CS7641 Assignment 2: Randomized Optimization

Xiaoming Su

Introduction

This assignment aims to analyze the performance of four local random search algorithms:
1. Randomized Hill Climbing (RHC)
2. Simulated Annealing (SA)
3. Genetic Algorithm (GA)
4. Mutual Information Maximization for Input Clustering (MIMIC)

The strength and weakness of different algorithms are analyzed and the parameters for each problem are optimized as well.
There are 2 parts in the report. The first part explores the application of RHC, SA and GA on generating weights for a neural network for classification problems. The second explores 4 different problems over discrete valued parameter spaces using all four algorithms. The problems are as follows:
1. Travelling Salesman
2. K Coloring
3. N Queens
4. Knapsack

1. Learning Weights for Neural Network

RA, SA and GA were applied to learn weights of a neural network for classification problems. The spambase dataset is the major dataset that is analyzed in this part. Two more datasets, wine-quality and pima-indians-diabetes, were also studied in order to get a detailed comparison of these three algorithms.

Name of Database	**Spam**	**Wine quality**	**Pima Indians diabetes**
Number of Attributes	57	11	8
Number of Instances	4601	1599	768
Number of Classes	2 (binary)	9	2 (binary)

Table 1. Database Information

Learning weights for neural network is a problem to finding weights that can fit the dataset best, which is to maximize the fitness function. The fitness function is represented by the inverse of the sum of square error. The weights in a neural network are continuous and real-valued, thus this problem is a global continuous optimization problem. The solution of continuous optimization problem is based on variable neighborhood search. All three algorithms can handle this problem successfully.
How parameters can influence the performance for each algorithms is discussed first. Then the performance of all three algorithms are compared.
The neural network for the spam database has 11 input nodes, 2 hidden layers and 1 output node. The neural network for the pima database has 8 input nodes, 2 hidden layers and 1 output node The neural network for the wine database has 11 input nodes, 2 hidden layers and 9 output nodes.

1.1 Genetic Algorithm (GA)

To encode a problem using Genetic Algorithms, one needs to address some questions regarding the initial population, the probability and type of crossover, the probability and type of mutation, the stopping criteria, the type of selection operator. All of these parameters and operators affect the performance of a GA and they are inter-related, i.e., they form a system [1]. This problem was implemented using Agagail library with some changes. The parameters analyzed are population size, crossover probability and mutation probability.

1.1.1 Population Size

1.1.1.1 Experimental Results

The population size was calculated by population percentage times the size of the training instances. Population percentage was chosen to be 10%, 30%, 70% and 90%. The error was the sum of square error between the desired label and neural network output value (0-1). For classification result, if the value was between 0-0.5, the class was set to False (0); the class was set to True if the output value is between 0.5-1. Figure 1 shows the error change in terms of iteration.

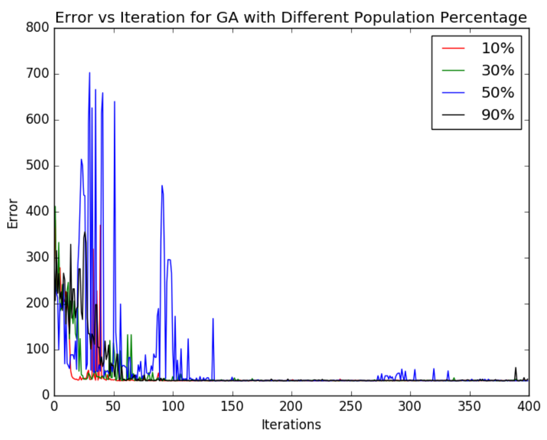


Figure 1. Error vs Iteration

From figure 1 we can find that if population size is really large, then more randomness is introduced and may cause sudden large error. With a population size of about 10% of the training size and that is about 5 times of the dimension (number of attributes) of the problem leads to the fastest convergence. The percentage of correct classification are all the same: training set: 98.193%, testing set: 97.668%.

From Figure 2, we can find that the training time becomes longer with the increase of the population size. But the relationship is not linearly and cannot be summarized based on simple observation.

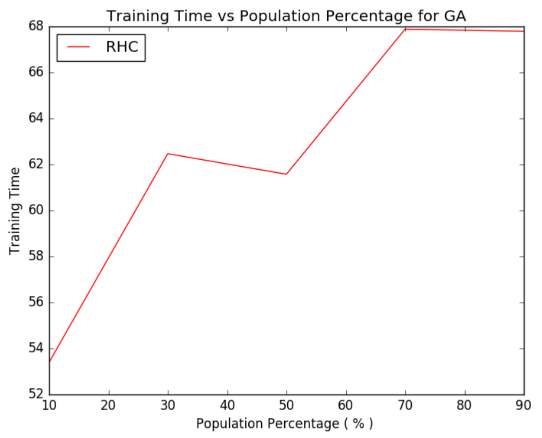


Figure 2. Training Time vs Population Percentage

1.1.1.2 Discussion

Though I didn't find any obvious relationship between performance and population size in this particular problem, population size can affect the result of many different problems. For example, the population size needed for building block problem to get a correct result is proportional to the number of building blocks in the problem [2].

Researchers usually argue that a "small" population size could guide the algorithm to poor solutions, and a "large" population size could make the algorithm expend more computation time in finding a solution [1]. Thus the problems whose results are affected by population size face a tradeoff between performance and training time.

We might want to ask for a specific problem, what is a good population size to start with? Storn and Price recommended a population size of 10 times the number of dimensions -- e.g. population size = 100 for a ten dimensional problem [3]. Besides this recommendation, there are some other consideration. In conclusion, the population size needed is problem-specific. What's more, remember the "No Free Lunch Theorem": there does not exist a "best choice" of parameters for all (or even most) of your test-instances.

1.1.2 Crossover Probability and Mutation Probability

1.1.2.1 Experimental Results

The crossover size was calculated by crossover probability times population. Crossover probability was chosen to be 10%, 30%, 70% and 90%. Figure 3 shows the relationship between the error and iterations. From the figure, we can find that the rate of convergence increases with the increase of crossover probability.

Figure 4 shows the relationship between training time and crossover probability. As the crossover probability increases, the training time also increases.

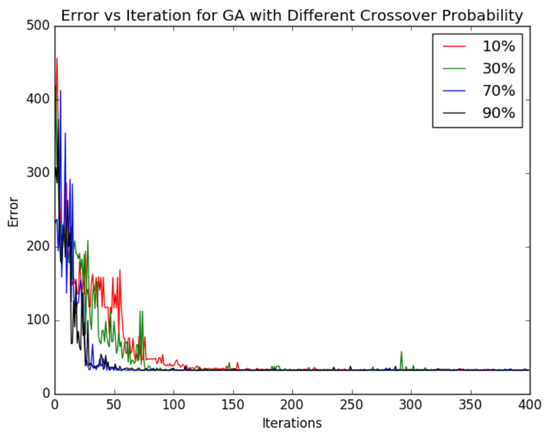


Figure 3. Error vs Iteration for GA with Different Crossover Probability

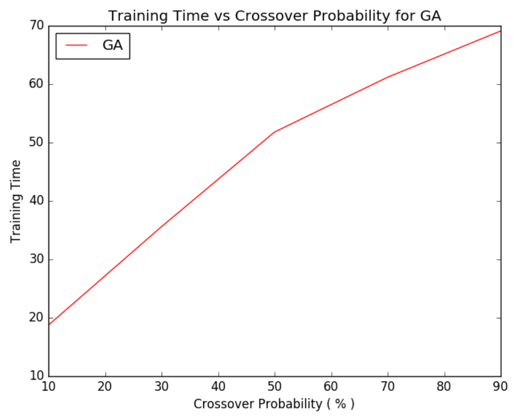


Figure 4. Training Time vs Crossover Probability

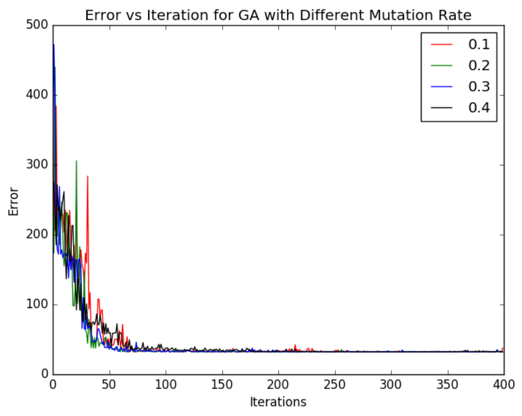


Figure 5. Error vs Iteration with Different Mutation Rate

The mutation size was calculated by mutation probability times population. Mutation probability was chosen to be 10%, 20%, 30% and 40%. Figure 5 shows the relationship between the error and iterations. The mutation size was calculated by mutation probability times population. Mutation probability was chosen to be 10%, 20%, 30% and 40%. Based on Figure 5, there is no obvious relationship between the rate of convergence and the mutation probability.

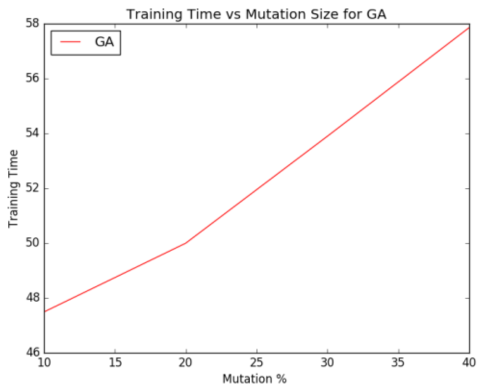


Figure 6. Training Time vs Mutation Probability

From Figure 6, we can find that the training time increases with the growth of mutation probability.

1.1.2.2 Discussion

Crossover and mutation are two of the most important genetic operators found in genetic algorithms. There has been much debate as to which of these is practically and theoretically more effective.

Some differences had already been observed [4]:

- Crossover exploited existing genetic material, while mutation explored for newer material.
- Crossover proved less and less effective as the similarities between organisms increased as the population converged, while mutation became more and more necessary as variation in the population reduced.
- Crossover tended to preserve, while mutation seemed to destroy.

In GA, mutation operators are mostly used to provide exploration and crossover operators are widely used to lead to converge on the good solutions find so far (exploitation). Consequently, while crossover tires to converge to a specific point, mutation avoids to convergence and explore more areas. Thus it makes sense that the experimental results show that a larger crossover probability leads to convergence faster.

It's hard to tell directly what a good size of mutation or crossover is. The best value of these two operators are very problem specific. But we do know in general that if exploration is preferred, the mutation probability can be set to a "large" number; if exploitation is desired, the crossover probability can be set relatively "large".

1.2 Random Hill Climbing (RHC)

Random hill climbing provide a solution to prevent the drawback of simple hill climbing that it's easy to get stuck in the local optimal instead of global optimal. The number of restarts is evaluated for RHC.

1.2.1 Experimental Results

Figure 7 shows the relationship between the error change vs iterations with different numbers of restarts. There is no obvious relationship between the converging rate and the number of restarts based on the expriment. I repeated the experimens many times, I still cannot get relationship based on the result I got. I thought that it may because of the nature of the dataset itself.

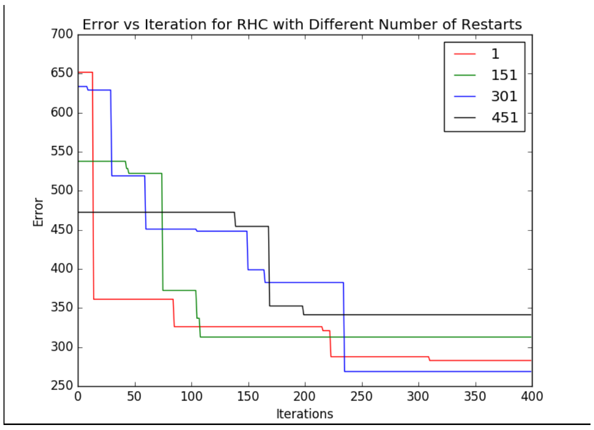


Figure 7. Error vs Iterations

Figure 8 shows the relationship between the training time and the number of restarts. In general, the training process is really quick for RHC compared with other algorithms. As the number of restarts increases, the training time should also increase linearly, however, based on the data, the linear relationship is not really obvious.

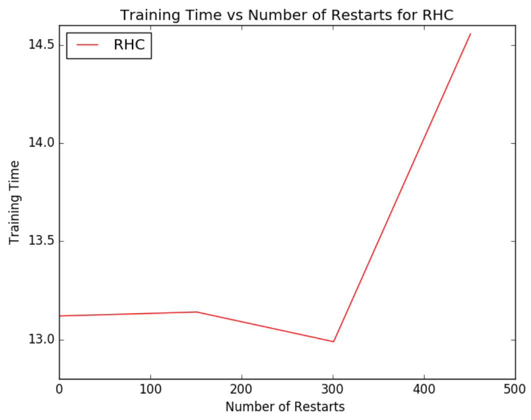


Figure 8. Training time vs Number of restarts.

1.2.2 Discussion

Random hill climbing is restarting simple hill climbing multiple times. With multiple tries, it's possible to get a global optimal instead of getting stuck in the local optimal. However, it's a "random" restarts, which means that if the basin of attraction is mall, it is also possible that the global optimal cannot be found.

1.3 Simulated Annealing (SA)

Simulated annealing is a single state method. Unlike simple hill climbing, the candidate with worse fitness function is always rejected, the SA accepted a "worse" candidate with certain probability:

$P = \exp(-\Delta E / T)$.

The temperature is a variable whose value decays during the progress of the optimization. Therefore, SA accepts "worse" candidate with a relatively large probability (explore, random walk), and as iteration number increases, it searches the solution space less permissively.

1.3.1 Experimental Results

For this problem, the influence of cooling exponent is analyzed with the initial temperature (1E11) unchanged. Figure 9 shows the error change with increased iterations with different cooling exponent. With a larger cooling exponent, the temperature decreases slower, that it is more like "random walk". The error decreases slowly. When the temperature cools fast (large cooling exponent), it converges to an optimum really fast. When the cooling exponent is really small, SA will be more like HC.

Based on Figure 10, no direct relationship between the training time and cooling exponent was observed. And that is reasonable, because SA is single state method, the cooling schedule doesn't influence training time that much.

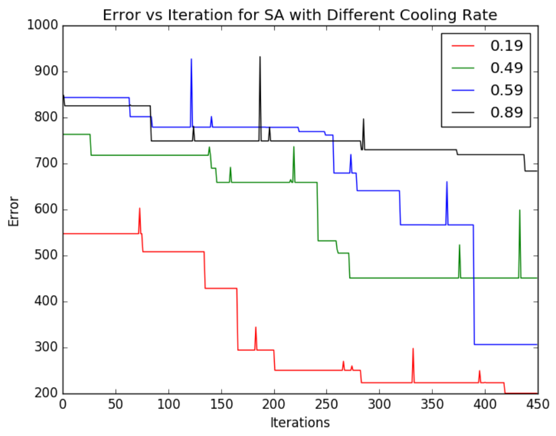


Figure 9. Error vs Iteration with Different Cooling Exponent

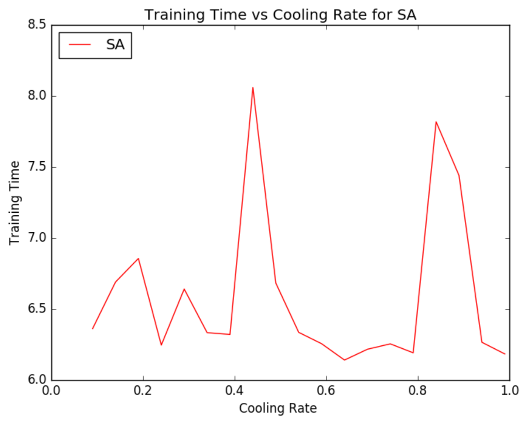


Figure 10. Training Time vs Cooling Rate For SA

1.3.2 Discussion

Simulated annealing take "worse" candidate with certain probability. It explores more at the beginning of the optimization process, then it exploits more gradually. If the temperature is really low, it's just like hill climbing. Cooling schedule influences the performance. When there are multiple local optima in the problem, it is better to explore more in order to find the global optimum instead of just a local optimum. On the other side, if SA explore more, the rate of convergence may be slow. By setting a proper cooling schedule and evaluating the trade-off between "random walk" and "hill climbing", SA can get a good result.

1.4 Summary

1.4.1 Spambase Dataset Summary

For the spam dataset, the best testing classification accuracy achieved by all the algorithms is 97.668%. On the other side, sometimes the SA or RHC can give classification accuracy at 2.332%. Notice these two numbers, they add up to 1. I repeated the experiment many times and they all get same results. So for the spam classification problem has two local optima in the solution space. If it get stuck in the local optimum rather than global optima, then the neural network will only classify 2.332% of the instances correctly.

1.4.2 Algorithms Comparison

Parameters: RHC: restarts 3

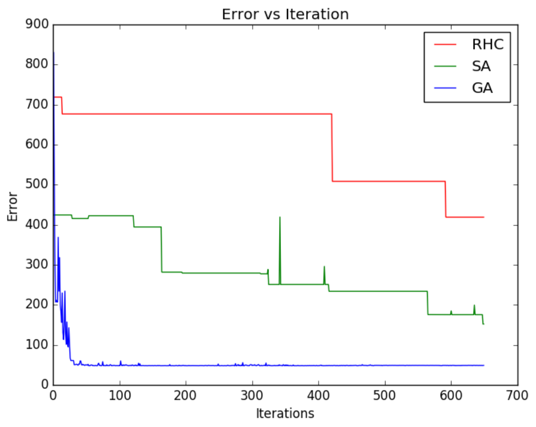
SA: 1E11, .84

GA: population 200, crossover 100, mutation 10

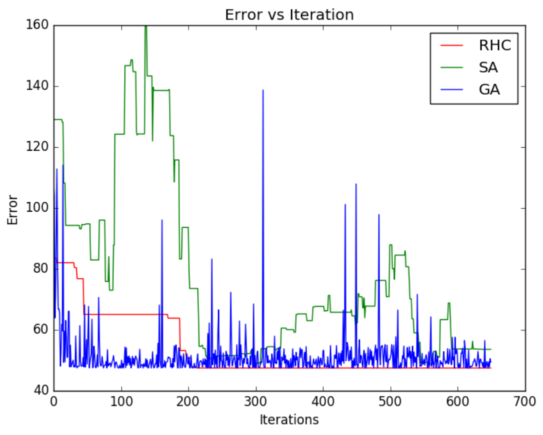
Figure 11 shows the error change of all three algorithms and the accuracy change for wine dataset. For the same algorithm, the error converges at different speed for different datasets. From Figure 11 (a) for spam dataset, we can see that the rates of convergence differ a lot. GA converges in about 30 iterations. The error of RHC and SA slowly decreases. For all the dataset, the rate of convergence of GA is the largetst. RHC is usually ranks the second. Though RHC and SA can be very similar by setting a different cooling schedule for SA.

Figure 11 (d) shows the accuracy (%) vesus iteration for wine dataset. Because of accepting "worse" candiate for a certain probability, the accuracy of SA fluctuates a lot. The GA even flucturates more than SA, which is because of mutation to explore more.

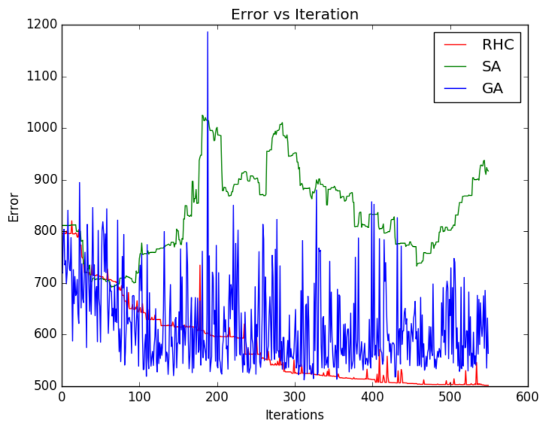
The SA is a single state method, RHC has multiple restarts (when the number of restarts is really small, RHC can be considered as simple hill climbing), GA is a population method. Both SA and GA explores and exploits, the HC only eploits. It's clear that HC is greedy by only exploiting to get a optimum (global or local). The fluctuation of error or accuracy for GA and SA shows the that they explore and try to get the global optimum. For the same problem: finding best weights for neural network, the performance of different algorithms varies from each other and the performance shows nature of the algorithm.



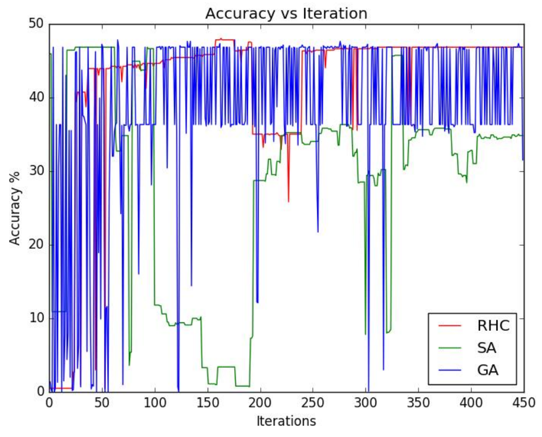
(a) Spam dataset: Error vs Iteration



(b) Pima dataset: Error vs Iteration



(c) Wine dataset: Error vs Iteration



(d) Wine dataset: Accuracy (%) vs Iteration

Figure 11. (a), (b), (c) shows Error vs Iteration for Three Datasets; (d) shows the classification accuracy for wine dataset

Figure 12 shows the training time vs iterations. The sequence of training time needed is GA>>SA>RHC. GA is a population method, it needs crossover and mutation. Thus it takes longer time for each iteration. With no training parameters changed, the training time grows linearly with the increase of iterationgs.

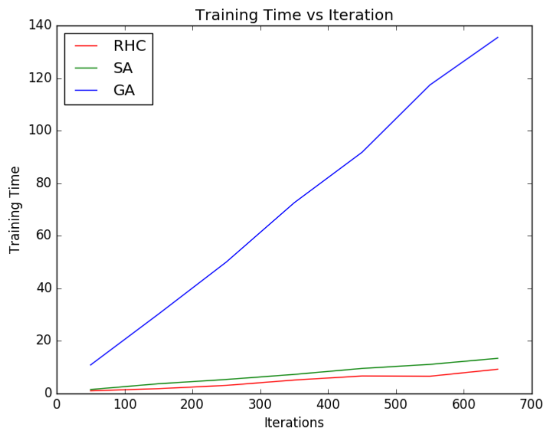


Figure 12. Training Time vs Iteration for Spam Data

2. Four Problems

2.1 Travelling Salesman Problem (TSP)

2.1.1 Problem Introduction

The travelling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science [5]. This problem was chosen to highlight GA.

2.1.2 Experimental Results and Discussion

Parameters:
RHC: iteration 200000
SA: 1E12, .95, iteration 200000
GA: population: 200, crossover 150, mutation 40, iteration 1000
MIMIC: population 200, population kept 100, iteration 1000

In order to evaluate the performance of four algorithms, the total traveling distance and time needed to solve the problem was recorded with different numbers of cities. Figure 13 shows the distance vs number of cities for the four algorithms. For this optimization problem, the shortest distance is preferred. Based on Figure 13, MIMIC gives the shortest distance when the number of cities is less than 20. As the number of cities increases, GA outperforms all the other algorithms and gets the shortest route. Thus, GA has the best performance according to the traveling distance.

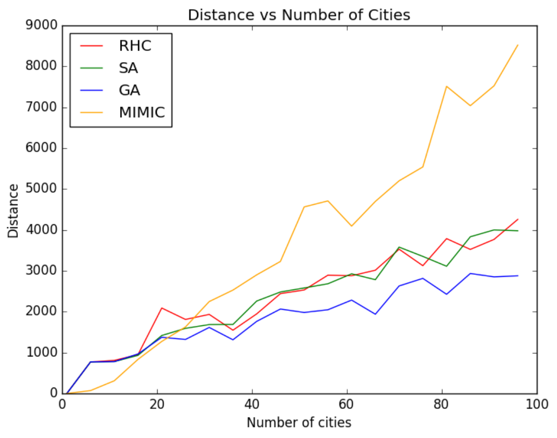
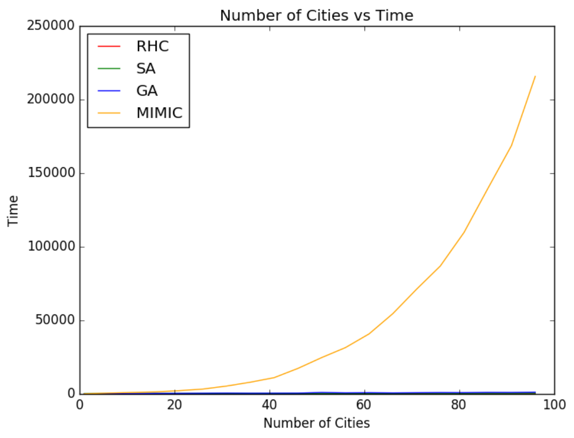
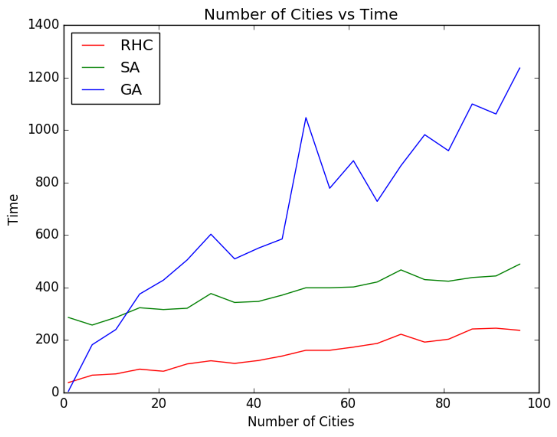


Figure 13. Distance vs Number of Cities

Figure 14 shows the change of running time with respect to the increase of number of cities. MIMIC gets the worst result. The time needed almost grows exponentially. In order to better compare the other three algorithms, Figure 14 (b) was shown without MIMIC data. The running time grows almost linearly for GA, SA and RHC. Because GA is a population method, it makes sense that it takes longer time compared with SA and RHC.
Overall, for TSP, GA has the best performance with shortest distance and good time complexity. RHC and SA have comparable performance. MIMIC performs worst.



(a)



(b)

Figure 14. Time vs Number of Cities

The reason that GA does better than the other algorithms is based on the nature of this problem. GA keeps the best population, which re good routes. What's more, when crossover occurs, it means that the two routes are changing a part of the path with each other. Thus, mating the instances which have a good tour more likely results in an overall better performance. Thus, GA can solve this problem really well.

2.2 K Coloring Problem

2.2.1 Problem Introduction

The problem is to assign K colors to N number of nodes of the graph such that no adjacent nodes have the same color. Each nodes have L number of edges [6]. This problem was chosen to highlight MIMIC, because the structure is really important in this problem.

2.2.2 Experimental Results and Discussion

Parameters: L: 3, K: 5
RHC: iteration 20000
SA: temperature: 1E12, cooling exponent 0.8, iteration 20000
GA: population 200, crossover 100, mutation 20, iteration 1000
MIMIC: population 1000, kept population 100, iteration 100
Figure 15 shows the relationship of the optimal fitness value and number of nodes N for different algorithms. All algorithms have similar fitness

values.

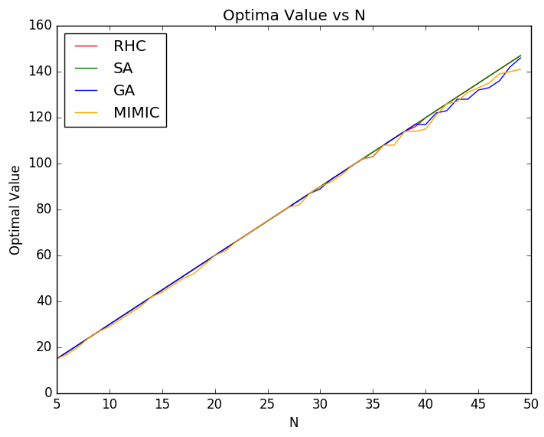


Figure 15. Optimal Value vs N

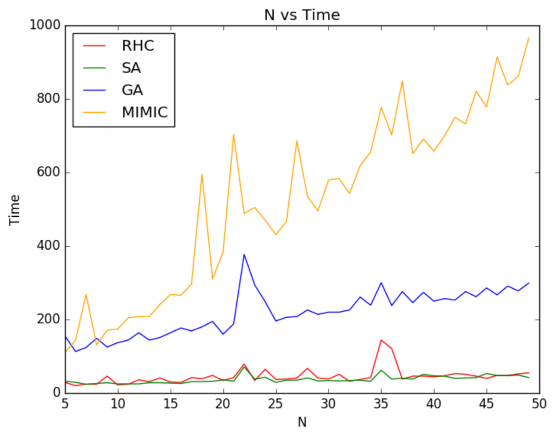


Figure 16. Time vs N

Figure 16 shows the running time for different algorithms. The running time ranks as follows MIMIC > GA >> RHC≈SA. The time grows almost linearly for all the algorithms.

The following table shows the number of experiments that each algorithms find the k coloring method successfully.

	RHC	SA	GA	MIMIC
Number of experiments	45	45	45	45
Number of experiments found k coloring method	44	45	31	23
Successful rate	98%	100%	69%	51%

Table 2. Successful Rate of Different Algorithms

The result is different from what I expected. SA is the best algorithm based on performance (accuracy) and speed. According to a paper [6], MIMIC should outperform other algorithms because it keeps structure information while the others do not. It is surprising to find that MIMIC performs much worse than GA and SA in this expriment. The real reason why MIMIC does not perform well on this problem is unclear. It's possible that this problem does not satisfy the dependency tree rule. This is especially the case when the neighbors are randomly determined for all the nodes, which may cause each feature to depend on not only one parent feature but many features, which make MIMIC hard to learn.

2.3 N Queens Problem

2.3.1 Problem Introduction

N queens problem is to place n non-attacking queens (no two queens are on the same row, column, or diagonal) on an N×N chessboard, for which solutions exist for all natural numbers N with the exception of N=2 and N=3 [7]. Just as TSP, this problem is a combinatorial optimization problem. Which means that it is to find an optimal value in a finite set of values. For this problem, the brute-force searching is not feasible due to the vast size of the set possible values. For example, if N=8, there are 4,426,165,368 possible arrangements of 8 queens on an 8×8 board but only 92 solutions. This problem was chosen to highlight SA.

2.3.2 Experimental Results and Discussion

Parameters: RHC: iteration 100

SA: temperature 1E12, cooling exponent 0.1, iteration 100

GA: population 200, mutation 10, iteration 100

MIMIC: population 200, kept population 10, iteration 5

The fitness value is the number of non-attacking pairs of queens, which means that the larger fitness value, the better performance. Figure 17 shows the optimal value (the largest number of non-attacking pairs of queens) with respect to the number of queens N by using different algorithms. All algorithms have comparable optimal value.

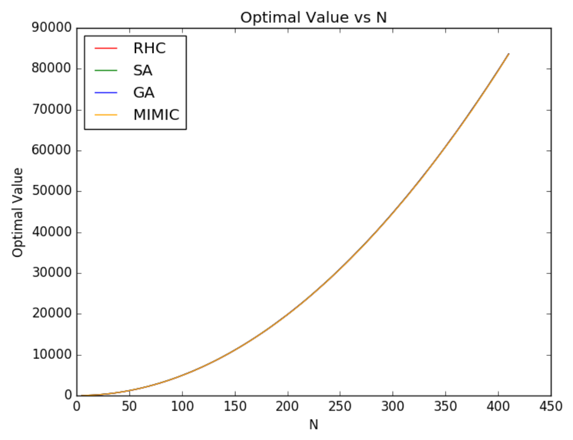
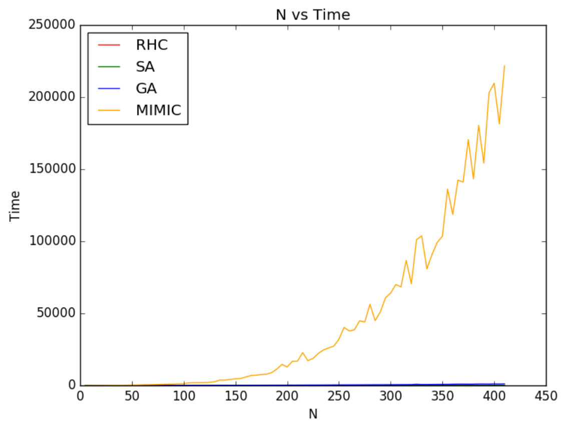
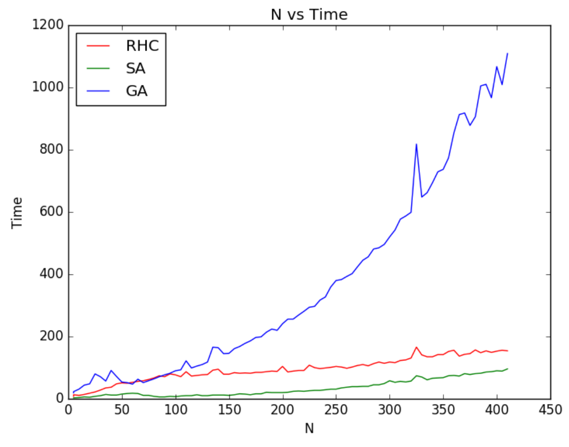


Figure 17. Optimal Value vs N

Figure 18 shows the running time needed with respect to the number of queens. Among all the algorithms, the running time grows exponentially for MIMIC. In order to compare the other three, Figure 17 (b) shows that the complexity for RHC and SA was **O**(N). The running time for different algorithms ranks as follows: MIMIC >> GA > RHC ≈ SA.



(a)



(b)

Figure 18. Training Time vs Number of Queens

Though the iteration number of MIMIC is less than the other algorithms, it's still really slow. It is because that MIMIC calculates probability distributions and mutual information. It takes long time to perform these statistical computation. With all the algorithms having similar optimal values, the algorithm that has the shortest time is the best. Thus the RHC and SA are the best ones with similar running time.

Let's think about why the very simple RHC and SA work really well for this problem. Here is the procedure of how SA works for this problem:

- Generate a random initial solution.
- Generate a "neighbor" solution, that is, based on the current configuration just change it slightly, so there's no huge jumps along the search space, we achieve this by taking a random queen and moving it a single step in a random direction.
- Accept the neighbor if simulated annealing determines so.

From the procedure, we can find that the way how SA works is a very good strategy to solve this problem.

2.4 Knapsack Problem (Extra)

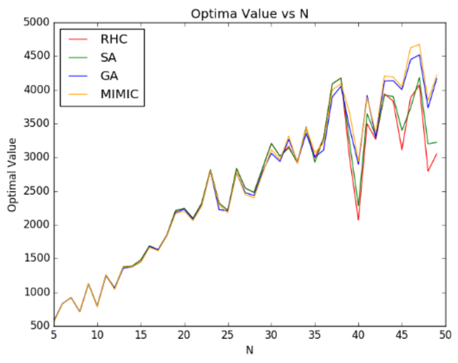
The problem is an extra one to see how the MIMIC performs in this problem because of its poor performance on K coloring.

2.4.1 Problem Introduction

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items [7].

2.4.2 Experimental Results and Discussion

Figure 19 shows the relationship between the optimal fitness value and number of items N for different algorithms. MIMIC gives larger fitness value compared with other results when N is large. MIMIC constructs of a structure that best represents the probabilistic model. Only considering the optimal value, MIMIC is the best.



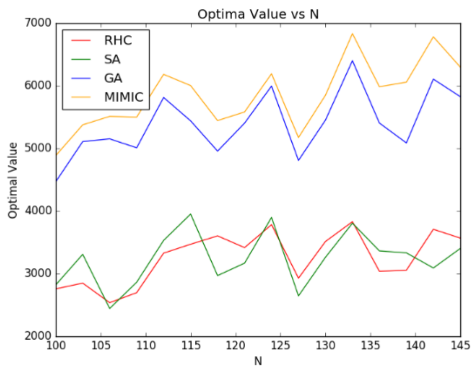


Figure 19. Optimal Value vs N

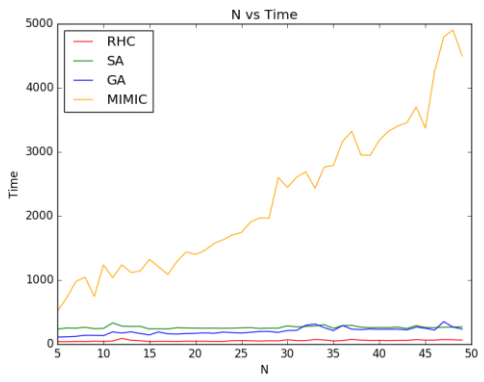


Figure 20. N vs Time

The training time ranks as follows based on Figure 20: MIMIC >> SA≈ GA > RHC.

One possible reason that GA and MIMIC have larger fitness value compared with SA and RHC especially when the numbers of items are large is that they try to capture some structure, while SA and RHC cannot.

3. Conclusion

GA and MIMIC are population method. For the problems where keeping the history can assist in creating a better optimization problem, MIMIC or GA performs better. SA and RHC have very similar performance. By setting proper cooling schedule, SA can outperform RHC. If the problem can be solved by random search in neighbor region, then RA and RHC can work really well, such as N queens problem. One drawback of SA and RHC is that they can get stuck in the local optimum instead of global optimum. What need to be mentioned is that the speed of SA and RHC is really fast because they are very easy. GA is relatively slow. MIMIC needs much less iterations than other programs, but it needs more time per run. These algorithms each work well on different types of problem. It depends on the nature of the problem and the nature of the algorithm.

Reference

[1] Diaz-Gomez P A, Hougen D F. Initial Population for Genetic Algorithms: A Metric Approach[C]//GEM. 2007: 43-49.

[2] G. R. Harik and F. G. Lobo, “A parameter-less genetic algorithm,” in Proceedings of the Genetic and Evolutionary Computation Conference, 1999, pp. 258–265.

[3] Storn R. On the usage of differential evolution for function optimization[C]//Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American. IEEE, 1996: 519-523.

[4] Senaratna N I. Genetic algorithms: The crossover-mutation debate[J]. Degree of Bachelor of Computer Science of the University of Colombo, 2005.

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

[6] De Bonet J S, Isbell C L, Viola P. MIMIC: Finding optima by estimating probability densities[J]. Advances in neural information processing systems, 1997: 424-430.

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