Assignment 2 : CS 7641

Muhammad Haris Masood
mmasood30@gatech.edu

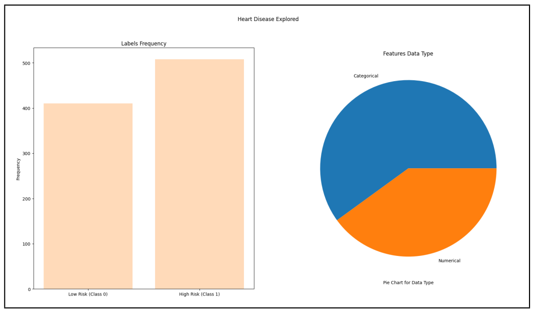

Figure 1. Heart Dataset

1. Introduction and Data

1.1. Overview

My chosen classification problem was identifying if an individual was high or low risk in terms of having a heart attack [2] (heart dataset). The dataset was found on Kaggle. The heart dataset is medical data, the features are several health markers for a patient, and the label identifies if the patient is at a high risk for a heart attack.

The exact characteristics of the dataset can be observed from Figure 1. We selected the heart disease dataset for our experimentation because it was relatively well balanced, included a good mix of both categorical and numerical features, and had shown promising performance with neural network models in our previous assignment. In addition to training a neural network, we explored random optimization. This exploration was done by utilizing four random optimization algorithms to solve three optimization problems.

2. Hypothesis

Based on extensive preliminary study we formed several hypotheses:

1) We believe that MIMIC will outperform other algorithms on problems that posses complex problem space. This is because MIMIC is capable of developing a deep understanding of the structure of the underlying problem space.

2) We believe that Simulated Annealing will converge on the global optimum faster than MIMIC and the Genetic Algorithm for simpler problems, but, may require careful tuning of temperature parameter.

3) We believe that the Genetic Algorithm will outperform other algorithms in solving combinatorial problems. The algorithm's utilization of a large diverse population, capable of encoding numerous combinations (via chromosomes), will excel in exploring large and complex combination spaces.

4) The Gradient Descent algorithm in Mlrose does not have any stochasticity, and we believe this will likely result in faster convergence than the other random optimization algorithms. However, it may converge on a local minimum more often [5]. We also believe that utilizing gradient descent without any stochasticity to train the model may result in overfitting.

5) Given the continuous nature of the problem space in neural networks, we believe that Randomized Hill Climbing will perform poorly because it is optimized for discrete problem spaces. Conversely, other randomized optimization strategies have shown to work effectively in continuous spaces, so they should yield better performance. [6] [4].

3. Experimentation Methodology

3.1. Preprocessing Data

For the heart dataset, we removed the Booking ID column, as that was just an index value. All categorical features were one-hot encoded to avoid any incorrect ordering in the data.

3.2. Metrics

Due to the balanced nature of the heart disease datasets we have selected accuracy as our metric for most of our experimentation and hyperparameter tuning plots. We believe accuracy provides us a reliable metric to compare the effects of different optimization strategies. In addition to accuracy we also utilized the number of function evaluations, the number of iterations and the wall clock time required till the the optimization algorithm converged. For the optimization problems we utilized score, number of iterations, number of function evaluations, wall clock time (till convergence).

3.3. Procedure

Our experimentation and paper can be broken down into two phases. Phase one involved using four optimization strategies: Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and the MIMIC algorithm (MA) on three different optimization problems. Each problem was attempted with three different problem sizes, ranging from ”small” to ”large”. Hyperparameters tuned for each problem size. Although the labels were shared between problems, the actual sizes were not. Appropriate sizes were determined through initial experimentation. Sizes that highlighted differences between algorithms were carried forward into the final more in-depth experimentation.

For RHC, hyperparameter tuning involved identifying the optimal number of restarts. For SA, we determined the ideal starting temperature and the optimal temperature decay function. Hyperparameters tuned for GA included population size, population breeding percentage, and mutation probability. Finally, for MA, population size and the percentage of the population to keep for the next iteration were varied. The exact methodology of selection prioritized the **score/fitness**, and if all hyperparameter values led to convergence, we selected the hyperparameter value that resulted in the **fewest number of function evaluations**, this indicated faster or more efficient convergence. The number of function evaluations served as a corollary to convergence because through our experimentation, we determined that higher function evaluations usually correlated with longer wall clock time when comparing within the algorithm.

The tuned optimization algorithm was tested on the problem, results plotted and compared. Each step of the experiment (tuning and evaluation) was repeated at least five times, and for each repetition a unique random seed value was assigned. The assigned seed value was utilized in conjunction with that repetition for all experiments and testing. This was done so results would be recreatable. The objective throughout our experimentation was to ensure all algorithms were able to converge. To achieve this we carefully tested the convergence criteria, max number of iterations, and, max number of attempts. It was found that a high value for the max number of iterations (1e6) coupled with a high value for max attempts (1e3) led to convergence, or, near convergence for most problem sizes and algorithms.

It’s worth noting that although it was feasible to explicitly determine the maximum value achievable for two out of the three optimization problems, we refrained from specifying it for several reasons. Firstly, we believed that providing the maximum value during hyperparameter tuning might introduce bias into our hyperparameter selection process. Secondly, we aimed to capture nuances in the optimization algorithms’ behaviors that would otherwise have been missed if a maximum value was set. For instance, consider simulated annealing, where the algorithm might reach a global optimum but, due to a high temperature value, may transition to another solution. Lastly, real-world problems often lack a known global optimum beforehand.

Phase two involved training a neural network model utilizing Gradient Descent (GD), RHC, SA and GA algorithms, and, comparing their respective performances. Our initial objective was to recreate the optimal neural network we

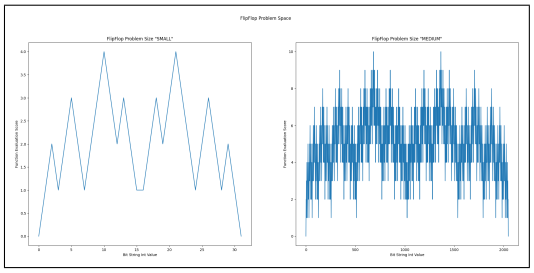

Figure 2. Problem Space for the FlipFlop Optimization Problem

identified in Assignment 1. However, we were unable to do so because several of the parameters we tuned in Assignment 1 were not available in Mlrose. Notable parameters missing in Mlrose included the Adam optimizer, dropout, and the elu activation function. For these reasons we developed a new network, which took inspiration from our optimal Assignment 1 model, but, was refined with hyperparameter tuning and K-fold cross validation using the GD optimization algorithm. Upon evaluation, we found that the performance of the final optimal model in Assignment 2 was comparable to that of Assignment 1. This structure was then used for the entirety of the neural network portion of the experimentation.

Our next step in the neural network experimentation involved taking our base model and utilizing the aforementioned random optimization strategies to train the model. The first step involved hyperparameter tuning, specifically, this involved identifying the ideal number of restarts for RHC, the ideal initial temperature and temperature decay function for SA, and, the ideal population size and mutation percent for GA. In addition to these parameters, learning rate was also tuned for each strategy. After tuning, final model performance across five runs was recorded and compared. Similar to the optimization problems, unique random seeds were assigned to each run.

4. Optimization Algorithms

4.1. Simulated Annealing

4.1.1 Problem Introduction - FlipFlop

The first optimization problem we experimented on was the FlipFlop problem. Given that a bit string can be of length x the score is calculated as the total number of pairs of consecutive elements.

Problem Space for the FlipFlop Optimization Problem is shown in Figure 2. We note the abundance of local optima present throughout the problem space. We also note that there exists two global optima, one for when the string is entirely comprised of 1’s and the other for when it is entirely comprised of 0’s. We note a main challenge with this optimization problem appears to be converging to a local optima.

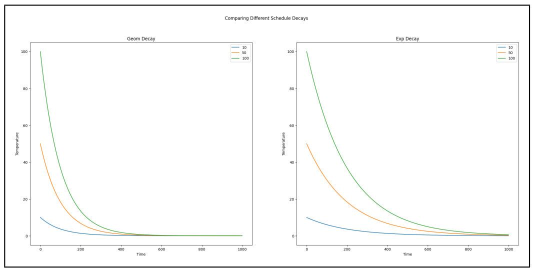

Figure 3. The effect of T_0 and decay function on T

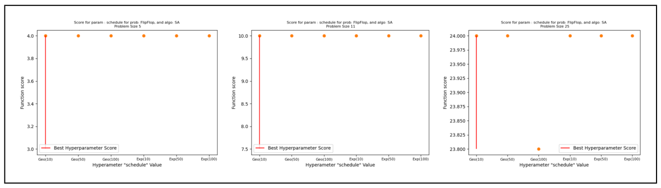

Figure 4. SA Temperature Tuning for the FlipFlop Problem

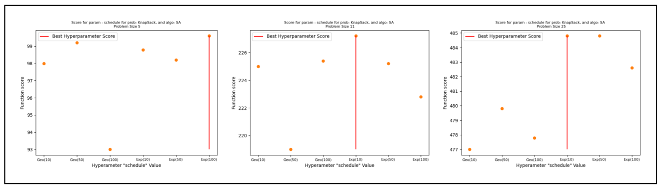

Figure 5. SA Temperature Tuning for the KnapSack Problem

4.1.2 Hyperparameter Tuning for Simulated Annealing

Initial temperature and decay function were tuned for SA. Figure 3 provides a clear visualization of temperature as a function of Geometric decay or Exponential decay, with different initial $\mathcal{T}$ values. Exponential decay results in a slower temperature decrease than Geometric decay, leading to a higher probability of accepting worse steps. This emphasizes the need to balance exploration and exploitation in optimization problems. Higher temperature encourages exploration, while lower temperature allows focus on promising areas for exploitation.

During our hyperparameter experimentation we observed some interesting results. The outcomes for the FlipFlop problem are illustrated in Figure 4, and we also included results from a different optimization problem, the Knapsack problem, depicted in Figure 5 (the Knapsack problem is discussed in more detail in the GA section and the weights used in that section are not the same ones shown here). We noted that for the FlipFlop problem nearly all T_0 and decay functions were able to converge on the global optimum. However, lower temperature value coupled with a faster decay led to faster/more efficient convergence. This finding aligns with the logic we have discussed so far, as a lower temperature allows the model to focus on exploitation earlier than if the temperature were higher.

In contrast, the results for the Knapsack problem differed. It appears that the slower Exponential decay was preferred.

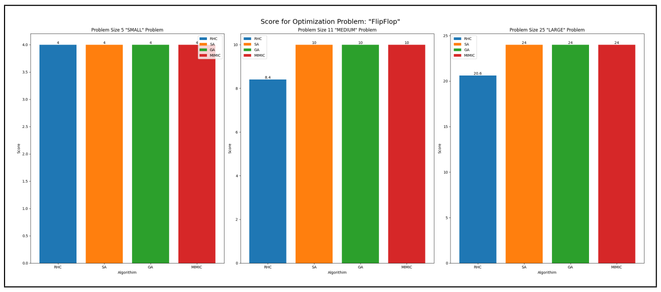

Figure 6. Score by Algorithm by Problem Size FlipFlop Problem

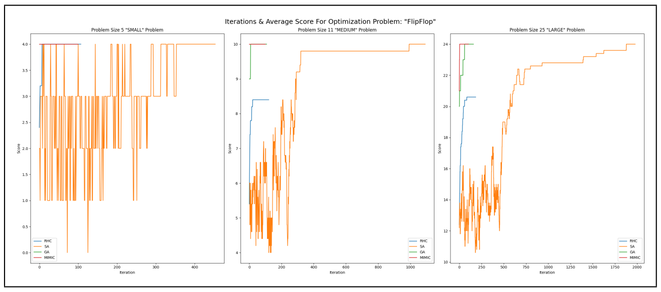

Figure 7. Iterations and Score by Algorithm by Problem Size FlipFlop Problem

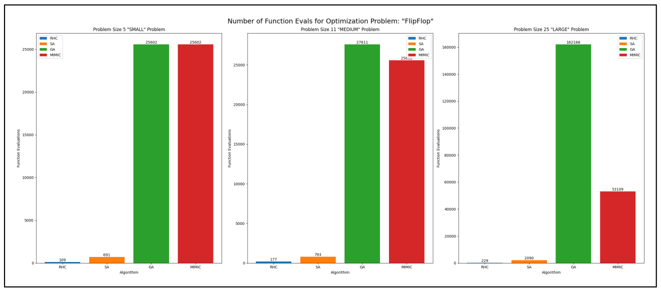

Figure 8. Function Evaluations by Algorithm by Problem Size FlipFlop Problem

Although we cannot conclusively state why this occurred we do have a leading hypothesis. The KnapSack problem is significantly more complex than the FlipFlop problem, so, it is possible that the a slower temperature decay meant that the optimization algorithm found the additional explorations beneficial, striking a better balance between exploration and exploitation. It must be noted that too high of an initial temperature T_0 value did not lead to better results. We believe this may be due to the algorithm entering a zone of imbalance in terms of exploration and exploitation, focusing too heavily on exploration prevented it from optimizing to the global optimum.

4.1.3 FlipFlop Problem Results and Conclusions

Observing Figure 6, we note that SA performed quite well in terms of score. The algorithm was successfully able to identify the maximum value (global optimum) for all three problem sizes which matched the performance of the GA

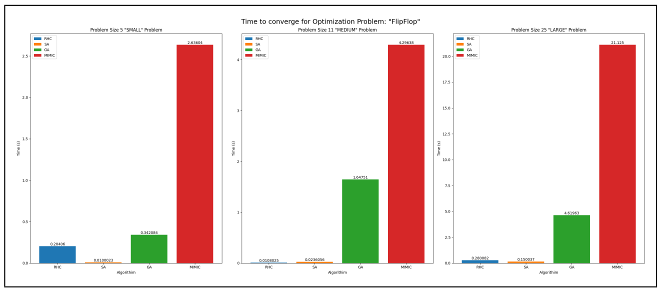

Figure 9. Wall Clock Convergence Time by Algorithm by Problem Size FlipFlop Problem

and MA strategies. Given that SA is a significantly simpler optimization algorithm, this is an excellent result. We also note that SA performed better than RHC, which showcases that the problem was not trivial.

Figure 7 provides some excellent insight into the workings of SA, especially when we observe the "Small" problem size. We note how the score of SA appears to bounce around, even after the max value has been achieved, this shows how in this stage the algorithm is favoring exploration over exploitation. We must note here that SA did perform significantly more iterations than RHC, GA and MA. This ties in with the fact that the temperature value dictates the balance between exploration and exploitation. We believe that careful tuning of the initial temperature value and decay should be able to drastically reduce the number of iterations. With that said, we must note that we did repeat this with many initial temperature values and still noted that SA performed significantly more iterations than the other algorithms for this problem.

Figure 8 and Figure 9 show the real strengths of SA. We note that compared to GA and MA, SA performed significantly fewer function evaluations, and, perhaps more importantly, had the best average wall clock time for convergence out of all optimization algorithms.

We believe that the strong performance of the SA algorithm is closely tied to the characteristics of the optimization problem. The problem space of the FlipFlop problem is relatively simple, with the major issue being an abundance of local optima, and, a few plateaus. To find the global optimum for this problem, an algorithm does not necessarily have to develop a deep understanding of the underlying problem space (as done so deliberately by MA and more "unconsciously" by GA), but it must have the ability to adequately explore the problem space, or risk getting stuck at a local optimum (as seen by the RHC algorithm). For these reasons SA is a good match for this problem, utilizing the temperature parameter to strike an ideal balance between exploration and exploitation.

4.2. Genetic Algorithm

4.2.1 Problem Introduction - KnapSack

To demonstrate the effectiveness of the Genetic Algorithm, we selected the KnapSack problem as a suitable optimiza-

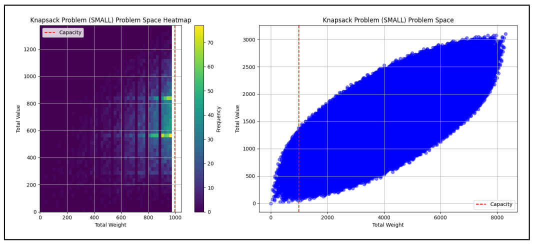

Figure 10. Problem Space for the KnapSack Optimization Problem

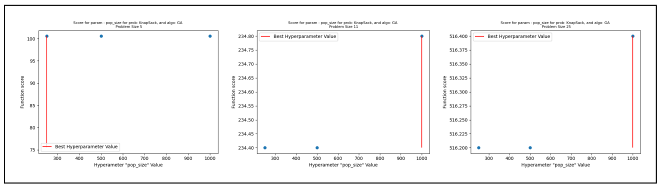

Figure 11. GA Population Size Tuning for KnapSack problem

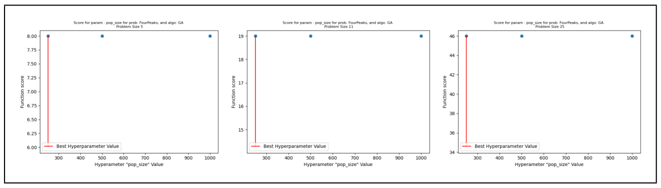

Figure 12. GA Population Size Tuning for FourPeaks problem

tion challenge. The primary aim is to maximize the total value of selected items while adhering to the weight constraint. Figure 10 illustrates an example problem space for the KnapSack optimization problem, highlighting its challenging and intricate nature. Comparing this problem space with that of the FlipFlop problem shown in Figure 2, we observe a notable increase in complexity.

4.2.2 Hyperparameter Tuning for the Genetic Algorithm

For GA, the population size, population breeding percent, and mutation probability were altered.

Review of relevant literature leads us to believe that a larger population should allow the algorithm to possess greater exploratory power. Additionally, a larger population allows for more diversity in the samples, which can result in higher chances of identifying the solution, especially in complex problem spaces [3]. In our testing, we did observe some evidence supporting this when we compared hyperparameter tuning plots between the KnapSack problem and the FourPeaks problem. Figure 11 shows that a higher population did provide better results for the KnapSack optimization problem when problem size was increased (Note: The problem sizes here are not the same as the ones in the final KnapSack experiments). This is in contrast to the FourPeaks problem, the results of which are highlighted in Figure 12. For FourPeaks, all population sizes led to convergence, but it can be noted that larger population sizes led to less efficient

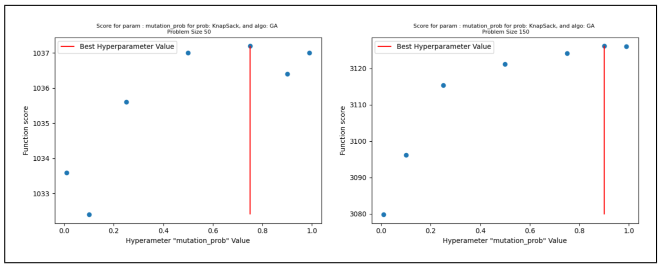

Figure 13. GA Mutation Probability Tuning for KnapSack problem

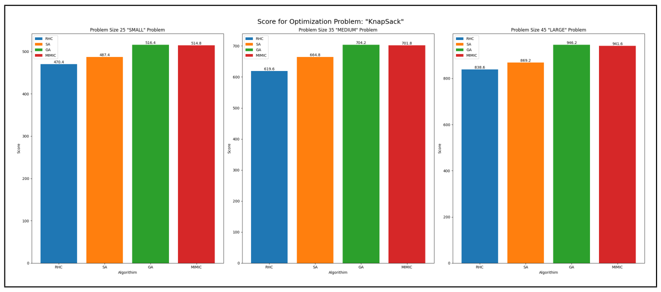

Figure 14. Score by Algorithm by Problem Size KnapSack Problem

or slower convergence, this is likely due to more function evaluations needed for a larger population.

Mutation probability was also changed. Mutation probability again highlighted the balance required between exploration and exploitation. A very high mutation probability led to the algorithm performing more akin to a random search, (too high of an emphasis on exploration - still good convergence but took significant time) , whereas, too low of a mutation rate resulted in the algorithm converging onto a local optimum (too high of an emphasis on exploitation). Figure 13 showcases this effect. A similar concept applies to our third hyperparameter, population breeding percent, swapping out more of the population with offspring placed more emphasis on exploration, whereas, keeping the a higher portion of elites in the next steps population placed more emphasis on exploitation. As with previous discussion relating to this trade-off, a balance is needed for optimal performance.

In terms of the efficiency/speed of convergence, initially, we believed that a lower mutation rate (less randomization in search) and a lower population breeding percent should result in faster convergence (with an increased risk of converging onto a local optimum). However, this was not observed in our results. There was no clear relationship, indicating that this relationship may be more complex than what we initially believed and may be more effected by initial state, or, other confounding variables.

4.2.3 KnapSack Problem Results and Conclusions

We note from Figure 14 that GA had the highest average fitness score out of all algorithms. Figure 15 highlights how GA converged to the global optimum for all three problem

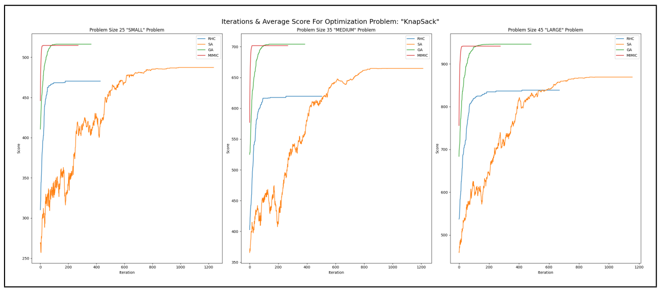

Figure 15. Iterations and Score by Algorithm by Problem Size KnapSack Problem

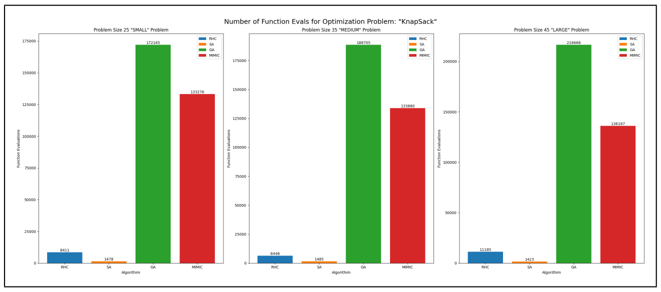

Figure 16. Function Evaluations by Algorithm by Problem Size KnapSack Problem

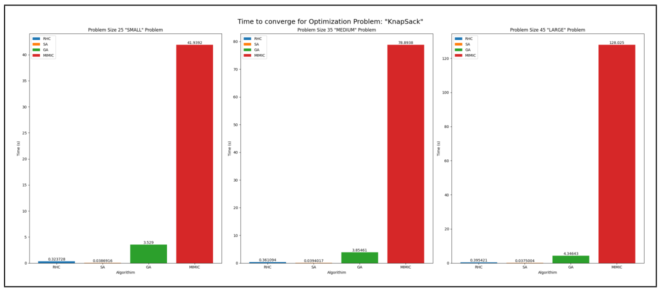

Figure 17. Wall Clock Convergence Time by Algorithm by Problem Size KnapSack Problem

size's within a small number of iterations, not significantly far off from MA. We do note, however, that GA did have the highest number of function evaluations as seen in Figure 16. Finally, GA performed quite impressively with regards to wall clock time highlighted in Figure 17.

he graphs lead to some interesting conclusions that warrant further exploration. We note from the results that RHC and SA perform relatively poorly, which is somewhat expected. RHC relies on restarts to explore the problem space, but this method is often suboptimal. As for SA, the temperature parameter controls exploration, which is again not optimal. We observe from the plots that early on, SA makes impressive gains, as seen in Figure 15, but it gets stuck in local optima once the temperature cools. It may be possible to achieve better performance through extremely careful tuning of the number of restarts for RHC, and, the temperature parameters for SA. However, both of these algorithms will still suffer from a fundamental flaw of having no mech-

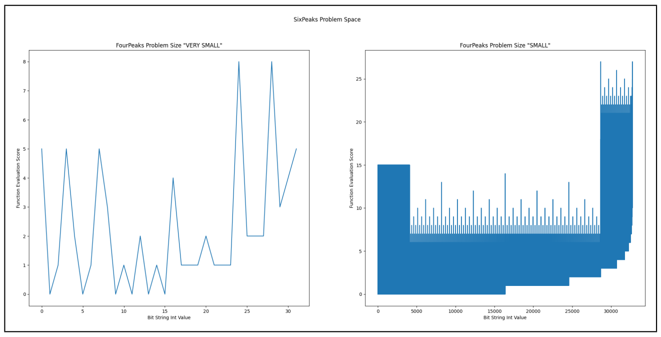

Figure 18. SixPeaks Problem Space Visualized

anism to capture the structure of the problem space. GA and MA do have this mechanism in-place, GA is able to code the underlying structure by it's chromosomes, and MA does this probabilistically. This leads to both algorithms performing relatively well. However, GA, is able to outperform MA because, for this problem, we believe GA is better able to balance exploration versus exploitation. With a random crossover point and a large diverse population (including elites, dregs, and offspring), GA effectively traverses the combinatorial problem space, allowing it to quickly test a large subset of potential combinations before converging. MA excels in building an excellent probabilistic understanding of the structure, but we believe it's less effective in testing a large number of combinations before converging onto a solution.

4.3. MIMIC Algorithm

4.3.1 Problem Introduction - SixPeaks

The SixPeaks problem presents a challenging optimization task. Figure 18 illustrates the problem space, revealing an abundance of local optima, numerous plateaus, and a generally intricate structure. Due to these characteristics, SixPeaks poses a difficult challenge for optimization algorithms, where simpler strategies may struggle to find optimal solutions.

4.3.2 Hyperparameter Tuning for the MIMIC Algorithm

In the case of MA we tuned the population size and the keep percent hyperparameter. Similar to other random optimization algorithms these hyperparameters are tuned to provide balance between exploration vs exploitation. A higher population of samples allows for more explorative power (similar to GA). The keep percent focuses on balancing exploration vs exploitation, a lower keep percent value would focus more on exploration, ensuring more samples are from the new distribution the algorithm has calculated. In our own testing we found that the population hyperparameter behaved as expected, more complex populations benefited from increased population size, but, simpler problems preferred smaller populations due to faster/more efficient con-

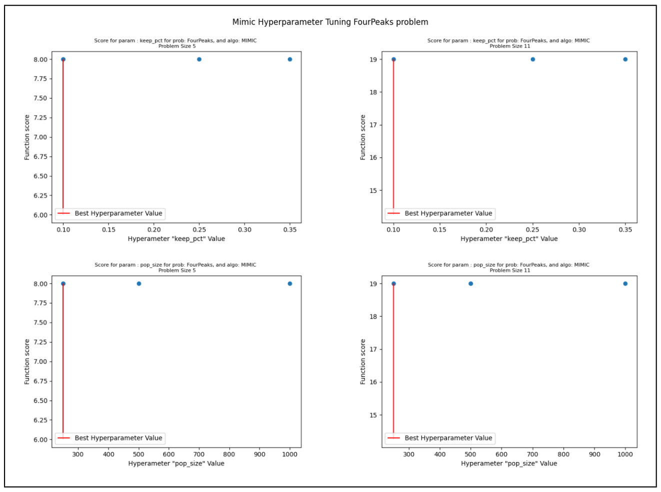

Figure 19. Hyperparameter Tuning for MIMIC on FourPeaks

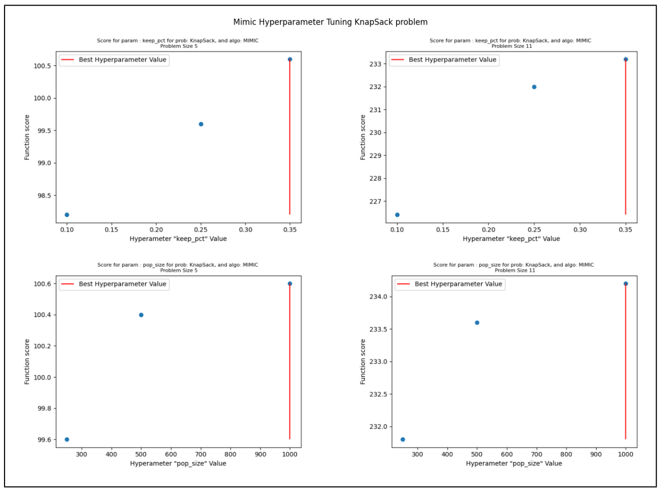

Figure 20. Hyperparameter Tuning for MIMIC on KnapSack

vergence. This result can be seen when we compare Figures 19 and 20. Our initial hypothesis for the keep percent hyperparameter was that simpler problems would prefer a higher keep percent value, as we believed that would lead to faster/more efficient convergence (more focus on exploitation). However, that was not the case, it appears KnapSack preferred a higher keep percent value, and, FlipFlop preferred a lower keep percent value. This is an interesting result and warrants further investigation. We believe this highlights that the keep percent hyperparameter may have a more complex effect on model behaviour than we initially assumed. Another possible reason could be that the initial state of the model/population is having an interaction effect with the keep percent value.

4.3.3 SixPeaks Results and Conclusions

Figures 21 through 24 showcase the results of employing our algorithms on the SixPeaks problem. Immediately we note the superior performance of MA. MA was able to con-

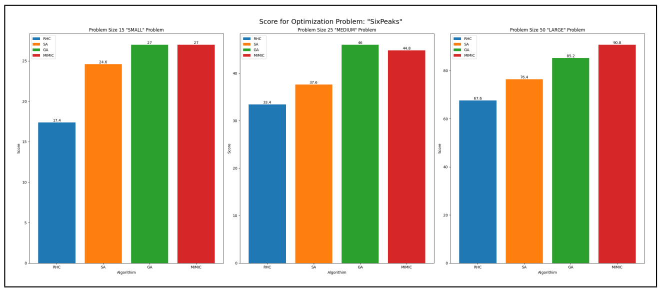

Figure 21. Score by Algorithm by Problem Size SixPeaks Problem

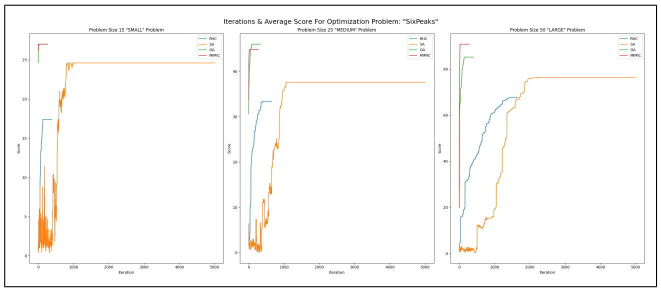

Figure 22. Iterations and Score by Algorithm by Problem Size SixPeaks Problem

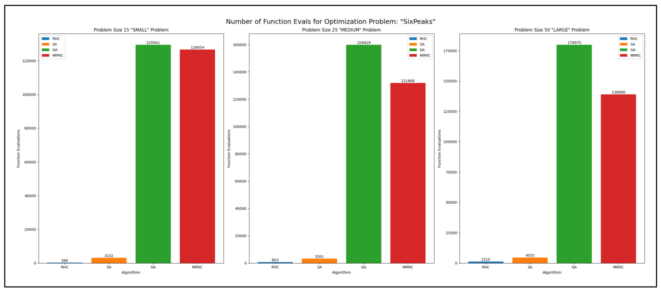

Figure 23. Function Evaluations by Algorithm by Problem Size SixPeaks Problem

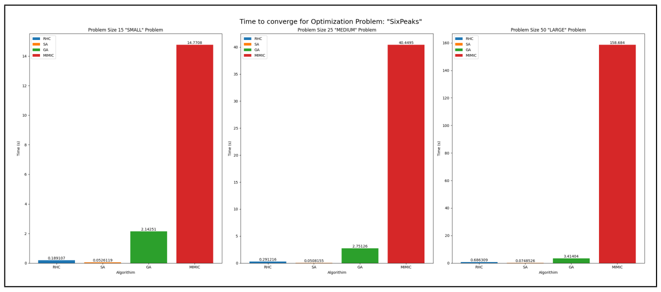

Figure 24. Wall Clock Convergence Time by Algorithm by Problem Size SixPeaks Problem

verge on the global optimum in nearly all runs. The results do show GA performing better than MA for the medium problem size in terms of score, but it must be remembered that this is an average of many runs and is likely due to randomness. We believe the results on the large problem size further confirm this, as we note that MA performs far better than any other algorithm. Additionally, MA is able to converge on the optimal solution in the fewest number of iterations. MA also requires significantly fewer functional evaluations than GA, which is the next best scoring algorithm. It must be noted that MA does take considerable wall clock time to converge, significantly greater than the other algorithms. This is one drawback of MA.

We believe MA's exceptional performance is due to its ability to build a probabilistic understanding of the problem space. Constructing an effective probabilistic model of the underlying problem space allows the algorithm to determine regions of the problem space that look promising in terms of where the global optimum is located. Additionally, the ability to build probabilistic relationships between variables further guides the model's search, improving its ability to identify an optimal solution. The complex problem space of SixPeaks results in the other algorithms converging on a local optimum, this is especially apparent with a large problem size. MA, however, is able to identify the optimal value, even at large problem sizes, due to it's ability to capture underlying structure. Indeed, relevant literature corroborates with our findings [1].

5. Neural Network Training

5.1. Gradient Descent

For the Gradient Descent algorithm, we observed performance similar to Assignment 1, as supported by the graphs in Figure 25. However, it's essential to note that we had to simplify the model architecture due to limitations in layer types, regularization options and activation functions.

In terms of bias and variance, we observed that the model training tended to exhibit low bias coupled with high variance, even with a simpler structure. Similar to Assignment 1, increasing the learning rate resulted in signs of low bias and high variance. To address this, lower learning rates were preferred. The Learning Curve display showed an improving test score, but it also indicated that the model still showed signs of overfitting. This was further confirmed by examining the loss curves, which showed a significant degree of overfitting across all 5 runs, with the loss function esentially decreasing to zero. Despite attempts to mitigate this by decreasing the number of iterations, overfitting persisted. Our results suggest that the loss function rapidly falls to zero within a few iterations. We attribute this to two factors: firstly, the inability to properly regularize the architecture, and secondly, the use of gradient descent optimization instead of the more conventional stochastic gradient descent (SGD). We believe that the stochastic nature of SGD helps reduce model overfitting, as evidenced by results from the subsequent randomized optimization sections.

5.2. Random Hill Climbing

For RHC all plots generated from our experiments are presented in Figure 26. As with all algorithms, we started our experimentation by performing hyperparameter tuning. Similar to other algorithms, the learning rate was tuned, and,

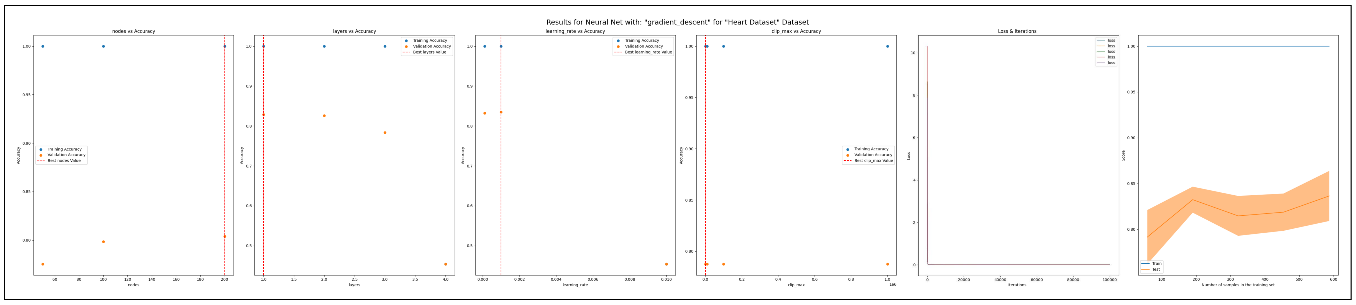

Figure 25. Neural Network Training Gradient Descent

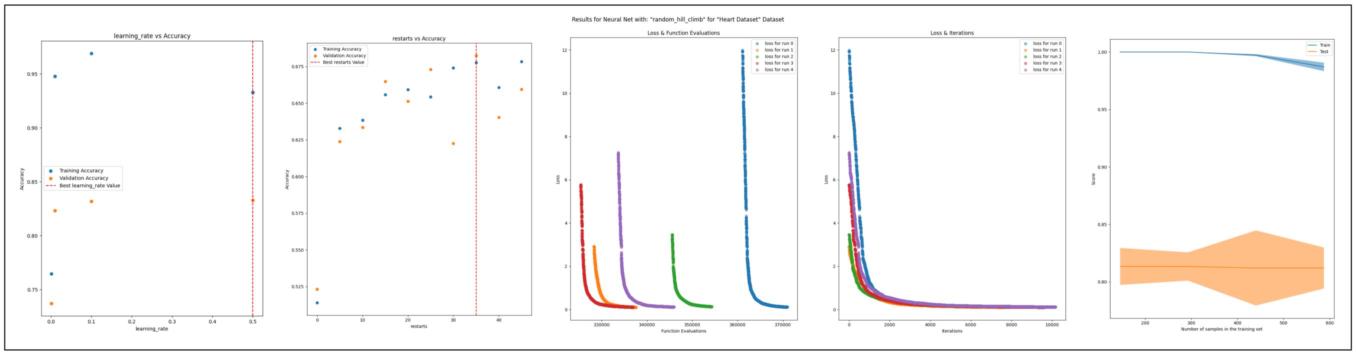

Figure 26. Neural Network Training Random Hill Climbing

specific to RHC, the number of restarts was tuned. From the learning rate plot we note that too low of a learning rate led to poor performance, this is likely due to the model being unable to converge due to too small of a step size. Interestingly, as opposed to GD, a large learning rate still had great performance, we believe this is due to the model being able to "restart" the optimization many times, thus, even if the model takes a poor step that leads to it converging on a local minimum it can just simply restart. In terms of time to convergence, we noted that the algorithm converged significantly faster by all metrics (number of iterations, fevals, wall clock time) when the learning rate was higher, and, this makes sense a larger step size leads to it converging on a minimum sooner.

We observed that there seemed to be a threshold value for restarts hyperparameter. Below this threshold, increasing the number of restarts improved model performance. However, beyond this threshold, increasing the number of restarts did not improve performance. We believe this may be due to the fact that given the simple nature of the algorithm it always converges to some local minimum eventually. While increasing restarts initially helps overcome the first batch of local minima, it eventually succumbs to another, and arbitrarily increasing restart size does not mitigate this phenomenon.

5.3. Simulated Annealing

The graphs presented in Figure 27 help showcase our experimentation, and results for SA. From the graphs we note that the model still performed well even with a high learning rate, contrasting with our observations for GD and resembling the performance of RHC. We believe this may be due to the fact that SA is able to maintain a large emphasis on exploration by incorporating significant randomization through the use of it's temperature parameter, thus, taking large wrong steps can be rectified for in subsequent iterations/steps (as randomization may just move the function to a basin of attraction for the global minimum). However, it does appear this increased level of stochasticity/exploration had a significant drawback, as the SA optimized model required significantly more iterations than the other randomized algorithms to converge.

From the learning curves we observe that the SA trained model was significantly less inclined to overfitting, even when trained on a small dataset, when compared to the GD model. This likely ties in with what we stated earlier, by ensuring there is some form of stochasticity in the optimization process, the model is significantly less likely to overfit the training data.

The graphs also emphasis that the network performed better when the SA algorithm was paired with a high initial temperature and a slower decay, indicating that the model benefited from the extra exploration.

5.4. Genetic Algorithm

The last optimization algorithm we implemented was GA. Figure 28 showcases important graphs for this algorithm. Here, we note that GD preferred a high learning rate. We believe this ties in with GD being able to balance exploration vs. exploitation very well for complex problems. By

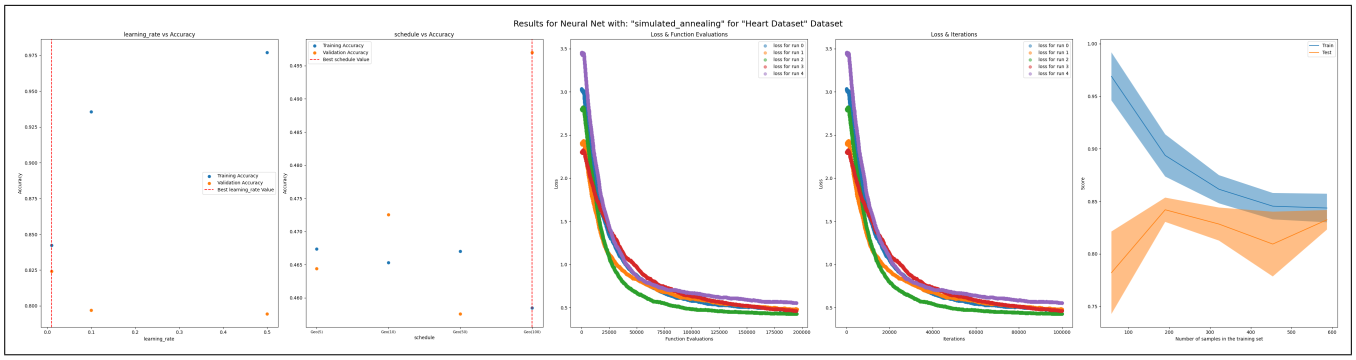

Figure 27. Neural Network Training Simulated Annealing

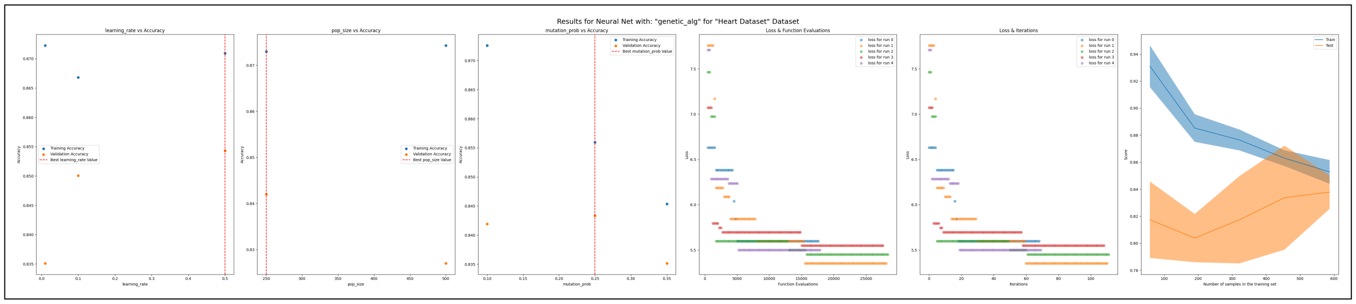

Figure 28. Neural Network Training Genetic Algorithm

utilizing a diverse population at every step, it's able to effectively explore the problem space and facilitate a larger step size.

For population size, we note that there appears to be a threshold. A population size that is too high seems to result in overfitting. We believe this may be because a too-high population causes the algorithm to capture the effects of noise in the problem space when it's unconsciously encoding the structure of the problem space.

For the mutation rate, too low of a mutation rate resulted in a model that had too low variance, and/or has overfit the training data. However, a model with too high of a mutation rate resulted in a model with too high a bias, leading to underfitting. This is likely because with too high of a mutation rate the algorithm is acting more akin to random search and is unable to effectively converge onto the global minimum. Whereas, too low of a mutation rate likely results in an effect similar to the one observed in GD, too little stochasticity makes the model learn the training data too well, even identifying noise as valid patterns, impacting model generalizability.

Finally we note the interesting shape of the loss curve. The loss curve starts at a high loss likely due to random initialization of the population, from then on it makes steady improvements, but, does appear to get trapped in a local optima for many iterations. We believe this is due to the population converging on the local optima, and, it being stuck there until stochastic processes are able to move it out. The loss curve also does indicate a higher final loss than observed for GD and SA, this likely means that the model has converged on a suboptimal solution.

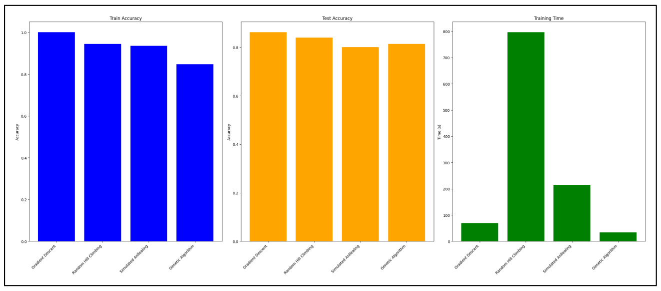

Figure 29. Neural Network Training Final Results

5.5. Neural Network Final Results and Conclusions

Figure 29 presents the average training accuracy, the average test accuracy, and wall clock time for all algorithms. It's evident from the graphs that GD appeared to overfit the training data compared to the other algorithms (averaging a 1.0 accuracy across multiple runs). This is likely due to the GD algorithm's inability to benefit from stochasticity, leading to it quickly converging on a weight value that resulted in the zero training loss. While this demonstrates excellent convergence, it unfortunately indicates that the model has overfit, as the test error rate of the model remains relatively high. . Based on the training scores, we can also confirm that GA did indeed converge onto a local optimum, given that it has the lowest training accuracy and highest loss after convergence.

Turning to the test scores we note all algorithms performed well, GD performed the best, but, the scores were not significantly different. The reason GD may have had an edge over the other models could be due to the fact that it is better adapted to handle continuous valued optimization problems.

Finally, upon reviewing the wall clock times, we observed that GA converged the fastest out of all three algorithms. We attribute this rapid convergence to its unmatched explorative power, allowing for quick and detailed exploration of the problem space (however, it did appear to converge onto a local minimum as indicated by the higher loss and lower training score). RHC had the slowest wall clock time, likely due to us setting a number of restarts. High wall clock time for SA can also be noted from the graph. We believe this could be because of the temperature value remaining high for many iterations, slowing down convergence.

The strong performance of the random optimization algorithms suggests that our underlying problem space wasn't overly complex. Even our simpler RHC algorithm was able to perform well. However, we note that SA and RHC took significant wall clock times to converge, indicating that the completely randomized approach of these algorithms was not optimal. Shifting our focus to GA, we observe that it converged even faster than GD but had worse performance, indicating a need for more careful hyperparameter tuning. The rapid convergence of GA underscores the benefit of capturing the underlying structure of the problem space. In conclusion, while GD outperformed the randomized algorithms, the other algorithms show promise, and GD's strong performance may be attributed to its suitability for tackling continuous-valued problems. Nevertheless, our results underscore the viability of other algorithms beyond GD in machine learning problems.

6. Conclusions and Next Steps

We formulated several hypotheses in the introduction of our paper, based on our literature review, and now, we can assess their validity. Our first hypothesis was confirmed through experimentation on the SixPeaks problem. MIMIC significantly outperformed other algorithms, converging to better optimum values, on average, and requiring the fewest number of iterations to do so.

Our second hypothesis also proved true. Simulated Annealing's excellent performance on the FlipFlop problem demonstrated its ability to solve simpler optimization problems quickly and efficiently. We also observed that temperature scheduling might require careful tuning for complex problems, as indicated by our hyperparameter tuning plots for the KnapSack problem..

The Genetic Algorithm's strong performance on the KnapSack problem highlighted the algorithm's explorative power and confirmed our hypothesis regarding its effectiveness in complex combinatorial problems.

Our next hypothesis pertained to the neural network sections, where we anticipated that the absence of stochasticity in Gradient Descent would result in faster convergence, albeit potentially leading to convergence on a local minimum. However, our findings only partially supported this hypothesis. While the lack of stochasticity did indeed accelerate convergence, our results did not suggest convergence on a local minimum. Additionally, we hypothesized that this absence of stochasticity would make the model more susceptible to overfitting, which our results corroborated. We observed the loss function rapidly decreasing to 0, yet the test set error remained relatively high.

Contrary to our expectations, Randomized Hill Climbing performed well in the neural network section, despite our initial belief that it would struggle due to the continuous optimization problem. This discrepancy suggests that the problem space for our neural network model may have been simpler than initially assumed.

Although our experimentation was detailed and in-depth, not all our questions were answered, and new ones arose from our results. The strong performance of Randomized Hill Climbing on our neural network, the behavior of keep percent for MIMIC, and the effects of mutation rate and percentage of breeding population are all areas that require further exploration.

References

[1] Jeremy De Bonet, Charles Isbell, and Paul Viola. Mimic: Finding optima by estimating probability densities. In M.C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1996. 7

[2] Fedesoriano. Heart failure prediction dataset, 2021. 1

[3] Stanley Gotshall and Bart Rylander. Optimal population size and the genetic algorithm. *Population*, 100(400):900, 2002. 4

[4] Marco Locatelli. *Simulated Annealing Algorithms for Continuous Global Optimization*, pages 179–229. Springer US, Boston, MA, 2002. 1

[5] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. 1

[6] Ye Tian, Haowen Chen, Xiaoshu Xiang, Hao Jiang, and Xingyi Zhang. A comparative study on evolutionary algorithms and mathematical programming methods for continuous optimization. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2022. 1