Randomized Search Techniques for Real-World Optimization Problems

Tyler Dickson
Georgia Institute of Technology
tdickson30@gatech.edu

***Abstract*—Optimization plays a critical role in both real-world decision-making and machine learning. Despite the widespread use of traditional optimization methods, there remains a gap in understanding how local random search algorithms can be effectively applied to discrete and continuous optimization problems. This paper addresses that gap by exploring three algorithms: Randomized Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithm (GA) across two problem domains: optimizing study time (a Knapsack Problem) and maximizing travel efficiency (a Traveling Salesman Problem). Here, we show that SA outperformed RHC in escaping local optima in both Knapsack and Traveling Salesman problem spaces, but GA excelled in the more complex state space of the TSP due to its genetic crossover and mutation capabilities. Additionally, these algorithms were used to optimize neural network weights for predicting donation spikes following school shootings. RHC struggled with local minima, SA effectively escaped local optima while maintaining precision-recall balance, and GA exhibited erratic behavior due to its unsuitability for continuous weight spaces. These results highlight the importance of algorithm selection based on problem characteristics, contributing valuable insights to optimization in both discrete and machine learning tasks.**

I. Introduction

Optimization is a crucial concept across various domains, from business to everyday decision-making. In the corporate world, managers strive to reduce costs, maximize time, increase profits, and improve efficiency. To achieve these goals, many organizations employ specialists and engineers who analyze business processes and utilize machine learning algorithms to determine the most effective strategies. However, optimization is not limited to businesses; it also plays a significant role in daily life. People frequently, whether consciously or subconsciously, make decisions to optimize various aspects of their routines. However, unlike organizations, individuals often lack access to sophisticated optimization techniques.

In this project, we bridge this gap by examining two everyday scenarios through the lens of optimization. We apply three local randomized search algorithms—Randomized Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithms (GA)—to two distinct optimization problem domains that mirror real-life events. We will refer to these domains as **Problem Space 1** and **Problem Space 2**. Next, we extend our exploration by using these algorithms to optimize the weights of a neural network, building on a problem space from previous work.

II. Problem Space Establishment

A. *Problem Space 1: Job Interview Preparation*

Problem Space 1 In this scenario, a Georgia Tech graduate is preparing for a software engineering interview with limited time (**50** hours) to review **40** Java-related topics. Each topic is assigned an importance value (**1** to **10**) and a time requirement (hours), modeled as a Knapsack Problem. RHC, SA, and GA are applied to optimize the study plan, balancing topic importance with available time.

B. *Problem Space 2: Exploring Job Site Location*

Problem Space 2 After the interview, the student receives a job offer in Denver, Colorado, and decides to explore **10** key attractions before making a relocation decision. With limited time, this scenario is framed as a Traveling Salesman Problem. Distances between attractions are computed via Google Maps, and RHC, SA, and GA are used to find the optimal route that maximizes the number of visited attractions in the shortest time Figure 9 provides a map visualization, while Table VII provides the lookup matrix.

III. Problem Space 1: Job Interview Preparation

A. *Hypothesis for Problem Space 1*

Simulated Annealing (SA) is expected to outperform both Randomized Hill Climbing (RHC) and Genetic Algorithm (GA) implementations for this problem. RHC's greedy nature limits its effectiveness, as it only makes moves that improve its current position, leading it to get stuck in local optima. In contrast, SA's ability to probabilistically accept worse solutions enables it to escape local optima and avoid plateauing at suboptimal fitness values.

B. *Data Preprocessing*

To enhance readability, iteration values, function evaluations (FEvals), and convergence points in all results tables for both **Problem Space 1** and **2** represent actual iterations. However, the X-axes in all figures depict iteration snapshots rather than the total number of iterations. These snapshots reflect checkpoints captured during algorithm execution but repeat across multiple runs, providing a simplified view for comparison. While they show trends, they do not represent the complete number of iterations performed.

Convergence occurs when the algorithm reaches a state where further iterations offer minimal improvements. This is

pinpointed by identifying the iteration where the highest fitness value is achieved and ensuring subsequent iterations show negligible change.

C. *Problem Space 1: Randomized Hill Climbing*

Our analysis of Problem Space 1 aims to highlight the advantages of Simulated Annealing (SA). Simulated annealing is often considered an extension of the many variants of stochastic hill climbing, which consists of a sequence of transitions across a state space while improving an objective function at each iteration until reaching a global optimum. [4] To fully appreciate these benefits of SA, we first examine the problem space using Randomized Hill Climbing (RHC) to understand its limitations.

The objective of RHC is to find the global maximum, which corresponds to the optimal solution. In this case, the global maximum is defined by our objective function, which attempts to maximize the fitness score by selecting study topics with the highest importance while adhering to the 50-hour time constraint. RHC follows a greedy approach, searching through the state space while evaluating one solution (or state) at a time, and only accepting new solutions if they yield a higher fitness value. The algorithm terminates when no further improvements can be made to the objective function. Upon termination, RHC randomly restarts from a new point in the state space, continuing this process for a set number of iterations.

D. *Randomized Hill Climbing: Analysis*

TABLE I
RHC: RESULTS BEFORE TUNING

Metric Runs	**Value**
Iterations	26794
Best Fitness	86
FEvals	14446
Wall Clock Time	27.49
Restarts	25
Convergence Iteration	1414

TABLE II
RHC: RESULTS AFTER TUNING

Metric Runs	**Value**
Iterations	6324
Fitness	94
FEvals	6790
Wall Clock Time	3.87
Restarts	38
Convergence Iteration	4872

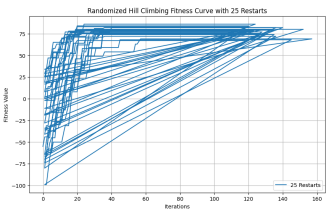

Fig. 1. RHC: Before Tuning

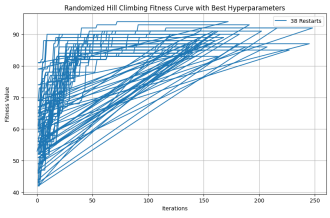

Fig. 2. RHC: After Tuning

The RHC algorithm was run in **Problem Space 1** before and after hyperparameter tuning using Python's Optuna library, focusing on the number of restarts. Figures 1 and 2 depict the Fitness Function Curves before and after tuning. In the case of our RHC algorithms, the hyperparameters are simply the number of restarts performed.

Before tuning, trials were conducted with **25**, **75**, and **100** restarts, with Figure 1 showing the best result from the **25**-restart trial. Figures 3 and 4 illustrate the outcomes for **75** and **100** restarts, showing varying degrees of optimization. Without prior knowledge of the ideal number of restarts or iterations needed to reach an optimal fitness value, it becomes necessary to test different restart levels. This process inherently leads to higher wall clock times and more complex computations. Post-tuning, we identify an optimal restart value of **38**, leading to improved efficiency.

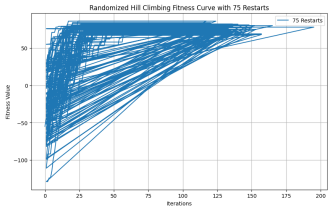

Fig. 3. RHC: Before Hyperparameter Tuning with 75 Restarts

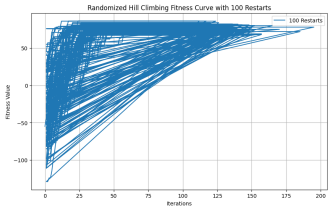

Fig. 4. RHC: Before Hyperparameter Tuning With 100 Restarts

Tuning the RHC algorithm yielded notable improvements. The best fitness value increased from **86** to **94**, suggesting that the tuned RHC was able to escape a local maximum and further optimize the solution. Function Evaluations (FEvals) rose significantly, reflecting more extensive state space exploration. Additionally, wall clock time decreased from **27.11** seconds to **3.11** seconds, suggesting that prior to tuning, the algorithm spent significant time stuck at a local maximum.

Figures 1 and 2show the convergence points, with Figure 1 displaying convergence at fitness **86** by iteration **24**, and Figure 2 showing convergence at fitness **94** by iteration **72**.

E. *Problem Space 1: Simulated Annealing*

Simulated Annealing (SA) offers an alternative to Randomized Hill Climbing (RHC) by implementing several key differences. As discussed in Section III-B, we observed how RHC took a greedy approach to maximizing its fitness function. Because RHC refuses to take a downhill move, its chances of getting stuck at a local maximum is always present. It relies heavily on the algorithm restarting to escape. Conversely, SA allows the algorithm to randomly accept moves instead of the best move. If the move improves the fitness value, its accepted. Moves which do not improve the fitness value are accepted within a certain probability. We can relate the probability of a worse move being accepted to the concept of temperature decreasing in metallurgy work [2]. In early iterations, the algorithm is more likely to accept worse moves. Over time, probability of accepting a worse move decreases similar to temperature cooling (annealing). By accepting a worse move, SA is able to avoid getting stuck in local maxima.

F. *Simulated Annealing: Analysis*

Using Python's mlrose-ky library, we applied the Simulated Annealing (SA) algorithm to **Problem Space 1** across three

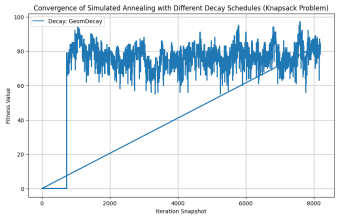

Fig. 5. SA: Geometric Decay Fitness Curve

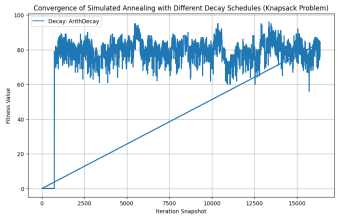

Fig. 6. SA: Arithmetic Decay Convergence

decay schedules: Geometric Decay, Arithmetic Decay, and Exponential Decay. Since temperature control significantly impacts SA's ability to optimize the fitness function, fine-tuning these decay schedules is critical to the algorithm's performance. The decay schedules determine how quickly or slowly the temperature decreases over iterations, influencing SA's exploration and convergence capabilities.

In the following sections, we examine the results for each decay schedule, with their respective equations defined as follows:

- T_0 is the initial temperature (at time $t = 0$);
- r is the rate of geometric decay; and
- T_{min} is the minimum temperature value. [2]

*a) **Simulated Annealing: Geometric Decay**:* First. Geometric decay was ran over **Problem Space 1**, denoted by the following equation:

$$T(t) = \max(T_0 \cdot r^t, T_{\text{min}}) \tag{1}$$

Best hyperparameters (Temperatures):

$$[66.43, 40.58, 77.21, 51.34, 133.18] \tag{2}$$

Hyperparameter search range was set between **5** and **150** to explore a wide range of temperatures. The initial temperature was set to **80**, with a decay rate of **0.99**, and a minimum temperature of **9**. Geometric decay reduces temperature by multiplying it by a constant decay factor. The algorithm allows for rapid exploration early on and gradually converges as the temperature approaches its minimum value.

The geometric decay schedule showed strong performance (Table IV), balancing exploration and exploitation well, with a fitness score of **97**. The rapid cooling in earlier iterations allowed the algorithm to explore a diverse solution space before focusing on convergence. The performance stabilized toward the end (Figure 5), indicating effective convergence.

TABLE III
SA: GEOMETRIC DECAY

Metric	Value
Iterations	40965
Fitness	97
FEvals	10780
Wall Clock Time	5.01
Convergence Iteration	7588

*b) **Simuated Annealing: Arithmetic Decay**:* Next, arithmetic decay was ran over Problem Space 1, denoted by the following equation:

$$T(t) = \max(T_0 - r^t, T_{\text{min}}) \tag{3}$$

Best hyperparameters (Temperatures):

$$[28.16, 21.45, 20.45, 8.22, 8.40] \tag{4}$$

For this schedule, hyperparameters were tuned by setting temperature search parameters between **3** and **50** to reflect arithmetic decay's sensitivity to high temperatures within a relatively small state space. Initial temperature was set to **10**, with a decay rate of **0.07**, and minimum temperature of **5**. Unlike geometric decay, arithmetic decay reduces temperature by subtracting a constant value from it, resulting in a steady, linear decrease in temperature.

While arithmetic decay can offer more predictability in how the temperature declines, it can also slow down convergence if not tuned properly. This linear reduction requires carefully balancing the initial temperature and decay rate to avoid premature convergence. Table IV displays the results of the implementation. Arithmetic decay showed relatively stable performance with a fitness score of **96** and reached convergence after **13352** iterations. The wall clock time was noticeably higher than geometric decay, at **19.12** seconds due to the slower cooling process (Figure 6).

TABLE IV
SA: GEOMETRIC DECAY

Metric	Value
Iterations	81925
Fitness	96
FEvals	19363
Wall Clock Time	19.12
Convergence Iteration	13352

*c) **Simulated Annealing: Exponential Decay**:* Finally, Exponential Decay was ran over **Problem Space 1**, shown in the following equation:

$$T(t) = \max(T_0 e^{-} rt, T_{\text{min}}) \tag{5}$$

Best hyperparameters (Temperatures):

$$[42.33, 31.53, 45.01, 37.42, 27.44] \tag{6}$$

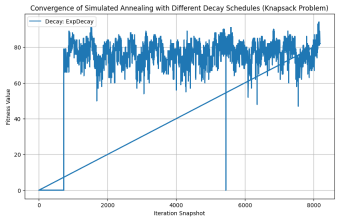

Fig. 7. Enter Caption

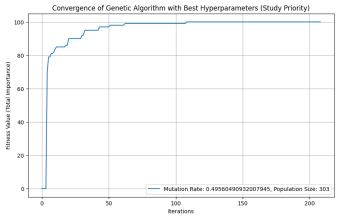

Fig. 8. GA Convergence Snapshot

Our exponential decay schedule hyperparameter search criteria confined our temperature search to values between **8** and **50** to account for this schedule's sensitivity to initial temperature without allowing it to drop below **9**. Initial temperature was set to **80**, with an exponential decay rate of **0.96** and a minimum temperature of **9**. Exponential decay rapidly reduces the temperature in early iterations, similar to geometric decay but at an accelerating rate.

Results of the exponential decay algorithm are displayed in Table V. The implementation balanced rapid exploration with steady convergence, achieving a fitness value of **94** with **10816** evaluations. While it converged relatively quickly in terms of iterations (**8178**), it fell just short of the Geometric and Arithmetic Decay methods in terms of fitness score. However, it was highly efficient with a wall clock time of **5.01** seconds.

TABLE V
SA: GEOMETRIC DECAY

Metric	Value
Iterations	40965
Fitness	94
FEvals	10816
Wall Clock Time	5.01
Convergence Iteration	8178

G. *Simulated Annealing vs. Randomized Hill Climbing*

SA consistently performed better than RHC across all implementations. RHC struggled to escape local optima due to its limited ability to accept worse solutions. However, SA is able to escape local optima by accepting less favorable moves early in the search process. This allows it to explore more of the solution space and avoid getting stuck in suboptimal solutions. The decay schedule plays a critical role in determining how much exploration versus exploitation occurs, with each decay method offering different strengths.

Across all three decay schedules, geometric decay provided the best overall fitness score, achieving convergence on a fitness value of **97** in **7588** iterations. The rapid initial exploration followed by slow convergence appeared to strike the right balance between exploration and exploitation for this particular problem space. [5]

H. *Problem Space 1: Genetic Algorithm*

The Genetic Algorithm (GA), inspired by evolution and natural selection, explores **Problem Space 1**'s state space by selecting the fittest individuals for reproduction. Crossover between fit individuals generates offspring, while mutations introduce diversity to prevent premature convergence. [6]

*a) **GA Results and Convergence**:* In **Problem Space 1**, the GA demonstrated an unexpected advantage over both SA and RHC algorithms (Table 15), achieving a fitness value of **97** in just **108** iterations, surpassing both algorithms in terms of final fitness value and convergence speed.

The best run, with a population size of **303** and a mutation rate of **0.4956**, demonstrated rapid convergence and balance between exploration and exploitation. Figure 8 illustrates the swift rise in fitness values, stabilizing near the optimal solution in fewer iterations than expected.

TABLE VI
SA: GEOMETRIC DECAY

Metric	Value
Iterations	63549
Fitness	100
FEvals	63549
Wall Clock Time	1.60
Convergence Iteration	108

*b) **Comparative Analysis: Why Was Our Hypothesis Wrong?**:* Despite my initial hypothesis that SA would outperform both RHC and GA due to its flexibility in escaping local optima, the results show that GA's population-based approach allowed it to achieve a better solution faster.

SA excels at escaping local optima through probabilistic exploration, accepting worse solutions to eventually converge on a global optimum. However, while our problem space isn't a traditional multiobjective optimization problem, certain aspects of it reflect the inherent complexity found in such problems. Specifically, balancing the importance of study topics with the constraint on study time introduces conflicting objectives, akin to the trade-offs seen in multiobjective optimization. [4]

GA's population-based exploration maintained a diverse set of solutions across generations, effectively balancing exploration of new areas while exploiting promising solutions. This parallel exploration allowed GA to converge more efficiently, outperforming SA. While SA excels in escaping local optima, the complexity of this problem space required more iterations for SA to fully explore and balance the implicit objectives.

Fig. 9. Visualization Map: Attractions to Visit in Denver

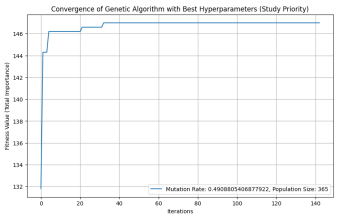

Fig. 10. Genetic Algorithm Convergence Curve

Fig. 11. GA: Results

Metric	Value
Mutation Rate	0.49
Iterations	143
Fitness	147
FEvals	52342
Wall Clock Time	2.07
Convergence Iteration	42

IV. Problem Space 2: Exploring Job Site Location

A. *Hypothesis for Problem Space 2*

Genetic Algorithm (GA) is expected to outperform both Randomized Hill Climbing (RHC) and Simulated Annealing (SA) implementations in **Problem Space 2**. The GA is well suited to perform implementations of the Traveling Salesman Problem due to the way it utilizes crossover operations to preserve optimal path segments from parent solutions. This will allow the GA to optimize its fitness value while exploring more of the state space.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	–	24.2	23.8	40.4	19.7	21.7	21.9	36.3	23.7	22.2
2	24.2	–	2.2	19.0	3.1	2.1	1.6	14.9	2.5	1.6
3	23.8	2.2	–	17.8	8.6	4.8	1.9	13.7	0.7	1.4
4	40.4	19.0	17.8	–	25.7	20.1	22.5	7.0	17.6	19.2
5	19.7	3.1	8.6	25.7	–	2.0	2.9	21.6	6.7	3.2
6	21.7	2.1	4.8	20.1	2.0	–	3.1	15.9	4.5	3.7
7	21.9	1.6	1.9	22.5	2.9	3.1	–	15.1	2.7	0.4
8	36.3	14.9	13.7	7.0	21.6	15.9	15.1	–	13.9	15.5
9	23.7	2.5	0.7	17.6	6.7	4.5	2.7	13.9	–	2.0
10	22.2	1.6	1.4	19.2	3.2	3.7	0.4	15.5	2.0	–

TABLE VII
DISTANCE MATRIX FOR DENVER'S MAIN ATTRACTIONS

B. *Problem Space 2: Genetic Algorithm*

The implementation of the GA over the state space of **Problem Space 2** allows for a unique interpretation of results. We can visualize the problem as analogous to evolution, where solutions evolve over time to become more optimized. Let's start with a high-level overview of how the GA interacts with **Problem Space 2**. [6]

First, selection occurs by choosing individuals from the population to become parents. Initial parents may be selected based on the path with the highest fitness, which corresponds to the shortest travel distance. This step ensures that the best-performing routes have a higher chance of propagating their characteristics to the next generation.

Next, crossover takes place. In evolutionary terms, crossover is akin to mating, where sections of the parents' itineraries are exchanged to create child solutions. In **Problem Space 2**, our GA swaps portions of the parents' itineraries to form new offspring, ensuring that the sequence of attractions remains valid—meaning no attraction is visited more than once. This allows the GA to combine high-quality portions of different itineraries, improving chances of generating better routes.

After crossover, mutation introduces random changes to individual itineraries. This might involve swapping two attractions or reversing the order of a subset of attractions. Mutation ensures diversity in the population, preventing the GA from getting stuck in local optima by exploring alternative routes that might lead to better overall solutions.

Finally, the GA generates a new population and completes the iteration. At each iteration, the fitness value of each individual (the inverse of the total travel distance) is recalculated and tracked to monitor progress. This fitness value helps the algorithm decide which solutions should continue into the next generation. Over successive iterations, the GA gradually converges on an optimal solution, balancing the need to exploit known good solutions and explore new possibilities. [6]

C. *Genetic Algorithm: Results Analysis*

After tuning hyperparameters, the GA was ran over **Problem Space 2**. The algorithm successfully converged on the optimal fitness value of **147** within **42** iterations. This demonstrates GA's ability to quickly find an ideal solution for the problem space. The fitness value corresponds to the shortest travel distance, or path, among the 10 attractions in Denver. The fact that the GA converged in a relatively small number of iterations indicates that the solution space is well explored with the chosen hyperparameters.

Table 15 highlights the results of the implementation. The Genetic Algorithm ran for **143** iterations and converged at a fitness value of **147** by iteration **42**, which indicates that the algorithm efficiently found the optimal solution early on. The tuned mutation rate of **0.49** played a key role in this performance. A mutation rate of **0.49** indicates that almost half of the portion of the population underwent random changes during each iteration. This helped maintain diversity and prevented the algorithm from getting stuck in local optima, allowing it to explore new paths effectively.

Despite the early convergence, the total number of function evaluations (FEvals) was **52,342**, which is relatively high. This reflects the large population size and the frequent mutation operations that the algorithm performed at each iteration. While this contributed to the successful optimization of the problem, the high FEval count indicates that it explored a wide solution space with frequent updates to each individual's fitness.

This combination of a tuned mutation rate and a large number of evaluations ensured that the GA effectively explored the solution space, quickly homing in on the optimal path for the student's travel itinerary.

Figure 10 shows the fitness value increasing rapidly in the initial iterations before plateauing around iteration **42**, where the algorithm reaches the optimal solution at 147. The sharp increase in fitness at the beginning represents our GA's ability to quickly identify good solutions, followed by a slower refinement phase where smaller improvements are made until convergence.

The bit string in the equation environment represents the optimal order of visiting attractions. Our GA produced the following optimal path:

$$\text{Best State (bit string)} : [5, 9, 7, 4, 3, 6, 8, 1, 0, 2] \quad (7)$$

Referencing the lookup matrix in figure 9, the student can successfully map the bit string equation to the identified optimal order of attractions to visit.

D. *Randomized Hill Climbing: Results Analysis*

Despite it's simplicity, RHC performed very well. Its implementation converged at iteration **73** with a fitness value of **147**, which is an identical fitness value to GA, but required **6048** total iterations and **6310** function evaluations (FEvals). In contrast, the GA required **143** iterations to complete and had a much higher number of **52342** FEvals. Despite exploring more of the state space, the GA converged faster at iteration **42**. Additionally, GA completed in **2.07 seconds** of wall clock time, while RHC took **3.62 seconds**.

These metrics highlight that while RHC was more computationally efficient in terms of fewer function evaluations, GA performed better overall in terms of wall clock time and the speed of convergence. The more complex operations in GA, such as crossover and mutation, led to more FEvals, but its population-based search allowed it to find the optimal solution faster in real time.

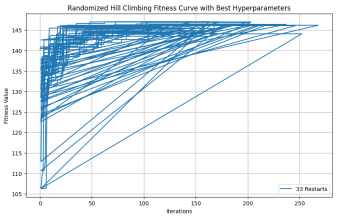

Fig. 12. Randomized Hill Climb Fitness Curve

Fig. 13. RHC: Results

Metric	Value
Iterations	6048
Fitness	147
FEvals	6310
Wall Clock Time	3.62
Convergence Iteration	73

Figure 12 displays the convergence at the upmost left position in the graph at **147**, with **33** restarts. This implementation allowed it to escape local optima and find the global solution. RHC's use of restarts were crucial for RHC to maintain competitive performance, compensating for the algorithm's more limited ability to explore beyond local regions. While RHC's search was highly efficient in function evaluations, the multiple restarts and iterative search took longer to compute.

The bit string in the equation environment represents the optimal order of visiting attractions. Our RHC algorithm produced the following optimal path:

$$\text{Best State (bit string)} : [0, 1, 8, 6, 3, 4, 7, 9, 5, 2] \quad (8)$$

Referencing the lookup matrix in figure 9, the student can successfully map the bit string equation to the identified optimal order of attractions to visit. Interestingly, the final solutions provided by RHC and GA differed in their specific routes despite reaching the same fitness value. This demonstrates that **Problem Space 2** has multiple optimal routes, with different algorithms exploring the space in distinct ways to find valid solutions.

E. *Simulated Annealing: Results Analysis*

SA was implemented using a geometric decay schedule on **Problem Space 2**. Results of the implementation are displayed in Figure 14. The algorithm performed very efficiently, converging at the optimal fitness value of **147** after **3338** iterations. Despite the high number of iterations, the algorithm's total runtime was remarkably low, with a wall clock time of just **0.08** seconds, demonstrating its computational efficiency.

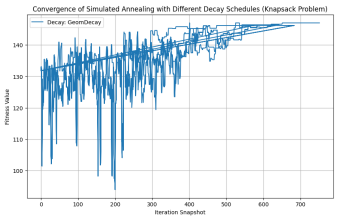

Fig. 14. Simulated Annealing Fitness Curve

Fig. 15. SA: Results

Metric	Value
Iterations	3338
Fitness	147
FEvals	1120
Wall Clock Time	0.08
Convergence Iteration	2987

The SA algorithm benefited from the geometric decay schedule, which gradually reduced the probability of accepting worse solutions as the iterations progressed. Initially, SA explored a wide range of solutions, allowing it to escape local optima where the fitness value fluctuates significantly in the initial phases. However, as the algorithm approached its optimal solution, the fluctuations decreased, and the fitness value steadily increased, eventually converging at **147**.

The SA's ability to converge at iteration **2987** indicates that the decay schedule was appropriately tuned for the problem space, allowing sufficient exploration without needlessly wandering around the state space. Additionally, the algorithm was able to converge quickly with a low number of FFEvals **(1120)**. This may suggest that the state space may not offer the level of complexity needed to force SA into implementing a large number of computations.

*1) **Hypothesis Analysis**:* When compared to the GA and RHC, SA required more iterations to converge, but it was significantly faster in real time. The wall clock time of **0.08** seconds is notably lower than both GA and RHC, making SA the most time-efficient algorithm for this problem space. However, the GA was able to significantly explore more of the state space by conducting **52342** function evaluations in

only **2.07** seconds, while the SA conducted **1120** FEvals in a shorter time span of **.08** seconds. This is partially consistent with our hypothesis that GA would outperform RHC and SA.

The bit string in the equation environment represents the optimal order of visiting attractions. Our RHC algorithm produced the following optimal path:

$$\text{Best State (bit string)} : [2, 5, 9, 7, 4, 3, 6, 8, 1, 0] \quad (9)$$

Referencing the lookup matrix in figure 9, the student can successfully map the bit string equation to the identified optimal order of attractions to visit. Interestingly, all three algorithms found different paths that maximizes the fitness function. This makes sense, and Problem Space 2 has multiple optimal routes.

V. Finding Optimal Weights In A Neural Network

A. *Weight Tuning: Problem Space Establishment*

We apply Randomized Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithm (GA) to tune weights in a neural network across a continuous problem space. Adapting these optimization algorithms to a non-discrete domain requires specific considerations. We evaluate the impact on training loss, accuracy, and overall performance, offering insights into both optimization effectiveness and the model's generalization on unseen data.

The neural network, previously built for analyzing Federal Election Commission (FEC) data, aims to predict whether a school shooting leads to a spike in donations to pro- or anti-gun committees.

In the following sections, we compare the training loss, accuracy, and classification performance for each algorithm, highlighting their respective strengths and weaknesses.

B. *Hypothesis*

SA will outperform RHC and GA implementions in finding optimal weights for the nerual network. While GA may explore a more diverse amount of the state space, additional complexity will be introduced do to its requirement to balance mutation rates and population sizes. This will ultimately cause the GA to exhibit more variance in its results while SA will quickly converge on an optimal configuration.

C. *Randomized Hill Climbing (RHC)*

The RHC algorithm was implemented over **10000** iterations with **10** restarts. Initial weights were generated randomly between **-1** and **1**, and hyperparameters were left untuned to highlight differences compared to SA and GA. The training loss curve showed sharp spikes corresponding to restarts, as the algorithm moved into worse regions before fine-tuning the weights. Accuracy improved gradually, reflecting the model's increasing ability to predict donation spikes. The training loss curve depicted on the left in Figure 16 shows how far off the model's predictions are from actual values—whether a donation spike occurs or not. RHC attempts to minimize this loss by exploring new weight configurations. The sharp spikes in the curve correspond to restarts or moves into worse regions

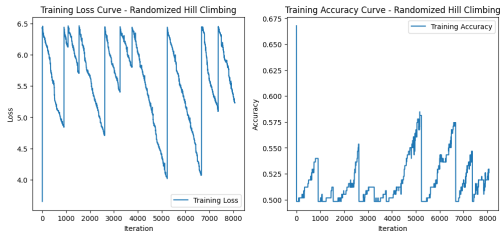

Fig. 16. RHC: Training Loss and Accuracy Curve

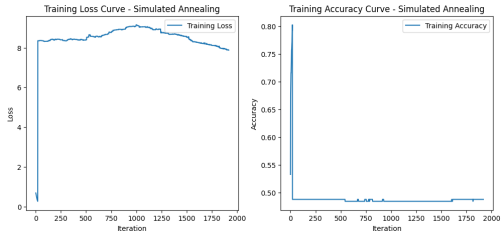

Fig. 17. SA: Training Loss and Accuracy Curve

of the weight space, while the gradual decreases represent fine-tuning as the algorithm refines the weights to reduce the loss.

The Training Accuracy Curve on the right of Figure 16 illustrates how accuracy improves as loss decreases. Jumps in accuracy occur when RHC finds better weight configurations, and the gradual increases reflect incremental improvements as the model becomes better at predicting donation spikes.

Results displayed in Table IX show that RHC achieved a best fitness of **4.02** but struggled with local minima, relying heavily on restarts to escape. While the model performed well with **90%** accuracy, **0.91** precision, and **0.90** recall, the instability in the loss curve suggests that tuning or alternative algorithms like SA or GA could yield more consistent results.

TABLE VIII
RHC: Results post Weight Optimization

Class	Precision	Recall	F1-score	Support
No Donation Spike	0.88	0.95	0.92	40
Donation Spike	.93	0.85	0.89	33
Accuracy			0.92	73
Macro Avg	0.91	0.90	0.90	73
Weighted Avg	0.91	0.90	0.90	73

D. *Simulated Annealing (SA)*

SA was run using a geometric decay schedule with Optuna-tuned temperatures. The algorithm converged with a best fitness of **-8.27** after 2000 iterations. The Training Loss Curve depicted in Figure 17 shows an initial sharp drop followed by a steady increase and gradual decline, indicating that SA initially explored a wide range of weight configurations, but then settled into a more optimal region.

The Training Accuracy Curve highlights the early exploration phase, with accuracy spiking initially but then dropping and stabilizing around **50%**. However, after fine-tuning, the

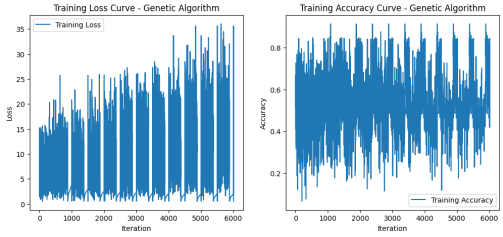

Fig. 18. GA: Training Loss and Accurcy Curve

final classification report shows a test accuracy of **92%**, with strong performance across both classes. In the Donation Spike class, there was a Precision of **0.89** and Recall of **0.97**. In the Donation Spike class, there was a precision score of **0.97** and recall score of **0.85**. This suggests that while SA's broad exploration initially caused some instability, the final solution was effective for tuning the neural network's weights.

TABLE IX
SA: RESULTS POST WEIGHT OPTIMIZATION

Class	Precision	Recall	F1-score	Support
No Donation Spike	0.89	0.97	0.93	40
Donation Spike	0.97	0.85	0.90	33
Accuracy			0.92	73
Macro Avg	0.93	0.91	0.92	73
Weighted Avg	0.92	0.92	0.92	73

*a) **Hypothesis Analysis and Implications**:* The SA implementation demonstrates its ability to explore a wide range of weight configurations, allowing it to escape local minima early in the process. While this exploration led to fluctuations in accuracy, the algorithm ultimately found an effective set of weights, as reflected in the high final accuracy and balanced precision-recall scores. These results are consistent with our hypothesis predictions. SA was more effectively able to find a more optimal weight configuration for predicting donation spikes following a school shooting.

E. Genetic Algorithm (GA)

The GA was implemented with a population size of **200**, a mutation probability of **0.1**, and a maximum of **1000** generations. Table X displays the results of the implementation. While the algorithm achieved a final test accuracy of **92%**, the training process showed significant instability, with fitness values fluctuating widely and training loss ranging from **5** to **35**. This variability suggests that GA struggled to converge due to high-magnitude weight adjustments each generation. Precision and recall for "No Donation Spike" were **0.89** and **0.97**, respectively, while for "Donation Spike" they were **0.97** and **0.85**.

*a) **Hypothesis Analysis and Implications**:* The results suggest that while GA can achieve good accuracy, it does so in a highly erratic fashion, as shown in Figure 18 by the unstable loss and accuracy curves. The broad search space explored by GA, combined with the continuous nature of the weights, likely caused the algorithm to oscillate between good and bad solutions. This highlights a key limitation of GA in optimizing neural network weights, which require more fine-grained adjustments than GA typically offers. Results exhibited by the GA is also consistent with our hypothesis that more specialized methods like Simulated Annealing or Randomized Hill Climbing are more effective in tuning neural network weights.

TABLE X
GA: RESULTS POST WEIGHT OPTIMIZATION

Class	Precision	Recall	F1-score	Support
No Donation Spike	0.89	0.97	0.93	40
Donation Spike	0.97	0.85	0.90	33
Accuracy			0.92	73
Macro Avg	0.93	0.91	0.92	73
Weighted Avg	0.92	0.92	0.92	73

CONCLUSION

The application of RHC, SA, and GA in this study highlights the importance of selecting algorithms based on the problem space. Contrary to our initial hypothesis that SA would excel in Problem Space 1, GA proved more effective due to its ability to explore the state space through crossover and mutation. In **Problem Space 2**, both SA and GA offered valuable insights, with neither clearly outperforming the other. SA's strength in escaping local optima and GA's ability to preserve strong solutions through genetic operations made both algorithms suitable for optimizing complex paths.

In neural network weight tuning, RHC struggled with local minima, while SA demonstrated balanced exploration and convergence. GA, despite achieving reasonable accuracy, exhibited erratic behavior, underscoring its limitations in continuous optimization tasks. These results reinforce the importance of choosing algorithms that align with the specific characteristics of the problem domain.

1) Implications for Future Work: This work provides a foundation for further exploration in optimization across various problem domains by showing how local search algorithms perform differently in discrete and continuous spaces. It encourages future researchers to refine algorithms to better handle complex spaces.

REFERENCES

[1] N. Kapila, "mlrose: Machine Learning, Randomized Optimization, and SEarch," [Online]. Available: https://nkapila6.github.io/mlrose-ky/. [Accessed: 15-Oct-2024].
[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2020, p. 241.
[3] N. Kapila, "mlrose: Machine Learning, Randomized Optimization, and SEarch," [Online]. Available: https://nkapila6.github.io/mlrose-ky/. [Accessed: 15-Oct-2024].
[4] A. M. Ramadhan, M. M. Tawfik, S. A. Mostafa, and A. Al-Hashemi, "An Efficient Swarm Intelligence Algorithm for Optimization: State-of-the-Art, Challenges and Future Directions," *Computational Intelligence and Neuroscience*, vol. 2019, Art. no. 8134674, 2019. doi: https://doi.org/10.1155/2019/8134674.
[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2020, pp. 326-327.
[6] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997, pp. 249-255.