Randomized Optimization Algorithms

Surabhi Amit Chembra
surabhichembra@gatech.edu

Abstract
This paper explores various randomized optimization algorithms, namely Random Hill Climbing, Simulated Annealing, Genetic Algorithm and MIMIC. In part 1, the first three algorithms are used to determine good weights for a neural network built using backpropagation in assignment 1 for letter recognition dataset. In the second part, three optimization problems of "Travelling Salesman", "Continuous Peaks" and "Flip-Flop" are used to analyze the performance of the four optimization algorithms. All analysis is performed using ABAGAIL(Github,2019)[1], Jython 2.7, PyCharm IDE and Windows 10 machine.

Optimization Algorithms
Randomized Optimization algorithms are optimization techniques which do not need gradient calculations and hence can also find optimum values for non-differentiable or non-continuous functions. The four random optimization algorithms discussed in this paper are introduced below.

1.Randomized Hill Climbing(RHC): RHC is an iterative optimization algorithm that chooses an arbitrary position in the search space and moves to better/improved positions in the neighborhood of the current position, until it finds a local optima and does not step out of it. Though it can find global optima for convex problems, it's random nature of selecting initial position attributes to getting stuck in local optima in many other scenarios. This simple and fast algorithm can use restarts, helping it to start from different initial positions and thus increasing the probability of being in the basin of attraction and thus reaching the global optima.

2.Simulated Annealing(SA): SA is a probabilistic approximation method inspired by annealing in metallurgy where heating and consequent slow cooling of metal improves its ductility and makes it more workable. SA starts from a random initial position/state "s" with an initial Temperature T and samples a new state "s1" in the neighborhood of s. It moves to s1 either if fitness value improves or with a probability of $e^{(f(s1)-f(s))/T}$. Temperature(T) is gradually decreased at the rate of cooling exponent(CE). In high temperatures, SA has high energy and behaves like an exploratory random walk. This exploration avoids getting stuck in local optima. At low temperatures, SA focuses on exploiting/improving and behaves like RHC. The probability of ending the search at any position "s" is given by Boltzmann distribution, $e^{(f(s)/T)}/z_t$, where z_t is a normalization constant. The initial temperature is set to 1E10 throughout the project.

3. Genetic Algorithm(GA): GA is an evolutionary algorithm motivated by biological natural selection and genetic recombination. A population of candidate solutions is evolved iteratively into new generations. Individuals selected using roulette-wheel selection or truncation selection from a generation undergoes crossover(mating) and then sometimes mutation. Cross-overs can be one-point crossover, uniform crossover etc, while mutation mainly involves flipping one bit. The parameters tuned are population size, mate and mutate. Mate and mutate are the number of instances in the new population that are generated by crossover function and mutation respectively. Each mating gives 2 offsprings from the selected parent instances, and offsprings replace the least-fit individuals in the original population. Different crossover and mutation rates traverse different directions in the search space thus affecting the performance of algorithm. Lower number of crossovers and mutations let more individuals/instances continue in next generation unchanged. Higher mutation rates bring in more diversity to the next generation. The selection of parents for crossovers needs searching time and thus high crossover rates can increase the execution time. The best-fitting parameter values are problem-specific, due to the complex interactions between these parameters.

4.MIMIC: Mutual-Information-Maximizing-Input-Clustering (MIMIC) is an optimization algorithm that can directly model the distribution using probability densities, successfully refine the estimates of the distribution and thus help convey the structure of the search space. It communicates information about not just points but also structure, from one search iteration to another. It starts with uniform sampling from the distribution and aims to reach the sample of optimal distributions. It is based on learning and representing the clusters of highly related parameters from the data using dependency trees. The parameter tuned in this project is "m", which decides the structure of the discrete dependency tree built.

Part 1: Weights for the Neural Network
This section focuses on using randomized optimization algorithms of RHC, SA and GA to find the optimal weights for the neural network implemented for the letter recognition dataset in Assignment-1 using backpropagation.

Letter recognition dataset(UCI, n.d)[4] has 20000 instances and 16 attributes. Each attribute takes integer values from 0 to 15. Each instance can be classified into one among the 26 capital letters (A to Z). The dataset was divided into training and testing sets using a 70:30 split and the testing set was used only for the final evaluation. Since the model is built using the training set, using training set itself to tune hyperparameters can cause overfitting and hinder generalization. So, a validation set was used for tuning hyperparameters. 10% of the training set was set aside as validation set. In assignment-1, the best-fitting parameter values for the neural network were found using weka(Weka, n.d)[3], and the ANN had 16 input layer nodes, 26 output layer nodes and 42 nodes in the single hidden layer. The learning rate and momentum were both 0.1. To compare with other algorithms, backpropagation in ABAGAIL was used to train the same neural network in assignment-1and it obtained similar accuracy as in assignment-1, which is around 87.8% training accuracy and 85.8% testing accuracy.
The random nature of the algorithms gives slightly different results for different runs. So, to cancel out this variance, the charts are plotted based on the average of 5 runs of each algorithm. Using large initial values for neural network weights can cause **overfitting** and hence values around 0 are preferred. Also, all-zero initialization leads to same error for all neurons and the model will not learn anything. Thus, to incorporate a source of asymmetry, small non-zero random initial values are used. **Accuracy** is computed as the number of correctly classified instances / total number of instances to be classified. There are 26 output nodes, each with value between 0 and 1. An instance with label "A" is correctly classified when node "A" has maximum value among its output nodes.
The neural network weights are continuous and real-valued, instead of discrete. Many optimization methods do a local gradient-based search and converge to local optima. SA usually performs a global search and uses Markov Chain Monte Carlo methods. For this project, ABAGAIL implementation has in-built classes(GitHub, 2018)[2] like NeuralNetworkOptimisationFunction.java and NeighborFunction.java to take care of this. NeighborFunction.java takes an instance of weights and returns another neighboring set of weights using a distance measure for RHC. Temperature decides the selection of next instance in SA, such that at high T, selection is random and at low T, fitness improvement is focused upon. In GA, StandaradGeneticAlgorithm.java keeps a population of instances which gets mated and mutated. Mating creates offspring from the parent's weights using CrossoverFunction.java. Mutation modifies the weights of an instance.

Randomized Hill Climbing: RHC is a simple algorithm with no much hyperparameter tuning options. The training set accuracy and validation-set accuracy constantly improved with iterations. In Fig.1[LEFT], the algorithm took 42K iterations to converge to around 60% validation accuracy. In Fig.1[RIGHT]., the cumulative training time rises linearly with the iterations as RHC takes more steps in the direction of improvement of accuracy. When stuck in a local optimum, RHC no longer improves as it cannot find an immediate neighbor with more fitness value.

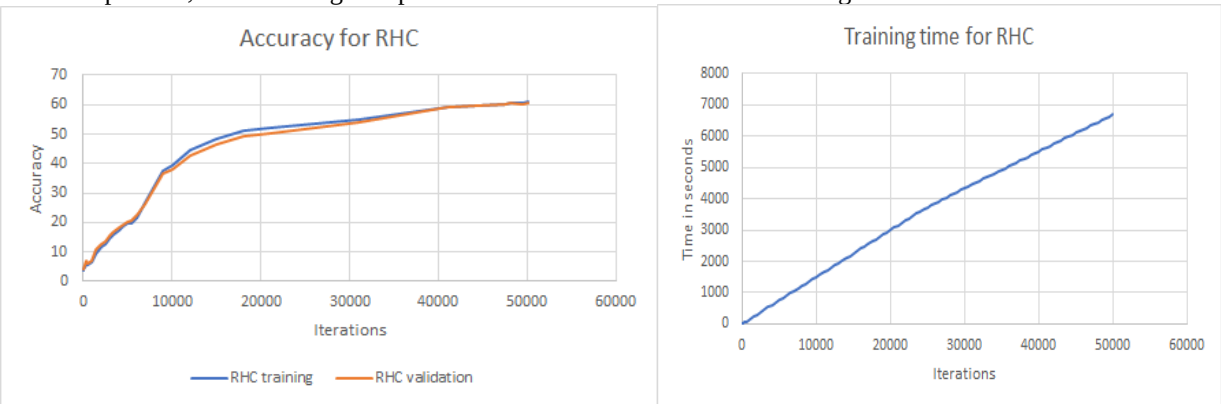

Fig.1. Accuracy for RHC[LEFT]. Training time for RHC [RIGHT]

Simulated Annealing: The initial temperature for SA was set as 1E10 and the cooling exponent was varied to find the best-fitting value. Fig.2[LEFT] shows variation in **validation-set accuracy** with parameter CE. CE=0.45 gives the best accuracy. Though CE=0.6 gives the fastest training and CE=0.45 takes longer in Fig.2[RIGHT], since accuracy is more important for the neural network model, I chose CE=0.45 as per the model complexity plot in Fig.3 Higher CE values can be over-exploratory as the temperature decay is slow-paced. Lower CE values might have abruptly decreased the temperature thus making SA act like RHC. Also, the training and validation sets show similar accuracies in Fig.3[LEFT].

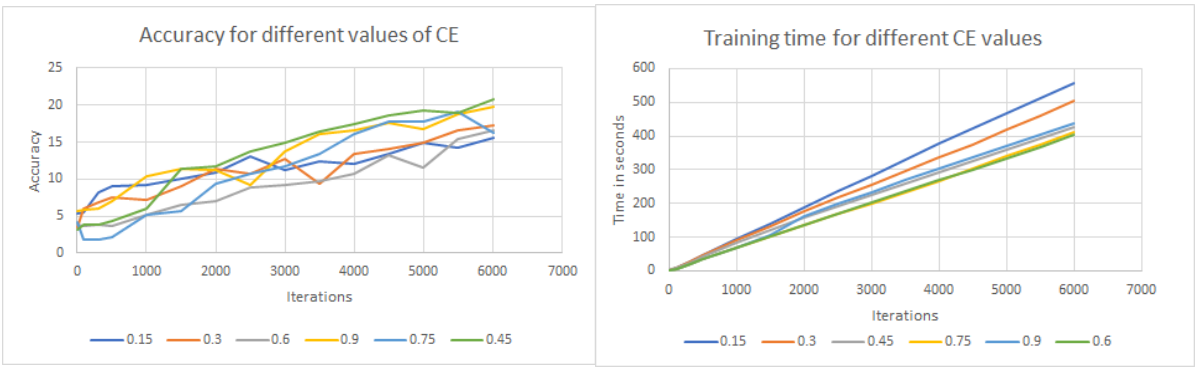

Fig.2.Accuracy for different values of CE[LEFT]. Training time for different CE values[RIGHT]

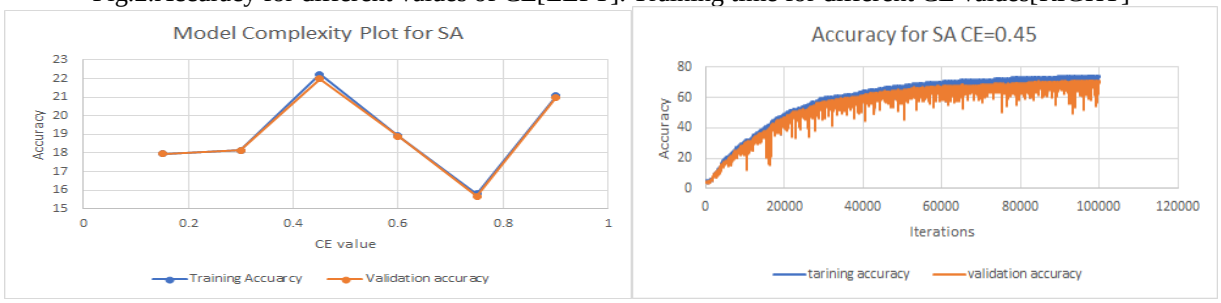

Fig.3.Model Complexity Plot for SA[LEFT]. Accuracy for SA CE=0.45 [RIGHT]

Genetic Algorithm: Several combinations of the hyperparameter values for population, crossover and mating were experimented and the best 4 combinations are shown in Fig.4[LEFT] (400,10,10) is chosen as the best-fitting values as it shows best **validation-set accuracy** in Fig.4[LEFT] and the least training time in Fig.4[RIGHT]. The hyperparameter tuning chart in Fig.5 shows that for both training set and validation-set, (400,10,10) gives best accuracy.

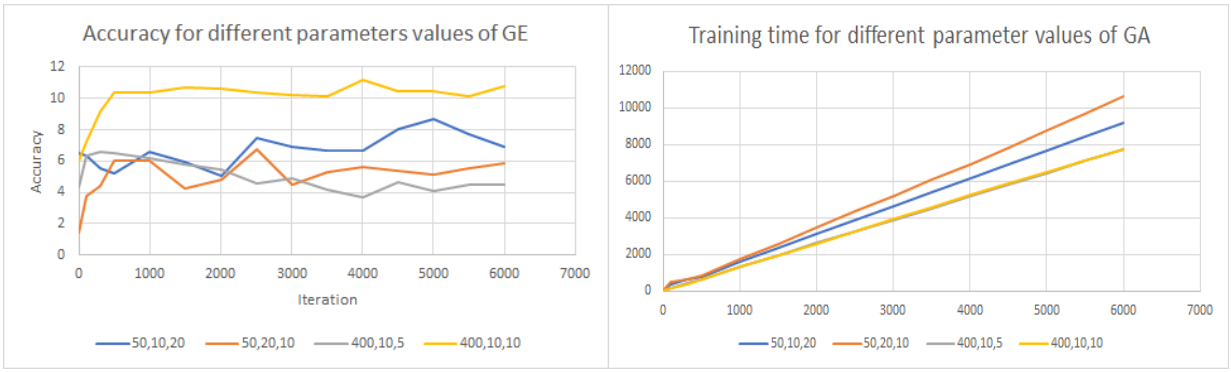

Fig.4.Accuracy for GA[LEFT]. Training time for GA[RIGHT]

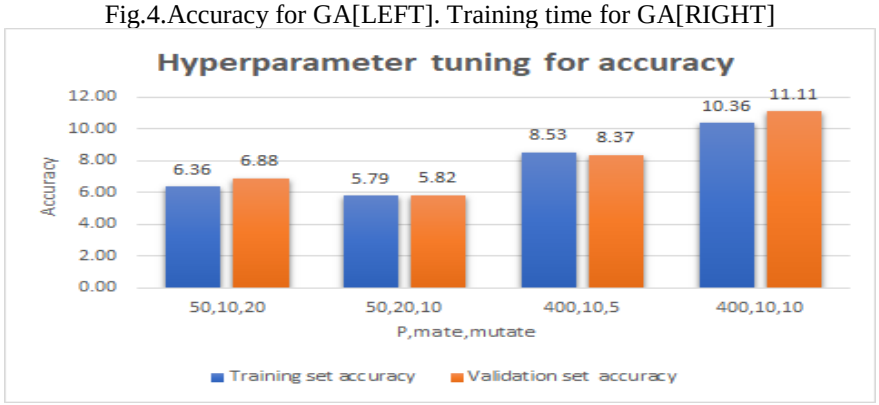

Fig.5. Hyperparameter tuning for GA

Analysis:

Algorithm	Iterations needed to converge	Training accuracy at convergence	Validation accuracy at convergence	Testing accuracy after convergence
Backpropagation	2000	87.26%	85.45%	84.67%
RHC	42000	60.86%	60.64%	60.73%
SA	98000	74.3%	70.75%	71.47%
GA	4500	10.35%	10.75%	11.15%

Table 1. Comparison of convergence of algorithms.

Fig.6.Comparison of training accuracy[LEFT]. Comparison of validation accuracy[RIGHT]

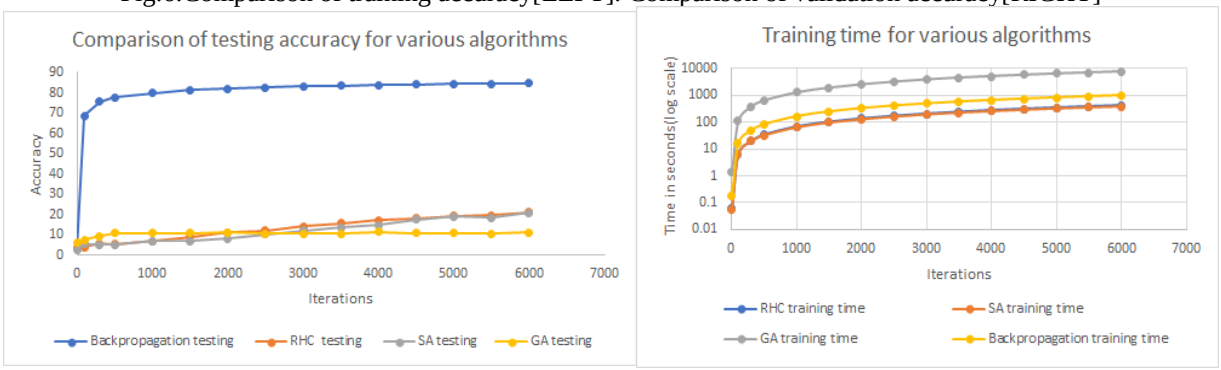

Fig.7.Comparison of testing accuracy[LEFT]. Comparison of training time [RIGHT]

Backpropagation shows the best accuracies in Table.1 and converges within 2000 iterations in Fig. 7[LEFT].GA shows a premature convergence at 10% validation accuracy and 4500 iterations. As in Fig.1[LEFT], RHC converges around 42000 iterations to 60% validation accuracy. Since the number of iterations needed is very large, convergence of RHC and SA are shown separately in Fig.1.[LEFT] and Fig.3.[RIGHT] respectively. SA took 98K iterations to converge to 70% validation accuracy. Including 100K iterations in Fig.6 can take away the focus from the trends of convergence of GA and backpropagation. In finding neural network weights, local optima are not much of a problem and the objective is to find a good local optimum, rather than a global optimum. Aggressive optimization to reach global optima can cause overfitting. For example, SA can step out of local optima(explore) when temperature is high, and this exploratory approach in pursuit of global optima can end up in overfitting if it tries to over-focus on noisy samples and thus obtain worse generalization. However, converging to bad local optima like GA(10% accuracy) in this case, is also bad. GA takes the most training time per iteration, followed by backpropagation in Fig.7.[RIGHT] RHC and SA have faster iterations. The run time for GA is higher than RHC and SA, because it performs several crossovers and mutations in each step, as opposed to hill climbing which chooses one neighbor per step. However, this helps GA to explore many new paths in the search space at once and the least-fit of those are replaced periodically by better offsprings. As the model complexity increases, GA training becomes slower and thus less opted. Backpropagation computes the gradient descent at each step and hence attributing to higher training time than hill climbing. Lesser iterations needed for the convergence of backpropagation can be attributed to the fact that it computes which way the slope is steepest (gradient descent of error function) as opposed to RHC who randomly tries all directions.

Part 2: Optimization problems

This section focuses on three optimization problems, each showcasing the strength of a randomized optimization algorithm. For each problem, each algorithm was repeated 5 times for each parameter value and the average was taken to plot the charts. The averaging helps in cancelling out the variance caused due to random nature of the algorithms.

1.Travelling Salesman Problem(TSP)

Problem description: This is a famous NP-hard problem in combinatorial optimization and the goal is to perform the shortest round-trip by visiting all cities exactly once. I set the number of cities(N) to 110. The shortest distance or the global minima of the distances is to be found here. To turn this minimization problem into a maximization problem, I am using the inverse of the distance as the fitness value, and hence the fitness values are mostly decimal-point values less than 1. This problem is i**nteresting** because it highlights the strengths of GA. GA being a multi-path evolutionary algorithm, suits TSP well. It helps to find a good balance between the diversification and intensification phases. Single path algorithms like SA is not the best option here. RHC being a local-optimization based iterative method, can easily get stuck in local optima. Since SA can step out of a local optimum by accepting a worse immediate neighbor as a stepping stone to global maxima, it has far better chances of finding global optima than RHC. The practical applications of TSP include travel planning where the number of cities can be very large. MIMIC being slower, can cause time complexity issues. Since the physical distance between various cities is to be minimized, there is a conditional dependence between different cities (each city should prefer to be connected to its neighbors) for best fitness values. So, MIMIC might do well as it can learn and represent these structural dependencies. But eliminating the lower fitness solutions from the sample periodically might put out solutions that are stepping stones to global maxima, and hence MIMIC can also get stuck at local optima.

Hyperparameter Tuning: Being a simple algorithm, RHC implementation has no tunable parameters. Using default parameters, RHC gives a fitness score of 0.0561 in Fig.8.[LEFT]. In Fig.8[RIGHT], SA gives best fitness value of 0.0566 at m=0.15(top blue line). Lower CE gave better fitness values and the intuitive reasoning behind it can be that the faster decay of temperature helps SA to exploit earlier and more than otherwise.

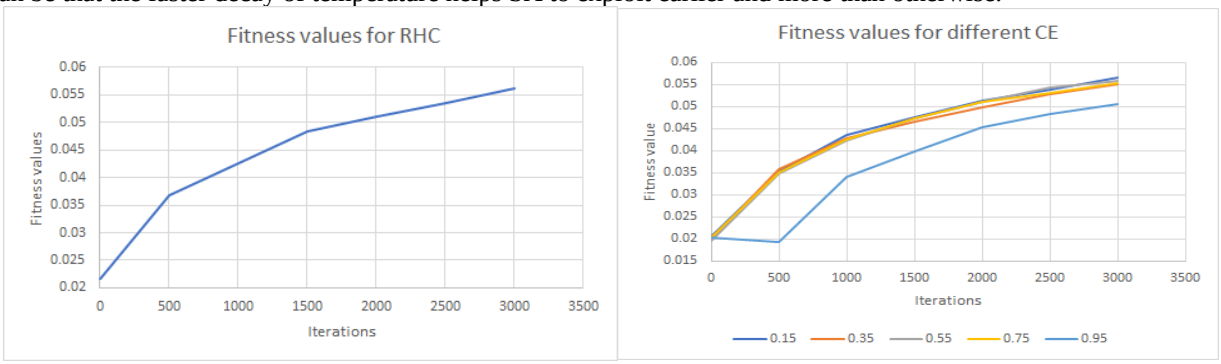

Fig.8.Fitness values for RHC[LEFT]. Fitness values for SA[RIGHT]

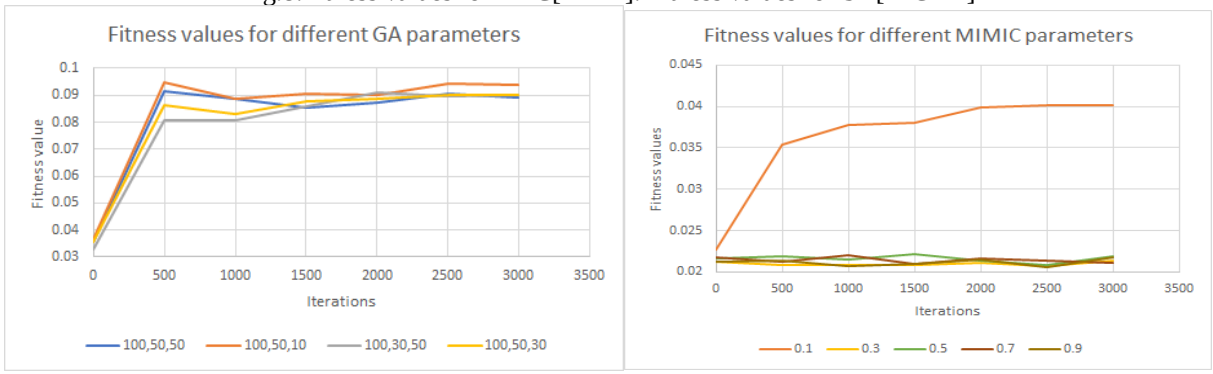

Fig.9.Fitness values for GA[LEFT]. Fitness values for MIMIC[RIGHT]

Several combinations of (P, mate, mutate) values were checked and the best four are reported here. GA gives best fitness value of 0.094 for (100,50,10) in Fig.9[LEFT]. When mutation count is increased to 10 from 50, performance goes down because too much diversification need not help increase fitness values. This is because offsprings with

high fitness, if mutated, might turn into individuals with low-fitness. Generally, when the crossover count was decreased, the performance was observed to go down. This might be because when mating is less, more least-fitting individuals remain through several generations as they are not getting replaced by offsprings with higher fitness values. MIMIC shows best fitness value of 0.04, in Fig.9[RIGHT], for m =0.1.
Since GA gave the best fitness value, I plotted the model-complexity curve and training time to determine the best parameter values for GA, in Fig.10[LEFT], which emphasizes the difference in fitness values for different parameter combinations at 4K iteration. (100,30,50) has least training time as crossover operations required is less. Since fitness value is more important than training time and (100,50,10) is second-best in terms of training time in Fig.10.[RIGHT], as it needs least number of mutations, it is chosen as best-fitting parameter values.

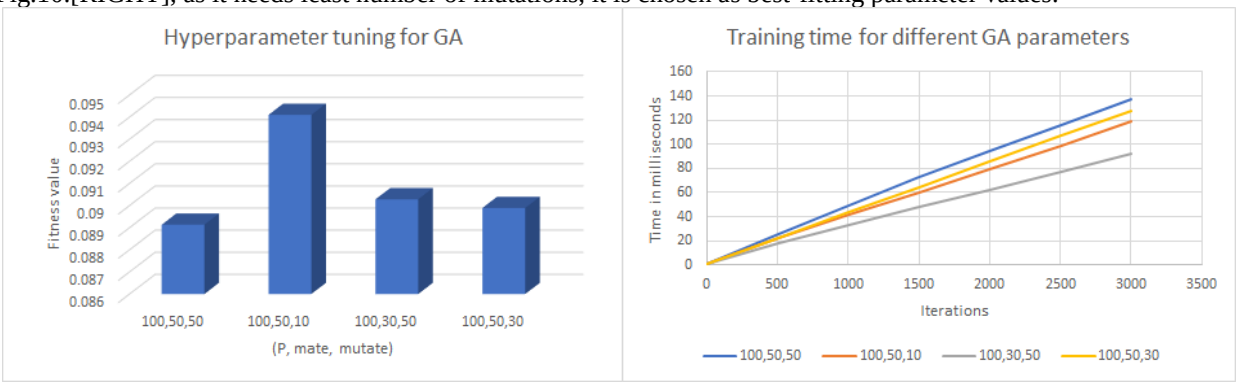

Fig.10. Hyperparameter Tuning for GA[LEFT].Training time for GA[RIGHT]

Analysis:

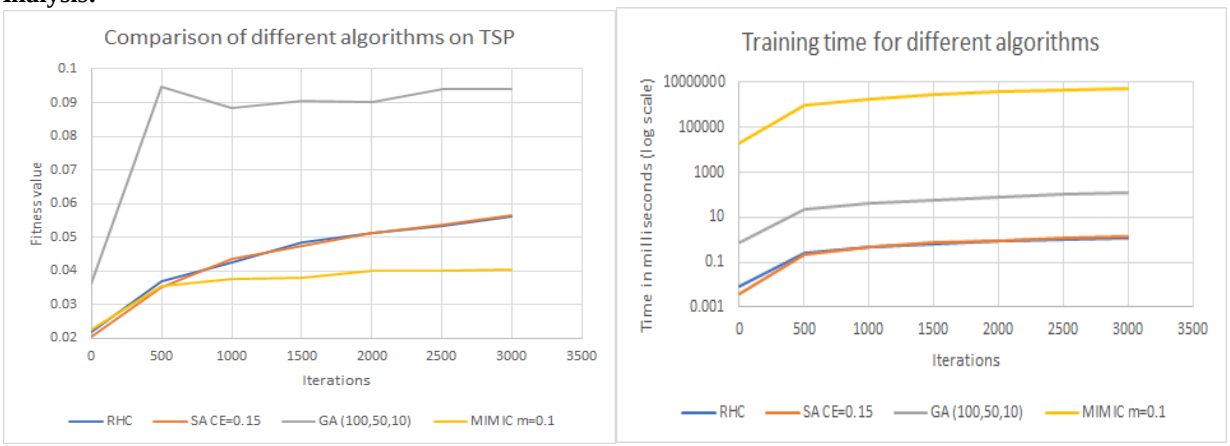

Fig.11. Comparison of different algorithms on TSP[LEFT]. Training time for different algorithms [RIGHT]

As expected, GA performed better than all other algorithms, converged to 0.094 at around 2500 iterations in Fig.11[LEFT]. MIMIC converged at a local optima of fitness value of 0.04 at around 2100 iterations. SA and RHC has not converged at 3K iterations and achieved a fitness value of 0.0566 and 0.0561 respectively. RHC and SA were further run to check convergence, and it was found that RHC converged by 24010 iterations to 0.0711 and SA converged to 0.072 by 23750 iterations. (plots showing their convergence is not included due to space constraints). Using a population-based version of SA, where a set of solutions evolve like an evolutionary algorithm might give better results. Also, each solution must use different temperature and CE, and parallel processing can be used for efficiency. In Fig.11[RIGHT], as expected, MIMIC took the longest training time, then GA and then similar time for RHC and SA. Building dependency tree and formulating the structure of the search space makes MIMIC slow. Performing selection, crossovers and mutations cause the delay in GA training. Probabilistic estimation of which paths to follow depending on temperature and fitness values, takes time in SA, but very less compared to MIMIC and GA. RHC has less time cost, particularly because of its algorithmic simplicity.

2.Continuous Peaks Problem (CPP) :

Problem Description: CPP focuses on finding the highest peak (global optima) in a search space that has many continuous small peaks (local optima). ABAGAIL implementation with discrete valued parameter space (Bit strings) was used for this project. N is the number of bits in the string. When a string having greater than T

contiguous bits set to 0 and greater than T contiguous bits set to 1 is found, a bonus reward of N is provided. I set N to 60 and T to 6 for this project. The goal is to maximize the length of the consecutive run of either 0 or 1 and to have both 0 and 1 to be contiguous for over T bits. So, the global maxima here is 113, when the bit string has either 7 consecutive zeros and the rest contiguous ones, or vice-versa. This problem is **interesting** because it highlights the strengths of Simulated Annealing. Since this problem can have several local optima, the exploratory approach in SA has better chances to avoid those and hit the global optima in the large search space, while RHC can get stuck in a local optimum, unable to climb down the local peak. The algorithms try to find the longest run of zeros and longest run of ones anywhere in the string. MIMIC can also do well as it estimates probability distributions, conveys the structure of the search space and thus helps to show the good and bad regions. Also, once it finds the high-fitness global peak, it is preserved in the population for the remaining iterations and helps eliminate instances with less fitness values, by increasing the threshold. GA implementation uses Single point crossover and one-bit mutation. With enough iterations, single point crossover has a chance of meeting the global optima.

Hyperparameter Tuning: Being a simple algorithm, RHC implementation has no tunable parameters and it gives a fitness score of 93 in Fig.12[LEFT]. For SA, CE =0.75 performs the best, giving a fitness value of 113in Fig.12[RIGHT].

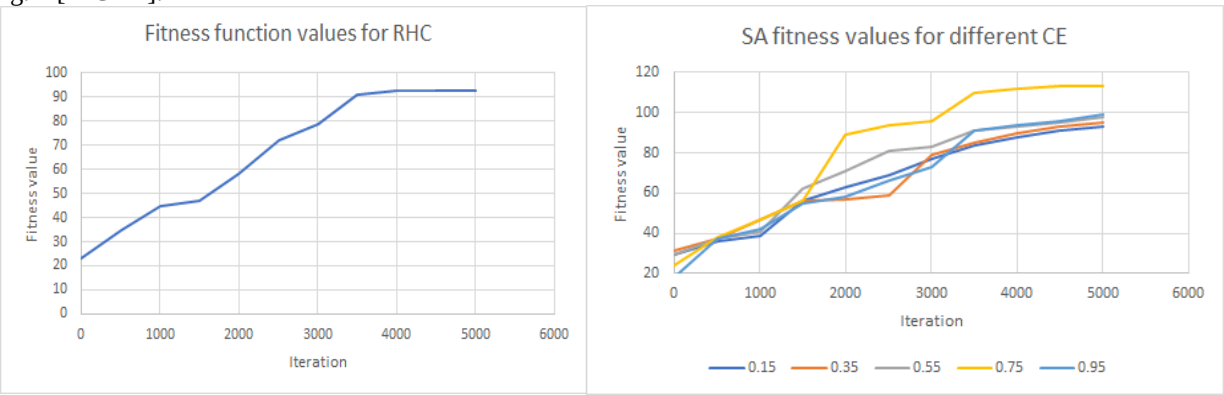

Fig.12.Fitness function values for RHC[LEFT] and SA[RIGHT]

For GE, (P, mate, mutate) values of (100,50,30) gives best fitness value, around 87 in Fig.13[LEFT] For MIMIC, m=0.7 performs best in Fig.13.[RIGHT] and gives a fitness score of 99.

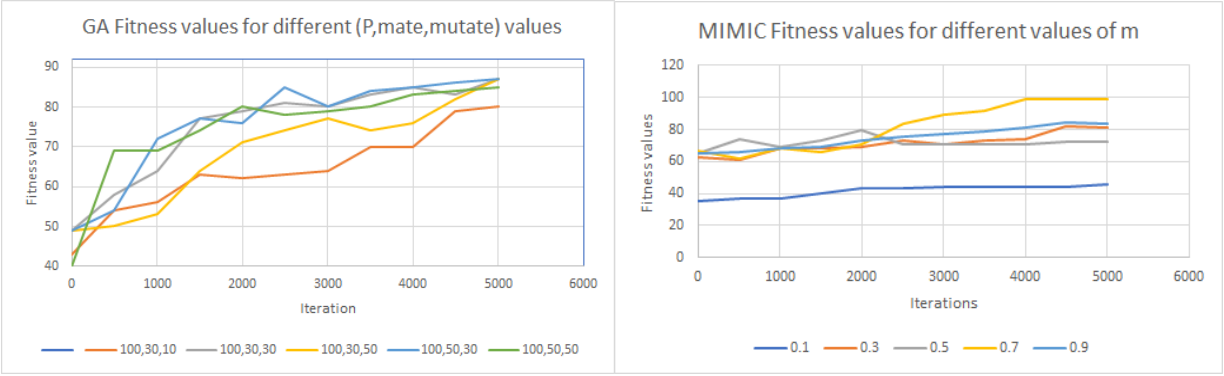

Fig.13.Fitness function values for GA[LEFT] and MIMIC[RIGHT]

Since SA showed best performance, the model complexity plot and training times are further plotted in Fig.14 and 0.75 gives the best performance. Values higher or lower than 0.75 gives too slow or too fast temperature decay, thus not giving the correct exploitation-exploration balance for SA.

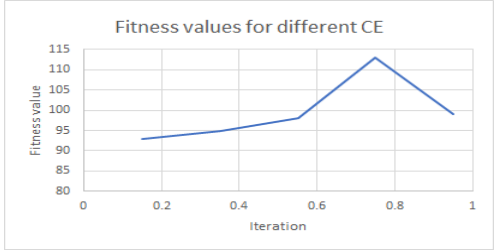

Fig.14.Model complexity plot for SA

Analysis:

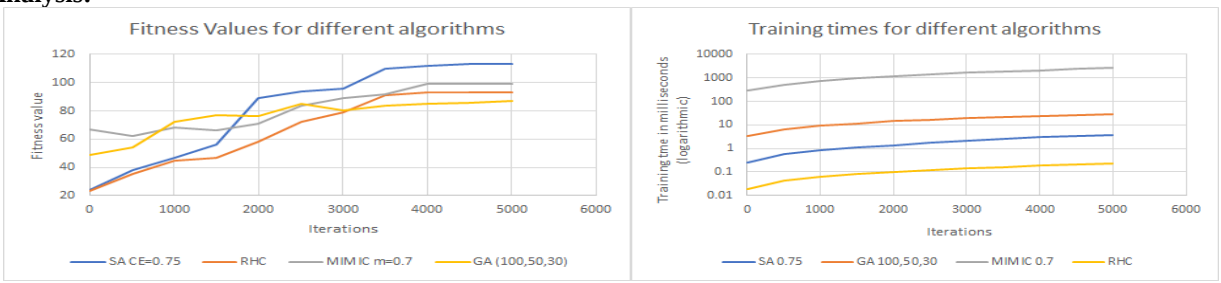

Fig.15.Fitness values for different algorithms [LEFT]. Training times for different algorithms [RIGHT]

From Fig.15[LEFT], as expected, SA gave best performance and converged around 3700 iterations. RHC converged at a local optimum with fitness value 93, around 4000 iterations. MIMIC also converged to a local optimum with fitness score 99, at around 4K iterations. GA converged to another local optimum with value 87 at around 4500 iterations. RHC, as expected, got stuck in local optima. Regarding training time used, MIMIC takes the longest in Fig.15[RIGHT], followed by GA, then SA and RHC, due to reasons explained in the case of TSP.

3. Flip-flop(FF) Problem

Problem description: The flip-flop problem is defined over bit strings and counts the alternating bit pairs from right to left until it encounters a non-alternating bit pair. This count is the fitness function value which is returned by the function. In ABAGAIL, I set the total length of bit string to be 850 and hence the maximum possible fitness value is 849 (global maxima). This problem is **interesting** because it highlights the strengths of MIMIC. The fitness value of each bit is dependent on its neighbors. A bit value of 0 surrounded by 1 on both sides or viceversa, contributes 2 to the overall fitness value. MIMIC captures this dependency between neighbors well. Since the bits are not independent, an unconditional probability distribution will not suffice and a discrete dependency tree or a chain is needed. GA with single-bit mutation and single point crossover might need an exhaustive search to reach the global optima. SA and RHC do not model the structure of the problem and hence might not identify that the bits are dependent on neighbors. This problem thus uses the strength of MIMIC representing structure and illustrates the weakness of RHC and SA not being able to do that.

Hyperparameter Tuning: RHC uses the default implementation in ABAGAIL with no tunable parameters and gave fitness value of 792 in Fig.16[LEFT]. For SE, CE=0.55 gave the best fitness value of 832 in Fig.16[RIGHT]. Higher or lower values of CE did not give the required fineness of temperature decay.

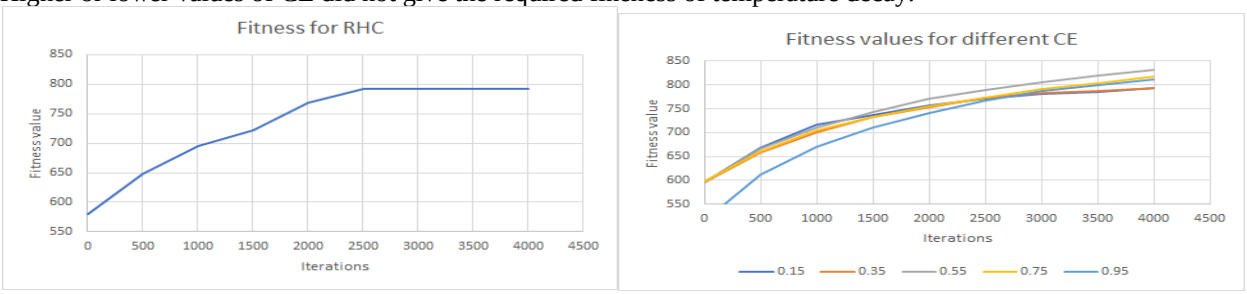

Fig.16.Fitness values for RHC[LEFT] and for SA[LEFT]

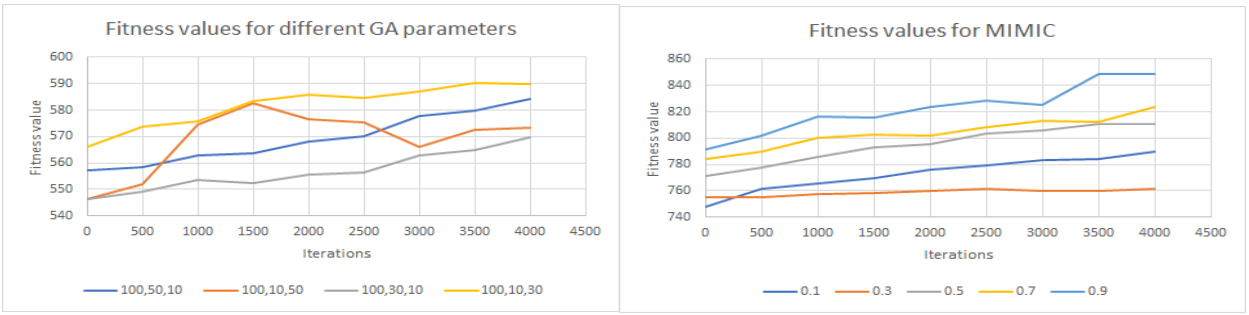

Fig.17.Fitness values for GA[LEFT] and MIMIC[RIGHT]

GA showed best performance at (P, mate, mutate) values of (100,10,30), with a fitness value of 587 in Fig.17[LEFT]. MIMIC gave best possible fitness value of 849 at m=0.9 in Fig.17[RIGHT].
Since MIMIC performed the best, the model complexity curve for MIMIC is plotted in Fig. 18[LEFT] to identify the best value for m. 0.9 gives best fitness value and least training time in Fig.18[RIGHT] and hence making it the best value for parameter m.

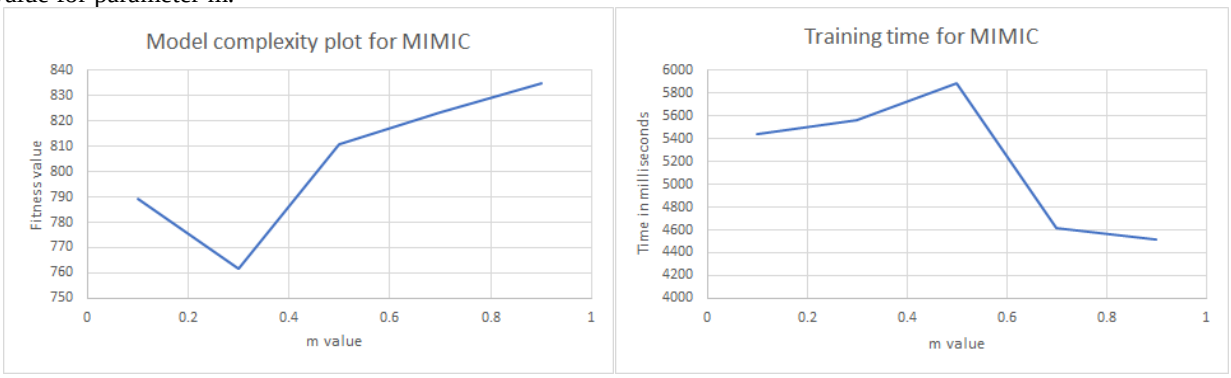

Fig. 18.Model Complexity Plot for MIMIC[LEFT]. Training time for MIMIC[RIGHT]

Analysis:

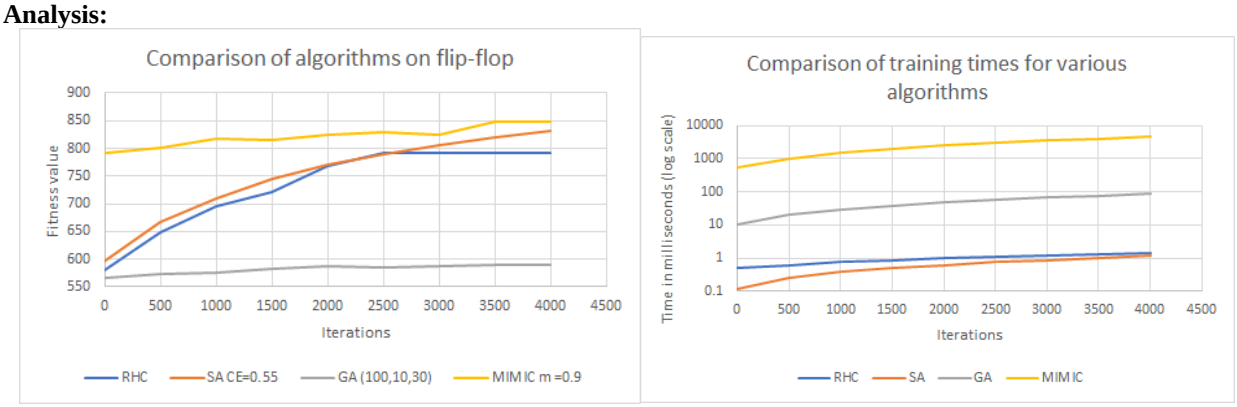

Fig.19.Comparison of fitness values [LEFT]. Comparison of training times [RIGHT]

The flip flop problem implemented using ABAGAIL uses DiscreteChangeOneNeighbor.java for RHC and SA where only one bit is flipped in each iteration. In Fig.20[LEFT], any one-bit change from 4th step decreases immediate fitness score and since RHC does only one step lookahead, it gets stuck in local optima with value 6. At high temperatures, SA can explore and choose fitness value of 5 over current 6 and thus it can reach the global maxima of value 7. This is a small-scale illustration of RHC getting stuck in the N=850 flip flop problem used in this project.

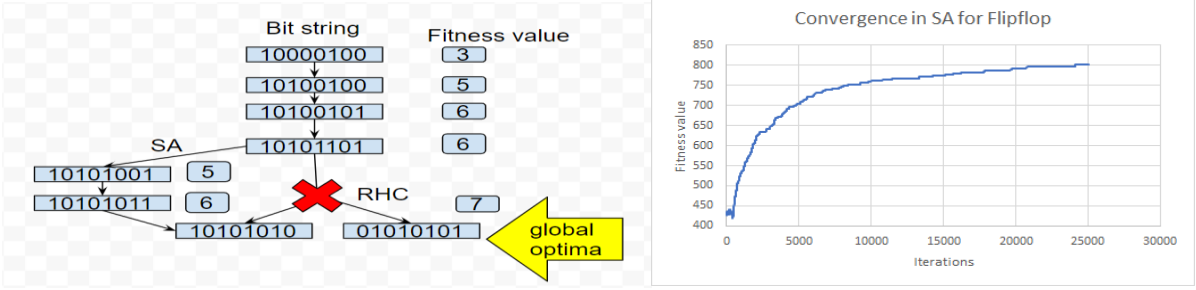

Fig.20.Illustration of local optima[LEFT]. Convergence of SA[RIGHT]

MIMIC learns a dependency tree that represents the structure of alternating bit pairs and reaches the global optima consistently. At 4K iterations, SA reached fitness value of 832, but has not converged yet. With enough iterations, it might be able to find global optima as well. However, as iterations increase, the temperature cools down and SA has lesser chances of finding the global peak. To find out when will SA converge, I ran SA alone again till 25K iterations and the result is plotted in Fig. 20[RIGHT]This time, SA converged to a local minimum of 801 at 24060

iterations (slightly different value from earlier run). MIMIC converged to global optima by 3500 iterations in Fig.19[LEFT]. RHC converged to a local optima of fitness value 792 by 2500 iterations. Being exploiting in nature, RHC cannot step out of it. GA had a premature convergence to a local maximum with fitness value of 590, at 3400 iterations. In Fig.19[RIGHT], MIMIC takes the longest training time, followed by GA, then RHC and then SA due to same reasons explained in the case of TSP.

Conclusion

In problems where local optima are to be avoided, RHC is not preferred as it get stuck in local optima. SA with CE and temperature fine-tuned to give right balance between exploitation and exploration can be used in such cases. MIMIC suits problems where structure of the search space is to be modeled. MIMIC does well on discrete bit-string problems like flipflop where bits have interdependence, as it can model these dependencies using dependency trees. Evolutionary algorithms like GA gives multi-path solutions, can optimize multiple objectives simultaneously and accelerate the process of convergence, with right set of parameters. GA works well with black box problems where there is no particularly informed model of the problem. Simple problems with only one optimum are well-suited by the simple algorithm of RHC. For a fixed number of iterations, MIMIC takes the most execution time, followed by GA from the plots in this report. Time complexity per iteration is relatively low and similar for RHC and SA. However, the number of iterations needed to converge to an optimum is very high for RHC and SA and less for GA and even lesser for MIMIC, as explained earlier in this report. The fitness values for different algorithms are problem-specific and depends on how well the parameters are tuned, how well the problem utilizes the strengths of an algorithm and how well it overcomes the weaknesses of an algorithm. The results and analysis of the earlier optimization problems has described this in detail. Since MIMIC stores the structure and GA stores a population at every iteration, the space complexity is higher compared to RHC and SA which just do one-step lookahead at each step. As the N value in the optimization problem increases, each algorithm needs more execution time.

Another criterion to compare algorithms is the number of function evaluations taken to converge. MIMIC takes more function evaluations per iteration as it computes the fitness value of many samples per step. Similarly, GA needs good number of function evaluations as it checks the values of all individuals in the population to select the parents for recombination. RHC and SA have very less function evaluations per iteration. However, since MIMIC and GA converges within lesser iterations, the total number of function evaluations needed till convergence is generally least for MIMIC, then GA, then similar for RHC and SA. For TSP problem, on an average, 100 function evaluations were needed per step for MIMIC, 50 for GA, 1 for RHC and SA each. But SA and RHC took around 24K iterations to converge while MIMIC and GA converged well within 2.5K iterations. So, in terms of number of iterations needed to converge MMIC is the best (least number of iterations) and in terms of computational complexity and functional evaluations per step, RHC or SA is preferred. Using grid search and model complexity plots to fine-tune various parameters of the algorithm can give improved performance. Hybridizing SA with other approaches gives better results. SA can be used with Reactive Tabu Search (RTS) for better results. When search revisits previously traversed solution, the tabu list length increases. In case of too many repetitions, RTS uses an escape mechanism with random walk. Substituting the random walk with controlled simulated annealing gives better results. Using random restarts for RHC can cancel out the effect of random initial points and when many restarts happen, it is more probable that RHC begin from the basin of attraction and thus reach the global maxima. In this project, I used multiple RHC runs, each with a random initial position, thus having a similar effect as restarts. If any of the run reached global peak, it was considered for plotting and otherwise, the average of different runs was used for plotting.

REFERENCES:

1. Pushkar. ABAGAIL Github. (2018). Retreived from https://github.com/pushkar/ABAGAIL
2. Tay, J., Github(2017).Retrieved from https://github.com/JonathanTay/CS-7641-assignment-2
3. Weka: Data Mining Software in Java. (n.d.). Retrieved September 10, 2018, from https://www.cs.waikato.ac.nz/ml/weka/
4. Letter Recognition Dataset. UCI Machine Learning repository. Retrieved September 10, 2018, from https://archive.ics.uci.edu/ml/datasets/letter+recognition