Supervised Learning

Ian Dover

November 9, 2023

1 Algorithms

1.1 Random Hill Climbing

Random hill climbing stands as a distinct approach in contrast to the widely-used backpropagation method for modifying the weights of a neural network. Unlike backpropagation, which leverages the chain rule to systematically compute gradients and adjust all weights in unison based on the error of the model, random hill climbing adopts a more exploratory method. Specifically, it sequentially navigates through each weight, and for each weight, it introduces random perturbations. Following this, the algorithm evaluates if the perturbation leads to a reduction in the loss function. If the new configuration results in a lowered loss, the perturbed weight is retained; otherwise, the algorithm reverts to the previous weight. This process continues iteratively, one weight at a time, seeking to find weight configurations that continually minimize the error. While this method offers a unique approach, bypassing the need for gradient computations, it can be less efficient as it relies on random sampling and might require many iterations to achieve optimal or near-optimal configurations.

1.2 Simulated Annealing

Simulated annealing presents itself as a distinct alternative to the gradient-driven backpropagation method, offering a unique approach to optimizing neural network weights. Central to this technique is the concept of "temperature", a parameter inspired by the annealing process in metallurgy. This temperature serves a dual role in the algorithm:

(1) Magnitude of Perturbation: It dictates the magnitude of random perturbations introduced to the existing neural network weights. At higher temperatures, the adjustments are more aggressive, allowing the algorithm to explore a wider solution space. This expansive exploration can aid in evading local minima and venturing into areas of the solution landscape that might be left unexplored by more deterministic methods.

(2) Acceptance of Sub-optimal Solutions: The temperature also governs the probability of endorsing a set of weights that, counterintuitively, performs worse than the current configuration. This acceptance is based on a probabilistic criterion, allowing the algorithm to sometimes favor worse solutions with the rationale of escaping potential suboptimal local configurations. The likelihood of this acceptance dwindles as the temperature decreases.

As the optimization process progresses, the temperature undergoes a "cooling" phase, decreasing incrementally based on a predefined schedule. This cooling mechanic translates to a gradual shift in the algorithm's behavior. In the early stages, characterized by higher temperatures, the approach is more exploratory, frequently accepting diverse weight configurations. However, as the temperature wanes, the algorithm becomes more conservative. The perturbations to the weights get smaller, and the readiness to accept sub-optimal solutions diminishes. This phase emphasizes refining the current best solutions, honing in on the optimal weight configuration with more precision.

This blend of broad exploration followed by meticulous exploitation allows simulated annealing to provide a balanced and alternative strategy to the typical gradient-based weight adjustments seen in backpropagation.

1.3 Genetic Algorithms

Genetic algorithms are computational strategies inspired by the fascinating intricacies of natural evolution. At their core, these algorithms use encoded representations known as "chromosomes" to define potential solutions to problems. To start the evolutionary journey, an initial varied set of these chromosomes is created, forming a diverse "population".

This population undergoes evaluation through a specially designed "fitness function". This function quantifies the effectiveness or aptness of each solution in addressing the problem at hand. Those chromosomes that achieve higher fitness scores are perceived as superior solutions. Following nature's principle of survival of the fittest, these high-ranking solutions are preferentially selected as "parents" for the next generation.

The production of new solutions, or offspring, happens through a process termed "crossover". Here, traits from two parent chromosomes are merged, inheriting features from both, potentially crafting even better solutions. But genetic algorithms don't solely rely on recombination of existing traits. They integrate a mechanism of "mutation" to ensure the algorithm doesn't get trapped in solution stagnation. By introducing small, random alterations to a subset of offspring chromosomes, mutation infuses the population with fresh genetic material, enabling exploration of new solution avenues.

With each cycle, the genetic algorithm evolves its population, gradually refining and improving the solutions. This iterative process is executed over several generations, each time assessing, selecting, and breeding new potential answers to the problem. The algorithm's journey typically culminates either after iterating for a predefined number of generations or when notable improvements become scarce.

In emulating the evolutionary mechanisms of nature, genetic algorithms provide a balanced approach to computational problem-solving, harmoniously blending the exploration of novel solution territories with the exploitation of already-identified promising regions.

1.4 MIMIC

Mutual-Information-Maximizing Input Clustering (MIMIC), introduces a significant paradigm shift from the traditional evolutionary algorithms, especially when compared to the genetic algorithms. Genetic algorithms, which have been the cornerstone of evolutionary optimization for decades, rely heavily on mechanisms like crossover and mutation to produce new generations of solutions, often referred to as "chromosomes". These mechanisms operate based on the foundational principle of natural evolution - survival of the fittest. The crossover combines features of two parent chromosomes, while mutations introduce small random changes to ensure diversity.

However, MIMIC deviates from this tradition by implementing a sophisticated model-based selection process. Instead of relying on the somewhat blind processes of crossover and mutation, MIMIC opts for a more informed approach. It constructs a detailed statistical model, often a Bayesian network, which encapsulates the dependencies between various hyper-parameters in a chromosome. These dependencies are not just superficial associations but represent complex inter-plays between the parameters, capturing the intricate structure of promising solutions. Such a representation ensures that the algorithm is well aware of how each parameter influences the others, providing a more holistic understanding of the solution space.

Once this model is in place, MIMIC does not go back to traditional evolutionary techniques. Instead, it uses the constructed probabilistic model to produce the next generation of chromosomes. By sampling from this model, the algorithm can generate new potential solutions that are consistent with the observed patterns and dependencies of high-performing chromosomes from the previous generation. This approach ensures that the generated solutions are not just random guesses but are informed by the inherent structure and patterns discovered within the top-performing solutions.

2 Optimization Problems

2.1 Flip Flop Problem

The Flip Flop (FFP) optimization problem deals with the analysis of alternations in a bit string. Specifically, it involves counting the number of times the bits in the string alternate, meaning that each time a bit changes from one value to another in consecutive digits, it's counted as 1. For instance, if you were to examine the bit string '0101', there would be three alternations: from the first 0 to the second 1, from the second 1 to the third 0, and from the third 0 to the fourth 1. This alternation count becomes the measure or score for the given bit string.

The ultimate objective of the FFP problem is to find a bit string that has the maximum possible number of alternations, making it the most "fit" string in the context of this problem. The ideal, or maximum fitness bit string, would therefore consist entirely of alternating digits. So, for a string of length 'n', the highest score possible would be 'n-1'. For instance, a 4-bit string with the highest fitness would be '0101' or '1010', both of which have a score of 3.

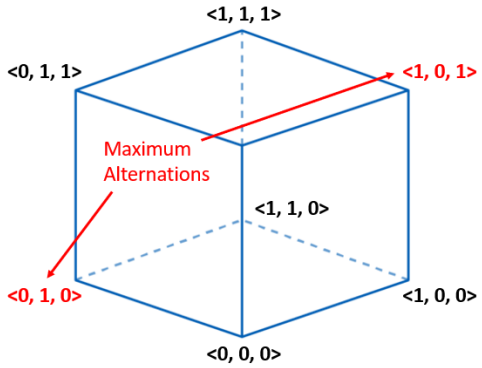

Figure 1: Flip Flop Problem

2.2 Four Peaks Problem

The Four Peaks optimization problem is a classic test problem in the realm of evolutionary algorithms and genetic programming, designed to present a landscape that has multiple local optima. The challenge lies in its unique fitness landscape, which contains two main peaks and two smaller peaks, thus making the search for the global optimum non-trivial.

To understand the problem more clearly, consider a bit string of length 'n'. The fitness function of the Four Peaks problem is defined in such a way that it rewards two things: the number of leading 1s and the number of trailing 0s in a string. The two primary peaks correspond to the strings with all 1s and all 0s. However, there's a twist. To get the reward for trailing zeros, there should be a certain number of leading ones and vice versa. This condition creates a trade-off, making it challenging to navigate the search space effectively. As a result, algorithms can often get trapped in one of the local optima, and finding the global maximum becomes a significant challenge. The presence of these four peaks (two major and two minor) gives the problem its name and exemplifies the intricacies of navigating complex optimization landscapes.

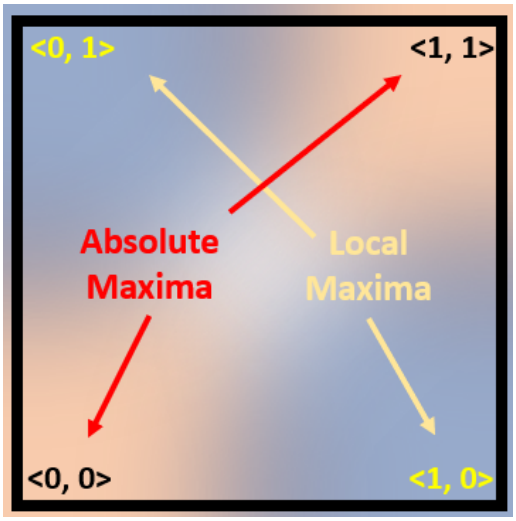

Figure 2: Four Peaks Problem

2.3 One Max Problem

The One Max optimization problem is a simple problem often used in the study of evolutionary algorithms and genetic pro-

gramming. Its straightforwardness makes it an excellent benchmark for testing and tuning algorithms, especially for beginners in the field.

At its core, the One Max problem involves a bit string of a specified length 'n'. The objective is to maximize the number of 1s in the string. The fitness of a given bit string is computed by simply counting the number of 1s present in it. For instance, for the string '010110', the fitness would be 3, as there are three 1s. The ultimate goal is to find a bit string that consists entirely of 1s, making it the optimal solution. Given its clear and singular peak in the fitness landscape, the One Max problem doesn't have the complications of local optima. However, despite its apparent simplicity, it provides a foundational platform for understanding the mechanics and behaviors of optimization algorithms, as they navigate towards the global optimum.

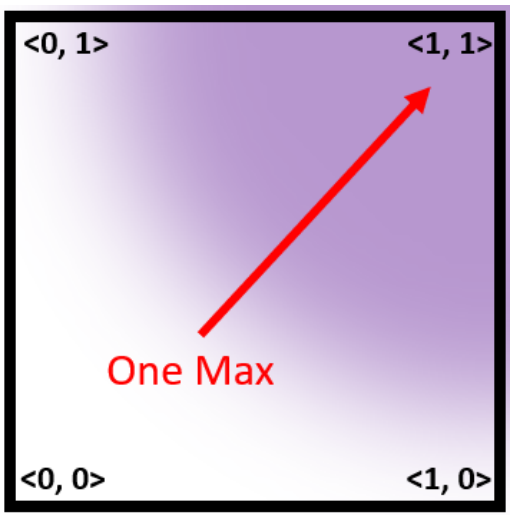

Figure 3: One Max Problem

3 Algorithm Performance

3.1 Flip Flop Problem

In the context of optimizing the flip flop problem of Figure (4), an intricate analysis was conducted, examining various optimization algorithms' performance over a span of 50 iterations. The evaluation count was plotted on the x-axis, while the fitness values were depicted on the y-axis.

Four distinct algorithms were engaged in this exploration, each varying in the number of evaluations per iteration. Notably, both the Random Hill Climbing algorithm and Simulated Annealing exhibited a relatively consistent behavior with approximately 50 evaluations of curve data. Remarkably, their performance closely mirrored each other, with Random Hill Climbing marginally outperforming Simulated Annealing throughout the entirety of the experiment.

Conversely, the MIMIC algorithm displayed a considerably wider range of evaluations, spanning from 50 to 600 per iteration. Intriguingly, this algorithm maintained a consistent fitness curve, essentially a horizontal line, which consistently remained slightly below the peak performance of Random Hill Climbing.

The Genetic Algorithm, characterized by the highest number of evaluations within the 50 iterations, ranged from 400 to 4000 evaluations. Astonishingly, this algorithm emerged as the top-performing one among the four.

Examining the graphical representation of the data, it is discernible that the majority of the algorithms appear to follow a similar global curve in relation to the number of evaluations. This suggests the possibility of extrapolating this curve to approximate the behavior of each algorithm.

Most notably, it appears that the Genetic Algorithm was progressively converging towards a fitness value of 70 as the number of evaluations increased, indicating its potential to reach an optimal solution for the flip flop problem. This convergence implies that with more evaluations or iterations, the Genetic Algorithm may continue to refine its solution and ultimately approach the desired fitness value.

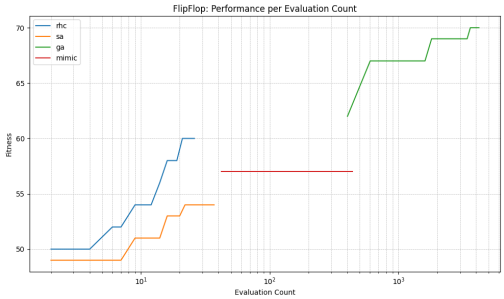

Figure 4: Fitness per Evaluation

In Figure (5), when scrutinizing the fitness progression per iteration, we discern a noteworthy parallel between the performance of MIMIC and the Genetic Algorithm. Both exhibit a similar trend in their iterative performance, with the Genetic Algorithm demonstrating superior performance, particularly in the earlier iterations.

A similar pattern of comparative performance emerges when examining the disparity between Random Hill Climbing and Simulated Annealing. It's worth noting that Simulated Annealing employs a strategy that balances exploration and exploitation through the use of a temperature parameter. Conversely, the Random Hill Climbing algorithm is solely exploitative, with the omission of a restarts parameter in this experiment. This distinction potentially elucidates why Random Hill Climbing outperformed Simulated Annealing in the initial iterations. The former focuses primarily on exploiting the current best solution, seeking the path of least resistance, while the latter, with its randomized exploration, tends to slow down its convergence.

An intriguing aspect to consider is that both Simulated Annealing and Random Hill Climbing algorithms seem to plateau at a sub-optimal fitness level. This observation raises questions about whether these algorithms may require more iterations to uncover the optimal fitness function, or if they are inherently constrained in their ability to discover the best fitness function.

In scientific terms, this phenomenon can be explained as follows: Convergence, in the context of optimization algorithms, pertains to the algorithm's ability to approach an optimal solution over successive iterations. The Genetic Algorithm's superior performance in earlier iterations suggests rapid convergence, while the slower progress of MIMIC may indicate the need for more iterations to reach the same level of performance.

Simulated Annealing's trade-off between exploration and exploitation, controlled by the temperature parameter, can affect its convergence rate. The absence of a restarts parameter in Random Hill Climbing emphasizes its purely exploitative nature, which can lead to quick initial gains but potentially hinder the exploration of better solutions.

The plateau in fitness observed in both Simulated Annealing and Random Hill Climbing suggests they may have reached local optima and could benefit from additional iterations or modifications to escape these suboptimal solutions and find the true global optimum.

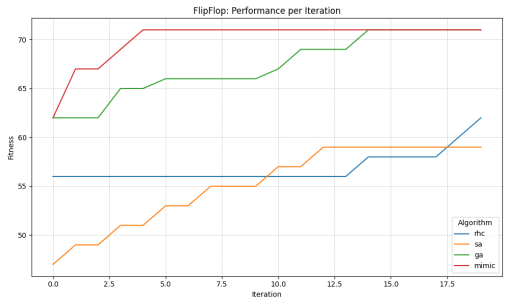

Figure 5: Fitness per Iteration

In Figure (6), a comprehensive exploration of optimization problem dynamics is presented, spanning a range of problem sizes from 0 to 100. Within this context, the term "problem size" conveys the cardinality of decision variables or elements crucial to the optimization process. Moreover, the designation of an optimization problem as "discrete" implies that its decision variables are constrained to discrete, often integer, values or a finite set of possibilities.

A conspicuous pattern emerges within this graphical representation, where the Random Hill Climbing algorithm and Simulated Annealing consistently yield analogous fitness values across varying problem sizes. Similarly, a parallel observation is discerned in the performance of the MIMIC and Genetic Algorithm pair, who, with unwavering consistency, outperform their counterparts—Simulated Annealing and Random Hill Climbing—across the entire spectrum of problem sizes, mirroring the trends observed in Figure (5).

The phenomenon of increasing fitness values with the escalation of problem size finds its explanation rooted in several pivotal considerations:

Search Space: Larger problem sizes lead to larger search spaces with more possible combinations of decision variables. The greater search space complexity makes it harder to find the best solution, resulting in higher fitness values.

Algorithmic Behavior: Optimization algorithms may perform differently as problem sizes change. Some excel with smaller problems but struggle with larger ones due to increased computational complexity. Others are designed to handle larger problem sizes efficiently, producing better results as the problem size grows.

Algorithmic Aptitude: The effectiveness of an optimization algorithm plays a significant role in the increase of fitness values with larger problem sizes. Some algorithms are better suited for larger problems, converging to improved solutions. Others may have limitations that hinder their performance with larger problem sizes.

Solution Quality: Ultimately, optimization algorithms aim to find high-quality solutions, often reflected in higher fitness values. The increase in fitness values as problem sizes grow indicates an algorithm's ability to handle more complex problems and improve the chances of identifying optimal solutions.

In conclusion, the rise in fitness values with increasing problem sizes, as depicted in Figure (6), results from a complex interplay of factors such as problem complexity, search space dimensions, algorithm behavior, capabilities, and the pursuit of optimal solutions. The precise interpretation of this phenomenon hinges on the unique dynamics at play in the context of the specific optimization problem being studied.

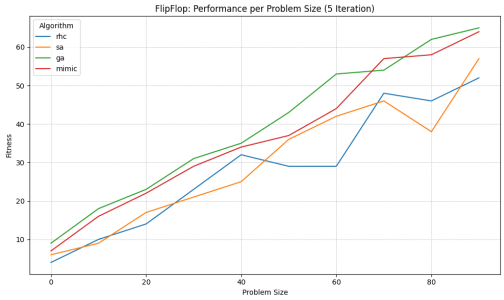

Figure 6: Fitness per Problem Size

The Flip Flop problem is a classic combinatorial optimization problem that serves as a useful benchmark for evaluating optimization algorithms. In this problem, you have a sequence of binary bits, and the goal is to find a configuration that maximizes the number of "flips" or transitions from 0 to 1 or vice versa.

Now, let us consider a hypothetical scenario where this Flip Flop problem space is designed in a way that it contains two absolute maxima. These absolute maxima are the global optimum solutions where the maximum number of flips occurs. However, the problem space is intentionally constructed with a series of descending local maxima, meaning there are many sub-optimal solutions that are easier to reach but do not lead to the absolute maxima.

The presence of these two global maxima, along with descending local maxima, makes the problem space challenging for optimization algorithms. Here's how this setup affects the behavior of the algorithms you mentioned:

Random Hill Climbing with Restarts: Random Hill Climbing is an optimization algorithm that explores the solution space by iteratively making small changes to the current solution and moving towards higher fitness values (more flips in this case). With restarts, it can escape local optima by restarting the search from different initial points. In a problem space with multiple absolute maxima and descending local maxima, this algorithm may converge to one of the global maxima or get stuck in a local maximum. The restarts help it explore different regions of the space and increase the chances of finding a global maximum.

Simulated Annealing: Simulated Annealing is a probabilistic optimization algorithm that incorporates randomness into the search process. It can escape local optima by accepting worse solutions with a certain probability, controlled by a temperature parameter. In this problem space, the annealing schedule may allow it to explore the space thoroughly, potentially leading to one of the global maxima.

MIMIC: MIMIC is a probabilistic optimization algorithm that employs probabilistic models to guide the search. It can capture dependencies between variables and explore promising regions efficiently. In a problem space with two global maxima, MIMIC can learn these dependencies and converge toward the global maxima over time.

Genetic Algorithm: Genetic Algorithms use a population-based approach with genetic operators like selection, crossover, and mutation to explore the search space. They are known

for their ability to handle multimodal problems with multiple optima. In this problem space, genetic algorithms can maintain diversity in the population, allowing them to explore various regions of the space and converge to one of the global maxima.

The similarity in fitness per wall clock time among these algorithms on this problem is likely due to the presence of multiple global maxima and their inherent ability to explore diverse regions of the solution space. While each algorithm may take different paths and require various levels of tuning, they can adapt to the problem's structure and eventually converge to one of the global maxima or a nearby local maximum.

In Figure (7), something interesting happens. The fitness per wall clock time behaves very similarly across all the optimization algorithms we are looking at. This challenges what we saw before when we compared fitness curves based on iterations and evaluation counts. It turns out that the number of evaluations and iterations doesn't perfectly match the time it takes.

Moreover, wall clock time is a crucial measure for assessing how efficient an algorithm is. So, if all the algorithms show similar fitness curves concerning wall clock time, it suggests they're pretty much equally effective.

There is another thing to notice here. Althoughit is not super obvious because of the way the graph is scaled, we can see that the fitness values level off later in terms of wall clock time. This suggests that all the algorithms might be converging to the same best solution or even the very best possible solution in the problem.

In simpler terms, Figure (7) tells us that these different optimization methods, while they work in their unique ways, all end up with similar results when we consider how much time it takes them. They're converging towards the same solution, which is a sign that they might all be equally good at finding the best answer.

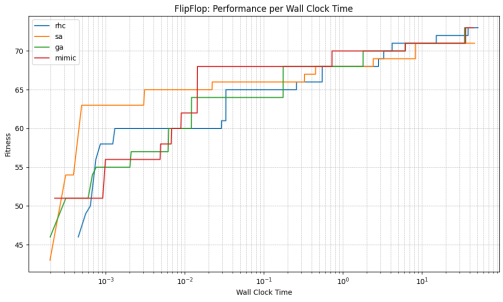

Figure 7: Fitness per Wall Clock Time

3.2 Four Peaks Problem

In our assessment of optimization algorithms on the Four Peaks problem (Figure 8), we scrutinized their performance over 50 iterations. Four diverse algorithms were studied, with Random Hill Climbing and Simulated Annealing displaying similar behavior at around 50 evaluations, albeit with Random Hill Climbing slightly ahead.

In contrast, MIMIC showcased a wider evaluation range (50 to 600 per iteration), maintaining a consistent fitness curve slightly below Random Hill Climbing. The Genetic Algorithm, with the highest evaluation count (400 to 4000), emerged as the top performer.

Upon close examination of the graphical representation, it becomes evident that the majority of the algorithms appear to follow a shared global trend in relation to the number of evaluations. This suggests the tantalizing possibility of extrapolating this trend to approximate the behavior of each algorithm.

It is important to note that the highest fitness value attained in the Four Peaks problem in Figure (8) is 25. However, the Genetic Algorithm does not appear to be close to converging to this optimal fitness level, suggesting that additional investigation into increasing the number of iterations may be warranted. Despite this, what stands out is the Genetic Algorithm's continuous refinement of its solution, progressively moving towards a fitness value of 70 as the number of evaluations increases. This ongoing improvement underscores its potential to approach a more favorable solution for the Four Peaks problem, but it is yet to reach the optimal fitness value of 25. Further iterations and evaluations hold promise for the algorithm to draw nearer to the desired fitness level.

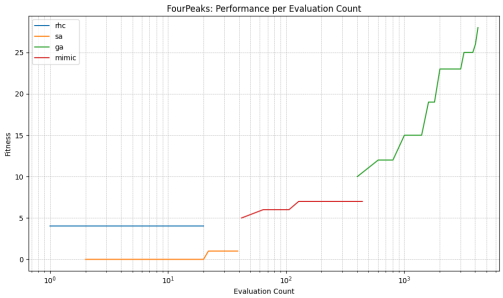

Figure 8: Fitness per Evaluation

In the examination of the Four Peaks optimization problem, as depicted in Figure (9), a salient observation can be made regarding the performance of different optimization algorithms. The genetic algorithm swiftly identifies a maxima, whereas random hill climbing, simulated annealing, and MIMIC exhibit difficulties in their initial iterations, demonstrating only marginal improvements in fitness. One could postulate that the inherent properties of random hill climbing and simulated annealing, which tend to discover new points in close proximity to their current position, impose a limitation on their exploratory capabilities. In contrast, the genetic algorithm boasts a more expansive reach, enabling it to rapidly traverse various regions within the problem domain. With respect to MIMIC, a deeper dive into its relative performance is warranted.

The Four Peaks problem is renowned for its deceptive fitness landscape, where multiple peaks pose challenges for optimization algorithms that rely heavily on local search strategies. Let's dissect the behavior of each algorithm in this context.

Random Hill Climbing and Simulated Annealing: Both algorithms primarily depend on the exploration of neighboring solutions. In essence, they move to new solutions that are within a defined distance of their current position. In terrains like the Four Peaks, this can lead them to becoming entrapped in local maxima, restricting their ability to discover more dominant, global peaks. Simulated annealing attempts to mitigate this issue by probabilistically accepting worse solutions, with the intent of escaping local optima. However, its performance hinges on parameters like cooling schedule, which, if not finely tuned, can still render it susceptible to the challenges posed by the Four Peaks landscape.

Genetic Algorithm: Genetic algorithms utilize a population-based approach, facilitating the exploration of a broader range of solutions concurrently. Through operations

like crossover and mutation, they can effectively "jump" between different regions of the search space. This quality proves advantageous in landscapes like the Four Peaks, as it augments the algorithm's ability to bypass local optima and steer towards more rewarding regions.

MIMIC: MIMIC employs a probabilistic model to guide its search, essentially learning the structure of the problem space as it progresses. In the early iterations, the model may not yet have grasped the intricate nuances of the Four Peaks landscape. This initial learning curve could explain MIMIC's tepid performance in the beginning. Additionally, the efficiency of MIMIC is often contingent on the number of samples and iterations granted. If prematurely halted, it might not be afforded adequate opportunity to refine its model and subsequently pinpoint the more elusive maxima.

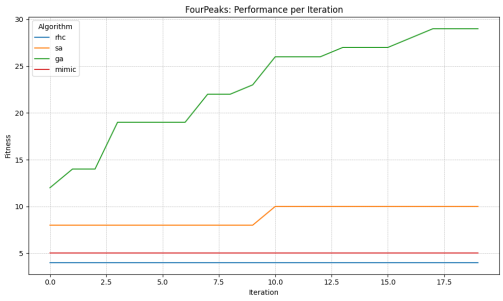

Figure 9: Fitness per Iteration

In an empirical investigation of the Four Peaks optimization problem found in Figure (11), when the problem dimensionality is systematically augmented from a scale of 10 to 100, a noteworthy observation emerges regarding the performance dynamics of various optimization algorithms. Specifically, the fitness values attained by random hill climbing, simulated annealing, and MIMIC remain remarkably invariant, showing no discernible fluctuations across the expanded problem scales. Contrarily, the genetic algorithm not only demonstrates the capability to procure superior fitness values but also exhibits a certain degree of stochastic variability.

Such consistent and invariant outcomes from the first trio of algorithms, across all instantiated problem scales, strongly imply their inability to locate an optimal or even near-optimal maxima within the given iteration constraints. One might postulate that these algorithms, due to their inherent search mechanisms, may require a substantially augmented iteration count to proficiently navigate and uncover the prominent "peaks" of the problem's fitness landscape. Conversely, the genetic algorithm, with its population-based approach, might be benefiting from a broader and more diversified search spectrum, granting it a probabilistic advantage in locating elevated fitness regions, albeit with the inherent randomness that accompanies its search processes.

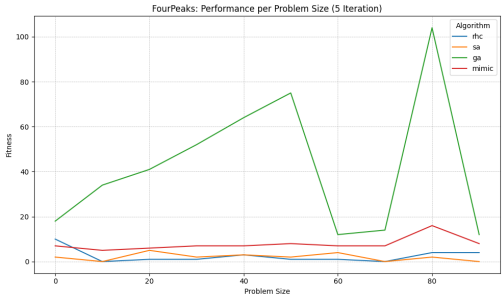

Figure 10: Fitness per Problem Size

From a meticulous analysis of Figures (8) and (10), juxtaposed with the data presented in Figure (7), an intriguing pattern regarding the performance of various optimization algorithms emerges. When evaluated with respect to wall clock time, these algorithms exhibit analogous behaviors across different problem instances, including the nuanced Four Peaks problem.

Given the consistent performance when measured by wall clock time, it's plausible that earlier observed disparities arise from a non-linear correlation between wall clock time and iteration or evaluation counts for these algorithms.

If we were to prioritize wall clock time as the paramount metric of efficiency, this analysis suggests that each algorithm, irrespective of its inherent operational dynamics, could be deemed satisfactory for the problem at hand. In essence, the choice of algorithm would be contingent upon other factors, as their temporal efficiencies appear to converge within the context of the data sets presented.

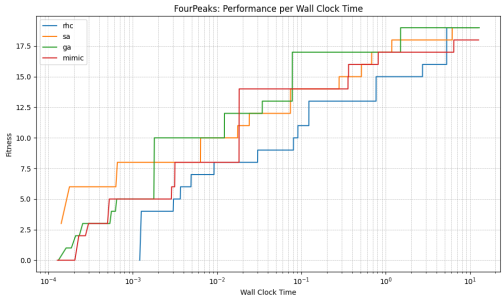

Figure 11: Fitness per Wall Clock Times

3.3 One Max Problem

In Figure (12), we observe the optimization algorithms endeavoring to ascertain the maxima of the One Max problem—a canonical optimization challenge. The genetic algorithm, intriguingly, does not exhibit signs of convergence by the terminal evaluation, suggesting that its exploration might still be underway.

On the other hand, simulated annealing, random hill climbing, and MIMIC show faster improvements in fitness with each evaluation. This quick progress highlights these algorithms' effectiveness for this problem. It also points out that the number of evaluations or iterations does not always match up with the actual time taken. This means some algorithms can improve quickly in terms of results, but this does not always mean they are faster in real time.

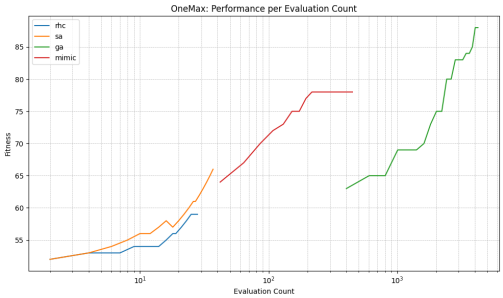

Figure 12: Fitness per Evaluation

In the illustrative Figure (13), it is noteworthy to observe that MIMIC, in its preliminary iterations, converges quickly to a local maxima, subsequently manifesting a period of stagnation. In stark contrast, the genetic algorithm exhibits a more sustained and progressive enhancement in its performance metrics, ultimately eclipsing MIMIC. Such a phenomenon might imply that MIMIC became ensnared within a particularly tough local maxima, inhibiting its escape within the stipulated iteration constraints. The uncertainty remains whether, with extended iterations, MIMIC would eventually transcend this local optimum.

Furthermore, a juxtaposition of the performance trajectories of random hill climbing and simulated annealing reveals a pronounced parallelism, thereby accentuating the inherent algorithmic similarities between the two. This correlation provides a lens through which we can discern the nuances of convergence, elucidating that while some algorithms might swiftly gravitate toward local optima, others might exhibit a more gradual yet relentless pursuit of potentially superior maxima.

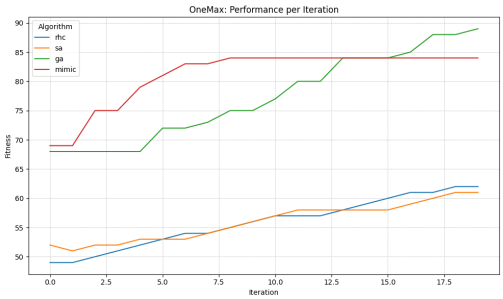

Figure 13: Fitness per Iteration

In Figure (14), one discerns a linear escalation in the performance of the algorithms, commensurate with the augmentation of the problem size. This observation resonates with the trends previously shown in Figure (6), pertaining to the Flip Flop problem. This could suggest that the One Max problem landscape is similar to the Flip Flop fitness landscape.

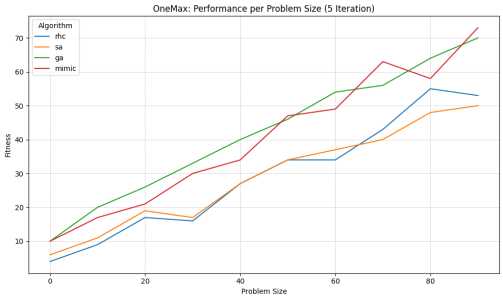

Figure 14: Fitness per Problem Size

Figures (7) and (15) offer empirical evidence of the logarithmic trajectory of the algorithmic performances when evaluated against wall clock time. In the context of convergence theory, such a logarithmic pattern exemplifies characteristics of asymptotic behavior. While a logarithmic function does not attain convergence in a mathematical sense, it exhibits quasi-convergence within finite temporal intervals, as its rate of growth diminishes considerably with increasing input magnitude.

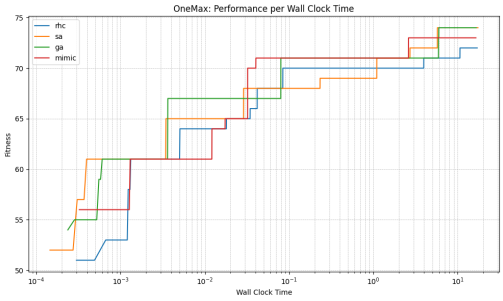

Figure 15: Fitness per Wall Clock Time

4 Datasets Introduction

4.1 Auction Dataset

The intricacies of auction verification have had significant financial implications, resulting in billions of dollars in revenue losses. Ensuring the prevention of undesirable auction outcomes is paramount to the formulation of auction policies. The dataset I've chosen for exploration is sourced from the esteemed "UC Irvine Machine Learning Repository" (DOI: 10.24432/C52K6N). It can be accessed directly at (https://archive.ics.uci.edu/dataset/713/). This dataset provides insight into auction outcomes by examining bid histories and determining the desirability of the auction's conclusion. Any duplicate entries in this dataset were diligently removed, culminating in a concise dataset of 2,043 rows. While the issue could be tackled as either a classification or regression problem, I've chosen to focus solely on its classification aspects for this project.

4.2 Dropout Dataset

Forecasting student dropouts is pivotal for instituting measures to bolster retention rates. By proactively identifying potential dropouts, we can intervene, thereby augmenting their chances of academic success. This dataset encapsulates various

facets of a student at the time of enrollment, encompassing past academic achievements, projected academic trajectory, demographic details, and socio-economic indicators. Each entry in the dataset portrays an individual student's trajectory, delineated as "graduated", "dropout", or "enrolled". For the scope of this project, these outcomes have been numerically translated to 0, 1, and 2, respectively.

5 Data Pre-processing

The provided code outlines the pre-processing pipeline for two datasets: "auction" and "dropout", sourced from the UC Irvine machine learning repository. Here's a breakdown of the preprocessing steps:

5.1 Loading Data

The load_datasets function reads in the two datasets from their respective locations. The "dropout" dataset specifically is delimited by semicolons. For the "dropout" dataset, the target labels ("Graduate", "Dropout", and "Enrolled") are converted into numeric labels (0, 1, and 2 respectively).

5.2 Categorizing Fields

The categorize_fields function segregates columns of each dataset into different categories: categorical, continuous, targets, and discrete. For the "auction" dataset, there aren't predefined categorical or continuous columns, so they're determined dynamically by excluding target columns. For the "dropout" dataset, predefined columns are set for categorical and continuous types.

5.3 One-Hot Encoding

In the onehot_encode_categorical function, categorical variables in both datasets are one-hot encoded. This means they're converted into a binary matrix representation which is suitable for machine learning algorithms.

5.4 Clustering Continuous Features

The cluster_continuous_features function attempts to cluster continuous features and replaces their original values with the cluster label. For each continuous column, an optimal number of clusters k is determined using the KMeans algorithm. This number is chosen by examining the second derivative of the inertia to find an "elbow". Continuous columns' values are then replaced by their respective cluster labels, effectively transforming them into discrete values.

5.5 Train-Test Split and Normalization

In the train_test_split_normalize_features function: Each dataset is split into training, validation, and test sets using a 64-16-20% split. Features labeled as "discrete" (which now include the formerly continuous columns) are normalized to have a mean of 0 and a standard deviation of 1 using the StandardScaler from the scikit-learn library.

5.6 Final Pre-processing

The preprocess_datasets function prepares the datasets for training and evaluation by: Dropping target columns from the feature matrices (X values) and storing target columns separately (y values). Ensuring that all dataframes have float data types.

6 Neural Network Training

In Figure (16) of the neural network trained with a genetic algorithm optimizer, when evaluating a neural network, or any classifier for that matter, both accuracy and the AUC provide insights into the model's performance. However, they capture different facets of this performance. The discrepancy between a high accuracy and a comparatively lower AUC could be indicative of various characteristics of your model and its decision boundary. Here's what it might signify:

Imbalanced Dataset: High accuracy with a lower AUC often indicates class imbalance, where the model is biased towards the majority class.

Decision Boundary Complexity: A high accuracy can mean the model generally classifies well, but a lower AUC suggests struggles at differentiation margins.

Optimization Differences: The genetic algorithm optimizer might align more with optimizing accuracy rather than class separation, affecting the AUC.

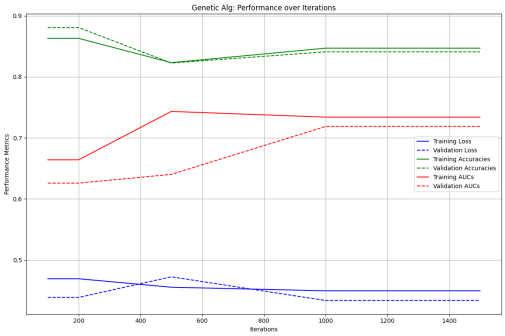

Figure 16: Performance per Iteration

In Figure (17), the neural network utilizing a gradient descent optimizer, specifically via backpropagation, exhibits a congruence between accuracy and AUC, a disparity notably absent when compared to the model optimized with the genetic algorithm. This observation suggests that the gradient descent mechanism, through iterative refinement, potentially engenders a more harmonized balance between overall classification correctness (accuracy) and the model's intrinsic capability to differentiate between classes across thresholds (AUC). The inherent nature of gradient descent, which leverages the error gradient to adjust weights, may foster a decision boundary that is more robust and generalizable across different decision thresholds. Conversely, the genetic algorithm optimizer, which employs a population-based search, might prioritize certain metrics in its fitness function, leading to an optimized yet potentially overfitted decision boundary that accentuates the accuracy-AUC divergence.

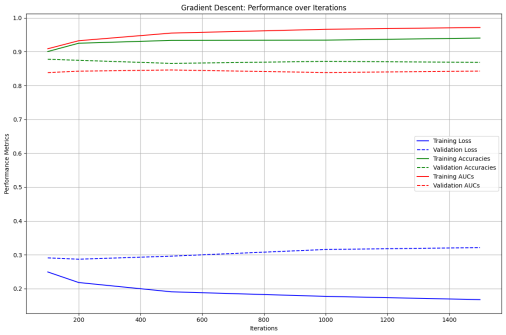

Figure 17: Performance per Iteration

In Figures (18) and (19), both utilizing random hill climbing and simulated annealing optimization strategies respectively, a pronounced divergence is observed between the accuracy and AUC metrics, with the models distinctly favoring accuracy. This underscores a potential scenario where the classifiers exhibit a high degree of correctness in overall predictions, yet struggle in discerning between classes across varied decision thresholds.

The nature of both random hill climbing and simulated annealing might provide insight into this phenomenon:

Algorithmic Search Mechanism: Both algorithms explore the solution space using local search techniques. While they can effectively find solutions that maximize accuracy, they may inadvertently prioritize regions of the search space that don't necessarily optimize the trade-off between true positive rate and false positive rate, which the AUC encapsulates.

Local Optima: These algorithms have an inherent tendency to converge towards local optima. In the context of classification metrics, they might latch onto a local solution that offers high accuracy but doesn't necessarily ensure a balanced differentiation between classes across thresholds, thus affecting the AUC.

Exploration vs. Exploitation: The balance between exploring new regions of the solution space and exploiting known good solutions is pivotal. In the quest for higher accuracy, these algorithms might overly exploit certain configurations that boost accuracy while inadvertently compromising the AUC.

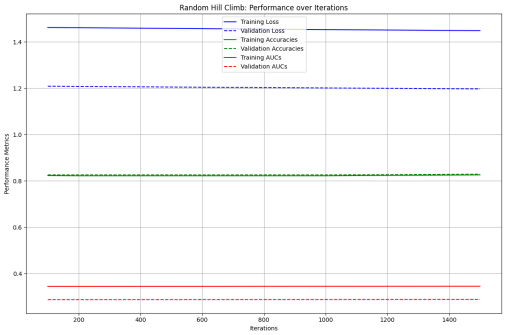

Figure 18: Performance per Iteration

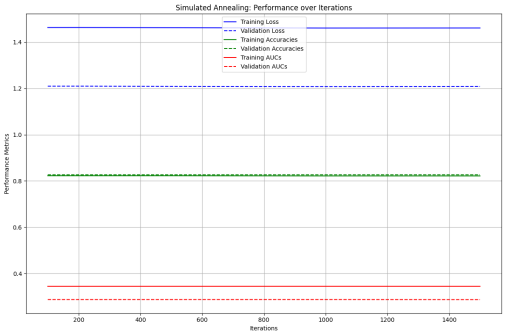

Figure 19: Performance per Iteration

Figures (20) and (21) show the performance of the genetic algorithm and gradient descent optimized neural networks in the earlier iterations (0-100). The purpose of these graphs where to uncover the early performance of these optimization algorithms in the model training. We can that these graphs are nearly identical to their predecessors (Figures 16 and 17).

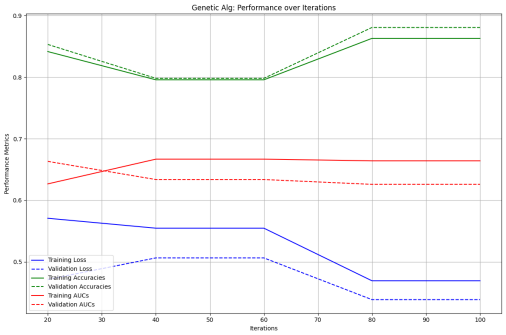

Figure 20: Performance per Iteration

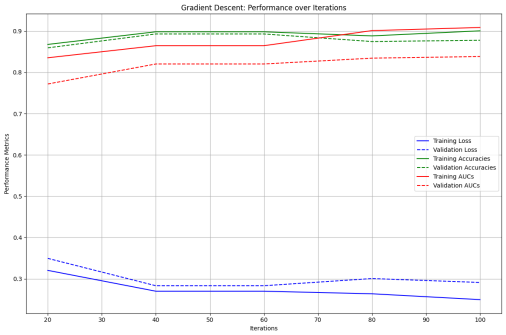

Figure 21: Performance per Iteration

7 Conclusion

Random hill climbing and backpropagation adjust neural network weights differently. Backpropagation uses the chain rule for systematic gradient computation, while random hill climbing introduces random perturbations to weights, evaluating their impact on the loss function. Simulated annealing employs a "temperature" parameter to determine the magnitude of weight changes and probabilistically accept sub-optimal solutions. Genetic algorithms, inspired by natural evolution, use "chromosomes" to define solutions, selecting high-performers for breeding. In contrast, MIMIC constructs a statistical model

representing parameter dependencies in chromosomes to generate new solutions. In a test, the Genetic Algorithm outperformed other algorithms in optimizing a specific problem.

Comparing Random Hill Climbing and Simulated Annealing, the former is primarily exploitative, while the latter balances exploration and exploitation with a temperature parameter. This distinction might explain why Random Hill Climbing initially outperforms Simulated Annealing. Both algorithms, however, appear to plateau at sub-optimal fitness levels, suggesting they might either need more iterations or face inherent constraints in discovering the best fitness function. Genetic Algorithms show rapid convergence, contrasting with MIMIC's slower progress.

Figure (6) provides a detailed exploration of optimization problem dynamics across varying problem sizes. Here, Random Hill Climbing and Simulated Annealing consistently produce similar fitness results, while MIMIC and the Genetic Algorithm outperform them across all problem sizes. As problem sizes increase, fitness values rise due to factors such as larger search spaces, algorithmic behavior, aptitude, and the pursuit of quality solutions. This increase in fitness with larger problem sizes demonstrates an algorithm's capability to handle more intricate problems and better identify optimal solutions.

The Flip Flop problem is a classic optimization challenge wherein a sequence of binary bits aims to maximize the number of transitions between 0 and 1. This problem becomes intricate when designed with two global maxima and numerous local maxima, offering optimal and sub-optimal solutions respectively. Different optimization algorithms, such as Random Hill Climbing with Restarts, Simulated Annealing, MIMIC, and Genetic Algorithms, approach this problem differently, each employing unique mechanisms to navigate the solution space. However, Figure (7) reveals that, concerning real-world time (wall clock time), these algorithms all perform similarly. This consistent performance indicates that while each method may vary in iterations or evaluations, in terms of time efficiency, they are comparably effective.

Delving into the Four Peaks problem, the landscape is deceiving due to multiple peaks, challenging optimization algorithms that largely rely on localized search strategies. Both Random Hill Climbing and Simulated Annealing primarily explore neighboring solutions, which can limit their efficiency in such landscapes. Genetic Algorithms, with their population-based methods, can effectively bypass local optima, while MIMIC, guided by a probabilistic model, might need more iterations for optimal performance. When problem dimensionality increases, as depicted in various figures, the genetic algorithm stands out in its performance, hinting that its broad search spectrum may provide an edge. Conversely, the consistent performance of other algorithms, irrespective of the problem's scale, suggests their search mechanisms might need more iterations to pinpoint optimal solutions efficiently.

In the study of optimization algorithms, wall clock time suggests that any given algorithm can be deemed efficient, depending on various factors. Notably, while some algorithms like genetic algorithms exhibit slower convergence, others like simulated annealing and MIMIC show quicker progress. This distinction, however, doesn't always correlate with actual real-time performance. Figures 12 through 14 provide insights into the algorithms' performance in different contexts, illustrating behaviors like MIMIC's initial rapid convergence and subsequent stagnation and the genetic algorithm's consistent performance improvement. The parallels between random hill climbing and simulated annealing accentuate their inherent similarities. Additionally, as per Figures 16 to 19, the trained neural networks present varying correlations between accuracy and AUC, indicating possible influences from imbalanced datasets, decision boundary complexities, and specific optimization differences.

The datasets chosen for analysis cover topics such as auction verification and student dropouts, sourced from the "UC Irvine Machine Learning Repository." A comprehensive pre-processing pipeline is employed for these datasets, including data loading, field categorization, one-hot encoding, feature clustering, train-test splits, normalization, and final preparations. The neural networks trained on these datasets exhibit varied performance metrics depending on the optimization techniques used. Particularly, a genetic algorithm optimizer tends to optimize for accuracy potentially at the expense of AUC, whereas gradient descent optimization balances both metrics better. This discrepancy points to the intrinsic characteristics and mechanisms of these optimization strategies.

Figures 20 and 21 focus on the early iterations of neural networks optimized with genetic algorithms and gradient descent. These figures reveal striking resemblances to their counterparts, Figures 16 and 17. Divergences between accuracy and AUC metrics are observed in models using random hill climbing and simulated annealing optimizations. This divergence can be attributed to the nature of these algorithms, which may prioritize local search techniques, converge towards local optima, and possibly compromise AUC in favor of accuracy due to the balance of exploration versus exploitation. Such nuances underscore the complexities and intricacies of optimization in machine learning and the importance of understanding algorithmic behaviors in model performance.