N-Queens – Highlight SA

N-Queens is a well-known NP-complete problem of coming up with strategy to place N queen chess pieces on a n by n chessboard with the constraint of none of the 2 queens can attack each other. [6]

Why interesting?

N-Queens problem is interesting because a queen has tons of possible attack moves horizontally, vertically and diagonally on a board. It's hard for human to consider all possible attack moves even with a small number of queens on the chess board. Adding extra queen dramatically increases the problem difficulties since it generates tons of possible attack moves. Having a queen in a different spot will also dramatically change the fitness score coming out of fitness function.

N-Queens problem is great to test out all 4 algorithms' strengths and weakness because all algorithms will try to build on current best solution or pass down the most useful information to next iterations. However, most optimal solution might look completely different from current best solution.

4. 2. 1 Set up

I decided to test out the algorithm with 8 queens, which represents a 8 x 8 (64) chessboard.

Local Optima

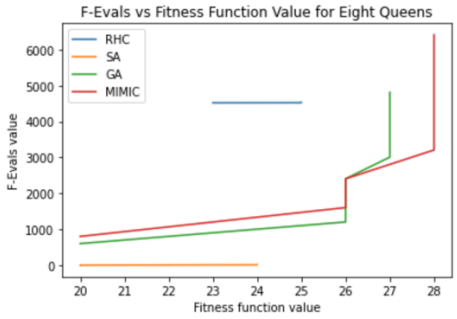
Since 8-Queens is a discrete optimization problem where each queen's position greatly affects the solution, there are tons of local optimum.

Global Optima

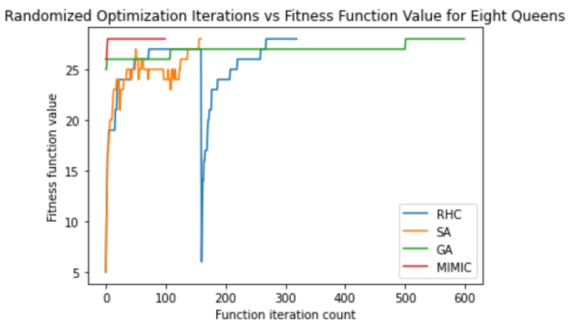
$$\binom{8}{2} = 28$$

4. 2. 2 Results

F-evals vs fitness



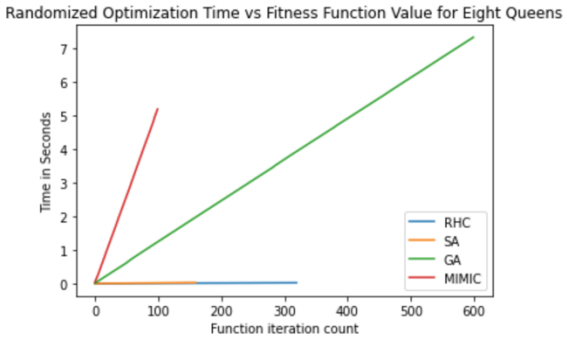
Iterations vs fitness



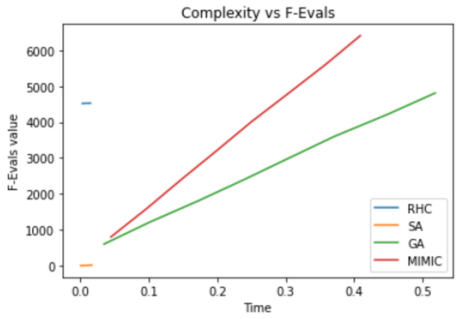
Best Hyper Parameter

Algorithm	Parameters	Value
RHC	Max Iterations, Max Attempts, Restarts	160, 160, 1
SA	Max Iterations, Max Attempts, Decay Function	160, 160, Exponential
GA	Max Iterations, Max Attempts, Pop Size, Mutation Prob	600, 600, 100, 0.1
MIMIC	Max Iterations, Max Attempts, Pop Size, Keep%	100, 100, 800, 0.1

Complexity vs Fitness



Complexity vs F-evals



Result Table

Algorithm	Best Fitness	Avg Fitness	Avg Time
RHC	28	25.22	0.0106
SA	28	24.39	0.0128
GA	28	26.98	3.685
MIMIC	28	27.75	2.593

4. 2. 3 Conclusion

Structure:

There is total 92 possible correct solutions with 28 as the max fitness score to the 8-Queens problem, which means max fitness score are spread out among the entire search space and the gaps between the local optima and global optima are small. Such structure benefits both RHC and SA well since with big enough max attempts, restarts and different temperature values, chances of stuck in any local optima is low, which leads to good performance. RHC and SA also benefit from 8-Queens 8x8 board structure set up, which allows quick calculation for the local neighbor search. Thus avg run time were both low for RHC and SA. Both GA and MIMIC took much longer to run due to the fact each has to either mutate, crossover or pass useful information between partial solution/group, however, those partial solution don't guarantee as the best solution.

Hyperparameter Tuning

For problem size 8 for the 8-Queens problem, we can see observe from the 3.2.2 Results that having too many of max iterations and too big max attempts actually hurts the RHC and SA performance since 8-Queens problem is a relatively easy optimization problem to solve. RHC also doesn't need lots of restarts since the gap between local optima and global optima are small. SA's decay function affects the perform a lot, which I will explain the SA highlight section. GA needs a much bigger max iterations and max attempts for the model to achieve the best performance. MIMIC doesn't require huge max iterations nor max attempts, but a big population size is required.

Problem Size:

16-Queens problem – Best Theoretical Score - 120

Algorithm	Best Fitness	Avg Fitness	Avg Time
RHC	119	118.12	38.95
SA	120	118.934	17.32
GA	120	117.322	312.34
MIMIC	119	116.029	474.2

32-Queens problem – Best Theoretical Score - 496

Algorithm	Best Fitness	Avg Fitness	Avg Time
RHC	494	488.396	72.58
SA	495	493.72	43.95
GA	495	493.0	537.27
MIMIC	484	479.444	635.54

As problem size increase, we see that not all algorithms can achieve the theoretical max fitness score anymore. RHC and SA's good performance are more obvious as problem size increase since both running time for RHC and SA are much shorter. I choose 8 as the problem size since all algorithms were able to achieve the theoretical fitness score as the max fitness score. It's much easier to compare model performance since we have to consider one less metrics (Best fitness).

Performance change with hyperparameter:

For all algorithms, having more max_iterations and bigger max_attempts will cost more time to run. With all algorithm already achieving the max fitness score with best hyperparameter setting for each, I don't think it's worth the effort to change max_iterations and max_attempts for 8-queens problem.

For RHC, increasing restarts will cost more time to run but potentially leads to better performance since it allows it to avoid stuck in local optima.

For SA, with different decay function, it affects the model performance a lot. For 8-Queens problem, Exponential Decay is the best Decay function. Detail comparison can be see in the SA highlight section.

For GA, bigger population size and bigger mutation probability will lead to longer model run time, but create potential better performance as it's able to mutate and pass down more info for other groups.

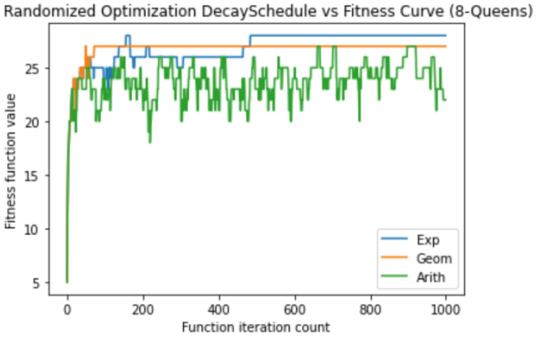
For MIMIC, having a bigger population size and bigger keep % will cost the model to run longer but again, it will possibly create better performance.

Convergence Property:

For the best answer perspective, as we can see from all the graphs and table from the results' section, all 4 algorithms were able to achieve the best fitness score of 28. Therefore, it's hard to judge which algorithm is the best in terms of getting best fitness score. As iterations increase, all fitness scores tend to increase as well.

For the reasonable amount of run time perspective, from the complexity vs fitness graph, SA took the shortest amount of time to run all the iterations, RHC was the second shortest, then MIMIC, and GA took the longest time. This isn't surprising to me since we learned that from class. Due to the nature of each algorithm, the straightforward RHC and SA will perform better in terms of time. As problem size increases, run time dramatically increases for MIMIC and GA since there are lots of info to store and pass down. MIMIC does have the smallest iteration but each iteration takes a very long time to run.

SA highlight:



As I stated in the previous section, SA worked really well for the 8-Queens problem due to it's fast run time since it only requires a small amount of iterations and fast run time for each iterations. SA also achieves the best possible fitness score of 28.

It's quite interesting to see how different Decay function affects SA as Exponential Decay is the only decay function that achieves the max fitness score. Exponential Decay worked out the best since it's the most aggressive decay function that provides the biggest decay change. With 8-queen problems having each queen affect each other so much, a quick decay function helps to find the best fitness function quickest.

4. 3. One Max – Highlight MIMIC

One Max is a well-known yet straight forward optimization problem which the goal is to evaluates the fitness of an n-dimensional state vector. [7]

Why interesting?

One Max problem is interesting for couple reasons.

1. It's straight forward and easy to test all 4 algorithms.
2. If algorithm runs correct, there shouldn't be any local optima to stuck in.
3. There is one and only global optima.

4. 3. 1 Set up

I decided to test out the algorithm with 10 vectors for the One Max problem.

Local Optima

As long as all algorithms keep searching, there shouldn't be any local optima.

Global Optima

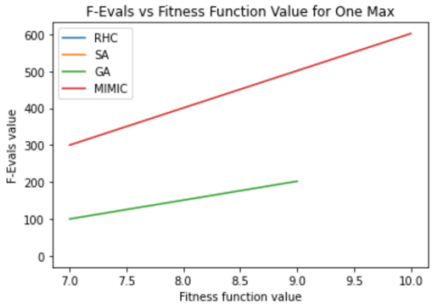
There is one global optima for the problem since the fitness function has a fixed mathematical formula.

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

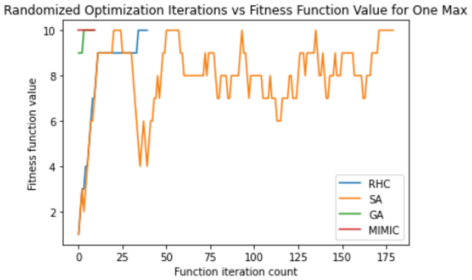
For 10 vectors of ones, it's 10.

4. 3. 2 Results

F-evals vs fitness



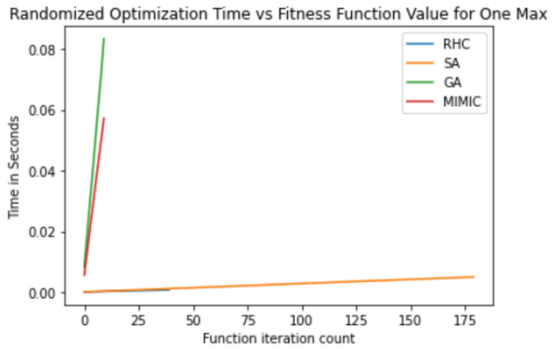
Iterations vs fitness



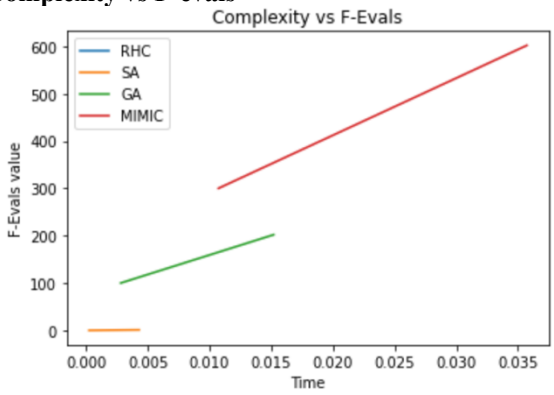
Best Hyper Parameter

Algorithm	Parameters	Value
RHC	Max Iterations, Max Attempts, Restarts	40, 40, 0
SA	Max Iterations, Max Attempts, Decay Function	180, 180, Exponential
GA	Max Iterations, Max Attempts, Pop Size, Mutation Prob	10, 10, 100, 0.1
MIMIC	Max Iterations, Max Attempts, Pop Size, Keep%	10, 10, 300, 0.1

Complexity vs Fitness



Complexity vs F-evals



Result Table

Algorithm	Best Fitness	Avg Fitness	Avg Time
RHC	10	7.925	0.000386
SA	10	7.972	0.0025078
GA	10	9.7	0.0453
MIMIC	10	10	0.031458

3. 2. 3 Conclusion

Structure:

As I stated in the problem description, One Max Problem is straight forward with the fitness function as the multiple of all vectors. So with enough iterations, each algorithm should all be able to reach the best theoretical ax fitness score. RHC and SA will benefit from such simple structure since there isn't any local optima as long as max attempts are big enough for each iteration. MIMIC will also benefit from such problem structure since it will only take so few iterations to reach the best fitness score, which make MIMIC super-efficient.

Hyperparameter Tuning

Hyperparameter tuning is quite simple for One Max since the problem structure is simple. All algorithms only use very small amount of iterations and max attempts. No crazy restarts, population size, mutation probability nor keep% is required at all since again the problem is so straight forward. No need to have any fancy tricks.

Problem Size:

One Max 100 vectors Best Theoretical Score - 100

Algorithm	Best Fitness	Avg Fitness	Avg Time
RHC	100	83.415	0.00645
SA	100	81.1716	0.00995
GA	100	97.124	1.3582
MIMIC	100	100	0.985

One Max 1000 vectors – Best Theoretical Score -

Algorithm	Best Fitness	Avg Fitness	Avg Time
RHC	1000	912.35	0.0098
SA	1000	912.35	0.0134
GA	1000	946.77	19.83
MIMIC	1000	998.123	5.345

Since One Max problem is so simple and straight forward, as problem size increase, all algorithms are still able to get the correct theorical max fitness score. RHC and SA are fast as always. GA took a significant time to run. I think I just have to readjust the hypermeter for GA in order to make it run quicker.

It's hard to judge what an ideal problem size is for One Max since it's such a simple problem. I will stick with my original choice of 10 since it was so fast to run and still shows the performance difference between all 4 algorithms.

Performance change with hyperparameter:

For all algorithms, having more max_iterations and bigger max_attempts will cost more time to run. With all algorithm already achieving the max fitness score with best hyperparameter setting for each, I don't think it's worth the effort to change max_iterations and

max_attempts for One Max problem with 10 vectors of ones or zeros.

For RHC, there is no need to have any restarts since there isn't any local optima to be stuck with.

For SA, sticking with exponential decay function seems like a good choice since there isn't any local optima. Using other decay function will probably hurt the model performance.

For GA, bigger population size and bigger mutation probability will lead to longer model run time but create potential better performance as it's able to mutate and pass down more info for other groups. As problem size increase, such changes are required.

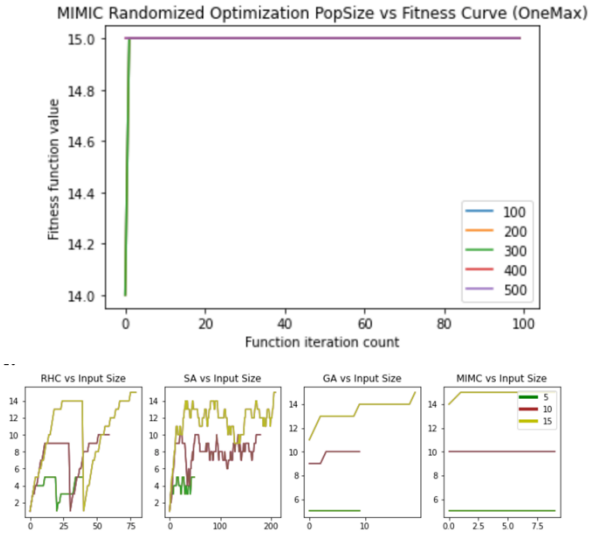
For MIMIC, as problem size increase, such changes are required, having a bigger population size and bigger keep % will cost the model to run longer but again, it will possibly create better performance.

Convergence Property:

For the best answer perspective, as we can see from all the graphs and table from the results' section, all 4 algorithms were able to achieve the best fitness score of 10. As iterations increase, all fitness scores tend to increase as well. MIMIC does outperform all algorithms with ridiculous high avg fitness score.

For the reasonable amount of run time perspective, from the complexity vs fitness graph, RHC and SA again are fast on their feet in terms of running the models. MIMIC converges so fast to score the best fitness score with only a few iterations.

MIMIC highlight:



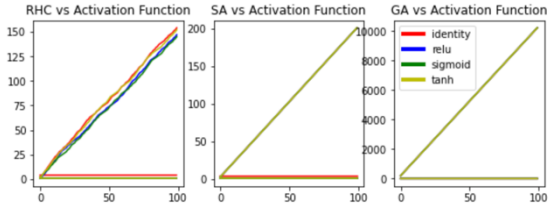
As I stated in the previous section, MIMIC worked really so well for One Max function with crazy high avg fitness score and few iterations. Even as input size increases, MIMIC is still able to maintain high fitness score without much effort. All the outstanding performance comes from MIMIC's ability to update the probability distribution function and selecting good samples using the function. It truly remembers and stores info from one iteration to next. It's actually learning how to solve the problem by passing down the most useful info. Indeed very impressing when we compare the results.

5.1. Neural Network

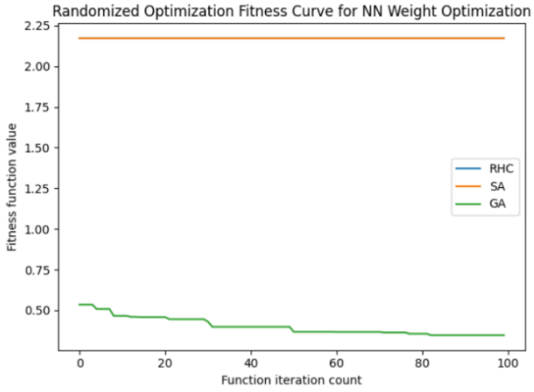
For this part of the assignment, we are using RHC, SA and GA to optimize the neural network that we built for assignment 1. Instead of using backprop to find the best weights for NN, our optimization algorithms will come in to help us find the best weights.

5. 1. 2 Results

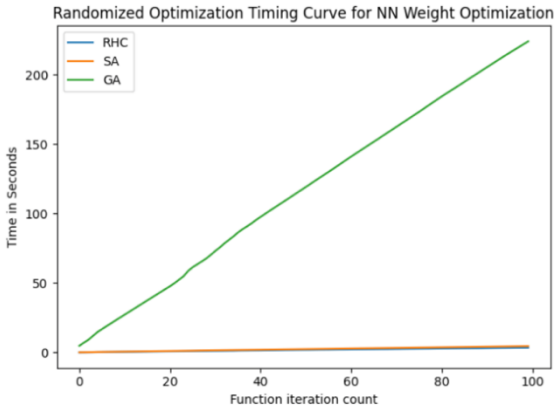
NN Activation Function Weights vs Fitness Curve



Fitness Curve vs NN Weight Optimization



Time Complexity Curve for Weight Optimization



Best Hyperparameter

Algorithm	Parameters	Value
RHC	Restarts	0
SA	Decay Function	Exponential
GA	Pop Size, Mutation Prob	100, 0.1

Result Table

Algorithm	Activation Function	Train/Test Accuracy	Loss Function Value
RHC	ReLU	74.73%, 74.74%	0.5871
SA	ReLU	74.73%, 74.74%	0.5871
GA	Sigmoid	78.66%, 76.44%	0.3837
Backdrop	Logistic	88.8%, 83.5%	0.2268

5. 1. 3 Conclusion

Best Randomized Optimization Algorithms

GA is the clear winner among the optimization algorithms since it was able to achieve the highest train and test accuracy. From the convergence time perspective, GA does take the longest to run due to the nature of the algorithm. GA's time complexity also follows a close to linear relation with iterations count.

GA outperforms RHC and SA since GA benefits Neural Network's structure. GA is able to pass down useful information about the weights through different iterations, and that's perfect for Neural Network since weights from different hidden layer affects the output so much.

Possible improvement for GA to get better results can be made by trying out more learning rate and mutation probability.

Compare with Backdrop

Unfortunately, none of the randomized optimization algorithms are able to beat backdrop in terms of train and test accuracy. The results aren't even close actually. I think this is caused by all the complex connections between neural network layers. A small change in one of the hidden layer weight will end up changing the results by a lot, which makes the global optima really different from the local optima. And none of the optimization algorithms are able to keep track of all weight changes. Therefore, make is hard to generate good results like backpropagation.

Backdrop Advantage

Combing Forward pass and backpropagation, it makes a great bidirectional way to update the Neural Network's weights, thus makes it so good to figure out the right weight for the hidden layer. Backpropagation is great for continuous and smooth space since it will be easy to calculate the derivate for each point and find the best weight. It becomes hard for backpropagation to run if the space isn't smooth.

Optimization Algorithm Advantage

Optimization Algorithm has it advantage as well. When the search space is discrete space, different optimization algorithms are able to focus on the best parameter for the given fitness function. It's straight forward and do normally generate good results when right algorithm is picked for the right problem with a good fitness function.

6. Conclusion

After running experiments for 3 different optimization problems – Traveling Salesman, N-Queens and One Max with 4 different optimization algorithms – RHC, SA, GA and MIMIC, it's quite clear that different algorithms perform well in different situation. And the tradeoff is easy to spot as well.

For problems where there aren't many local optima or the gaps between global optima and local optima are small, both RHC and SA do really well since both run very fast and produce great results.

For problems where the relationship between features matter more than the feature values, both GA and MIMIC work really well since it has the ability to store and pass information from iterations to iterations.

But there are tradeoffs for sure, time/space and performance are once again the tradeoff here. Given longer run time and more memory/space needed for GA and MIMIC, generally a better performance in terms of fitness score can be expected. On the other hand, RHC and SA benefits from its lightweight calculation so it leads to quick run time.

With that being said, it's very hard to determine which randomized algorithm is actually the best. It all depends on the type of optimization problem and fitness functions. Each algorithm has its own advantage and there is tradeoff between time/space and performance.

7. References

[1] “Hill climbing”, Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Hill_climbing. [Accessed: 14-Mar-2021].

[2] “Simulated annealing”, Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Simulated_annealing. [Accessed: 14-Mar-2021].

[3] “Genetic algorithm”, Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Genetic_algorithm. [Accessed: 14-Mar-2021].

[4]“Mutual-Information-Maximizing Input Clustering”, [Charles L. Isbell, Jr]
Available: https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf. [Accessed: 14-Mar-2021].

[5] “Travling Sales Problem,” Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Travelling_salesman_problem . [Accessed: 14-Mar-2021].

[6] “Eight queens puzzle,” Wikipedia.[Online].Available: https://en.wikipedia.org/wiki/Eight_queens_puzzle. [Accessed: 14-Mar-2021].

[7]“OneMax-Problem”mlrose.[Online].Available: https://readthedocs.org/projects/mlrose/downloads/pdf/stable/ . [Accessed: 14-Mar-2021].