Beyond Backpropagation: Genetic Algorithms and Heuristic Search for Neural Networks and Discrete Optimization

Nader Liddawi
nliddawi3@gatech.edu

***Abstract*—Randomized optimization algorithms, including Randomized Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithms (GA), provide stochastic alternatives to gradient-based methods, leveraging heuristic search to navigate complex optimization landscapes. This report provides an extensive investigation into these methods in both discrete and continuous problem spaces. First, we benchmark them on two discrete tasks: FourPeaks at sizes 30, 60, and 90, and the Traveling Salesman Problem (TSP) at sizes 10, 50, and 100. We then adopt them to replace standard gradient descent (GD) when training a neural network for a marketing campaign classification problem. We hypothesized that GA, with a constrained iteration budget, could surpass RHC and SA in test F1 for NN training and potentially approach GD in runtime. We systematically explore hyperparameter bounds for each algorithm and rationalize the wide variety of observed behaviors using machine-learning theory, focusing on iteration budgets, restarts vs. populations, and acceptance schedules. Through rigorous experimentation and statistical analysis, we demonstrate GA's superiority in discrete, modular tasks but highlight its limitations in high-dimensional continuous spaces, where GD remains unmatched in F1 and speed. These findings underscore how problem structure dictates optimization efficiency: GA excels in modular discrete tasks due to recombination benefits but fails to scale effectively in high-dimensional continuous weight spaces, where gradient-based methods dominate.**

---

1. Introduction

1.1 Background

Randomized optimization (RO) techniques are pivotal in addressing NP-hard or non-convex optimization challenges, where traditional methods falter due to rugged landscapes or lack of gradient information [2]. RHC, SA, and GA each sample candidate solutions in ways that eschew the need for gradient or derivative information. RHC focuses on local improvements with the possibility of restarts; SA includes a temperature-driven acceptance of worse solutions; GA invests in a population of solutions that evolve through mutation and crossover. Despite their shared randomness, their performance can differ substantially based on problem characteristics—particularly problem size, ruggedness, and dimensionality [1].

1.2 Hypothesis

We hypothesize that: "A well-tuned Genetic Algorithm (GA), by leveraging population-based search for exploration, should both outperform Randomized Hill Climbing (RHC) and Simulated Annealing (SA) in final test F1 score and converge in less time than a conventional gradient-descent baseline, thanks to a capped number of iterations and attempts." This hypothesis draws on the population-based advantage of GA for exploration, plus the assumption that we can limit the iteration budget so that GA might converge more swiftly than a baseline that could run for many epochs [7]. The code output may or may not validate this claim, so we evaluate it directly in Section 3.2.

1.3 Paper Structure

Sections 2.1 and 2.2 briefly describe the discrete tasks (FourPeaks and TSP) and the imbalanced marketing dataset used for the neural network experiments, including a rationale for the hyperparameter search bounds. In Section 3, we dive deeply into the results in the tables, discussing the performance differences among RHC, SA, and GA, along with theoretical justifications. Section 4 summarizes the main insights to conclude the paper and recommend further study.

---

2. Problem Setups and Algorithmic Mechanics

2.1 Discrete Optimization Tasks

1. FourPeaks

o *Definition:* The Four Peaks problem is a combinatorial optimization problem where the objective is to maximize the number of consecutive 1s at both the beginning and end of a binary bitstring [2]. A threshold parameter, t_pct (typically set to 0.15 in mlrose), determines the minimum length required for a sequence of 1s to qualify as a "peak." The challenge lies in balancing the trade-off between maximizing the number of 1s at both ends while avoiding a configuration that prematurely maximizes one peak at the expense of the other.

o *Sizes:* 30, 60, 90. Table I below includes the top hyperparameter combos for each of RHC, SA, and GA.

o *Why these Hyperparameters:*

RHC: `num_restarts` up to 20 boosts exploration to escape local optima, `max_iterations` up to 500 to balance convergence, `max_attempts` up to 50.

SA: `schedule_index` ∈ {exp, geom, arith}, `initial_temperature` ∈ [1,20], `temperature_decay` ∈ [0.01,0.5]. We limit temperature_decay's upper bound to 0.5 to avoid extremely slow cooling.

GA: `population_size` ∈ [30,200], `mutation_prob` ∈ [0.01,0.3], `crossover_prob` ∈ [0.4,0.9], `max_iterations` up to 200. This range fosters enough population diversity while containing overhead.

2. Traveling Salesman Problem (TSP)

o *Definition:* TSP is a combinatorial optimization problem where a set of randomly sampled (x, y) coordinates represents cities [4], [5]. The goal is to determine the shortest possible route that visits each city exactly once and returns to the starting point. To convert the standard minimization objective into a maximization problem, the negative of the total travel distance is used as the objective function [4], [5]. This transformation allows TSP to be framed within optimization frameworks that inherently maximize fitness values, such as RHC, SA and GA.

o *Sizes:* 10, 50, 100. Below, Table I logs the best combos.

o *Why these Hyperparameters:* The same bounding logic as above. Because TSP can be quite large, we still cap `max_iterations` ∼ 500 for RHC/SA and ∼200 for GA to avoid severe runtime blowups. Restarts (RHC) and large populations (GA) can each benefit or harm performance, depending on cost vs. solution quality.

2.2 Marketing Dataset for Neural Networks

1. *Data Processing:*

We analyze an imbalanced marketing dataset (85% no, 15% yes response), dropping missing values and applying one-hot encoding for categorical features. To prevent dominance by large-scale features, we standardize numeric variables. SMOTEENN addresses class imbalance by oversampling the minority class while filtering noisy examples. The data is split into 70% training, 15% validation, and 15% testing. To ensure model generalizability, we use 5-fold stratified cross-validation, optimizing hyperparameters via Bayesian Optimization. F1-score guides selection, balancing precision and recall for a fair assessment of performance on imbalanced data.

2. *Baseline & Parameter Bounds:*

The baseline is a gradient-descent NN with a single hidden layer of 100 neurons, a sigmoid activation, a regularization `alpha`=2.4638e^{-4}, and `learning_rate`=5.829e^{-3}. For RHC/SA/GA, we keep the same hidden size and activation but disregard alpha and learning rate, since random optimization does not rely on gradient-based updates. Hyperparameters were optimized using Bayesian optimization, with bounds chosen to balance exploration and computational feasibility (e.g., max iterations 500 for RHC/SA, population size 200 for GA).

2.3 Central Hypothesis on Neural Network Tuning

As a reminder, we hypothesize that GA, if its iteration budget is well-chosen, can surpass RHC/SA in final test F1 and potentially run in less time than the gradient-descent baseline. However, as we will show in Section 3.2, while GA does outperform RHC/SA in test F1 score, it remains significantly slower than the gradient-descent baseline, partially contradicting our initial hypothesis.

2.4 Reproducibility Note

We use a fixed random seed for consistent algorithmic behavior, but runtime remains hardware-dependent. Execution times vary due to CPU architecture, memory speed, parallelism, and background processes. While relative trends between algorithms should stay consistent, absolute timing may fluctuate across environments.

---

3. Results and Analysis

TABLE I
FourPeaks Results (Best Fitness & Time)

Size	Algorithm	Best Fitness	Time (s)	Key Hyperparams
30	RHC	38	∼0.013–0.017	Restarts≈12–14, Iter=233–234
30	SA	54	∼0.005–0.007	Exp/Geom, T≈1–17, Decay≈0.09–0.49
30	GA	54	∼0.10–0.20	pop≈80–108, mut≈0.1–0.2, cross≈0.48–0.70
60	RHC	20	∼0.01	Restarts≈8, Iter=445
60	SA	23	∼0.01	Exp/Geom, T≈19, Decay≈0.2–0.3
60	GA	83	∼0.31	pop=161, mut=0.278, cross=0.853
90	RHC	9–10	∼0.004–0.009	Restarts≈15+, Iter=480+
90	SA	10	∼0.006–0.008	Geom/Exp combos
90	GA	107	∼0.16–0.41	pop=108, mut=0.116, cross=0.706

3.1 Discrete Optimization: FourPeaks and TSP

1) 3.1.1 FourPeaks Analysis:

Case A: Size=30

RHC: Best fitness=38; top parameters included RHC_NumRestarts ∼12 to 14, with max_iterations up to 233–234. The time was ∼0.013–0.0175 seconds, extremely short.

o **Analysis**: Despite efficient exploration via restarts (fitness=38), RHC's exploitative focus cannot match SA/GA's performance (54) as it cannot merge partial solutions across peaks in a 30-bit string, despite its minimal runtime overhead.

- **SA**: Achieves fitness=54 consistently with different initial_temperature (1.3911 to 17.1468) and temperature_decay (0.0864 to 0.4979).

o **Analysis**: FourPeaks(30) presents a well-defined search landscape, allowing SA's probabilistic acceptance of suboptimal moves to facilitate rapid boundary crossing. The low sensitivity to decay suggests that a broad range of cooling schedules can effectively guide SA to the global optimum in this relatively compact search space. When temperature allows a few "bad moves," both peaks are easily found, matching GA's best fitness (54). However, at sizes 60 and 90, SA struggles against GA, highlighting its

difficulty in escaping local maxima in larger search spaces.

• **GA**: Reaches the global best fitness=54. For instance, GA_PopulationSize=80, mutation_prob=0.1353, crossover_prob=0.4917, with 129 max_iterations yields 54 in ∼0.1085 seconds.

o **Analysis**: GA thrives by combining partial solutions—e.g., a half string of 1s from one parent, half from another. Even with moderate iteration counts, GA reliably reaches fitness 54 by recombining beneficial partial solutions, enabling structured search through crossover. Although its per-iteration overhead is higher than RHC/SA, the total runtime remains sub-second, demonstrating GA's efficiency in modular landscapes with strong building blocks. From a convergence perspective, this method accelerates progress toward the global optimum by leveraging diversity in candidate bitstrings.

Case B: Size=60

• **RHC**: Best fitness=20. Attempts to push restarts from 3 to 8 improved outcomes, but the overhead remained fairly small (0.002–0.01s).

o **Analysis**: A 60-bit string amplifies the challenge of forming dual peaks, so RHC's local moves often maximize just one end. Though 8 restarts and 445 iterations help, the algorithm's single-solution scope (vs. population-based recombination) stalls at 20 fitness, far from the global optimum.

• **SA**: Achieves up to 23 with schedule "exp," initial_temperature=19.45, decay=0.3072, in 0.012s. Another combo with "geom" also yields 23.

o **Analysis**: SA's ∼+3 advantage over RHC comes from accepting worse neighbors early, escaping local traps, but it falls short of GA's higher results. Code logs show ∼0.009–0.012s for top combos, still modest overall.

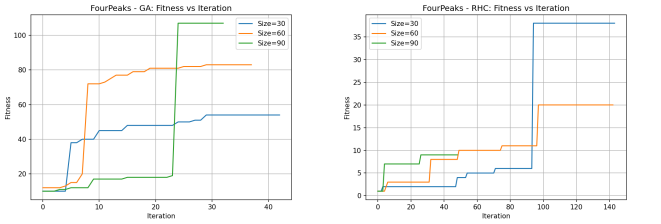
• **GA**: Best fitness=83 with population_size=161, mutation_prob=0.2783, crossover_prob=0.8532, max_iterations=72. This is a drastic improvement over RHC's 20 or SA's 23.

o **Analysis**: The 60-bit string's modular structure favors GA, as crossover efficiently combines high-quality substrings from parents with leading and trailing 1s. Unlike RHC and SA, which rely on local perturbations, GA accelerates convergence by preserving partial solutions. Despite RHC's faster runtime (0.01s vs. GA's 0.3176s), GA's fitness (83 vs. RHC's 20) highlights the superiority of population-based search over single-solution heuristics.

Case C: Size=90

• **RHC**: Sits at fitness=9 or 10 even with restarts up to 19.

o **Analysis**: The large 90-bit space drastically reduces the chance that local climbs converge on the double peak.



(a) GA Fitness Progression in FourPeaks (b) RHC Fitness Progression in FourPeaks

Fig. 1. Comparison of GA vs. RHC in FourPeaks. GA effectively combines partial solutions through crossover, leading to higher fitness, while RHC stagnates due to local search limitations.

The frequent local maxima hamper progress, even with ∼480–500 max_iterations. The 0.004–0.009s runtime is quick, but yields a very poor solution.

• **SA**: Reaches fitness=10, quite similar to RHC.

o **Analysis**: While SA can cross local boundaries, the bitstring is so large that forming both peaks is still improbable. The code suggests that SA stabilizes around 10 across multiple combos, indicating local maxima remain a severe barrier.

• **GA**: Achieves a striking 107 at best with pop_size=108, mutation_prob=0.1162, crossover_prob=0.7059, max_iters=94.

o **Analysis**: GA's 10× fitness advantage (107 vs. 10) demonstrates crossover's ability to merge partial solutions (leading/trailing 1s) that single-solution methods cannot combine. Despite longer runtime (0.16-0.41s), population-based exploration prevents premature convergence and effectively consolidates complementary solution components.

Key Takeaways for FourPeaks

• Hyperparameter Sensitivity: For RHC, restarts are crucial. For SA, initial temperature has moderate impact, but the acceptance of random bad moves generally suffices. For GA, the interplay of population size, mutation probability, and crossover probability is pivotal—too small a population yields lower coverage, too high a mutation can degrade building blocks.

• Time Differences: Algorithm runtime varies significantly by problem type. RHC processes small-scale TSP problems rapidly (0.01-0.05s) but falters on FourPeaks; SA delivers consistent moderate performance; GA demands greater computational resources (0.16-0.41s for FourPeaks-90) but achieves superior results when recombination benefits exist—illustrating the fundamental tradeoff between exploration capability and computational efficiency.

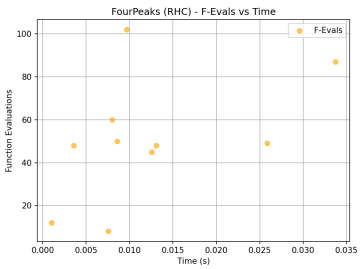


Fig. 2. Function Evaluations vs. Time for RHC in FourPeaks. RHC is computationally efficient but fails to achieve optimal fitness compared to SA and GA.

TABLE II
TSP Results (Best Fitness & Time)

Size	Algorithm	Best Fitness	Time (s)	Key Hyperparams
10	RHC	∼550	<0.1	Restarts=14, Iter=480+
10	SA	∼550	<0.1	Exp schedule, T=1–2
10	GA	∼550	<0.5	pop=41–58, cross=0.58–0.79, mut=0.18–0.23
50	RHC	3447–3450	0.2–0.4	Restarts=11–16, Iter=476–487
50	SA	3414–3415	∼0.05–0.06	Exp schedule, T≈1–2
50	GA	3446–3447	∼1.5–2.4	pop=51, cross=0.67, mut=0.278
100	RHC	∼6930	0.6–1.3	Restarts=15–20, Iter=473–482
100	SA	∼6750–6760	∼0.06	Arith/Exp combos, T=1–4
100	GA	∼6790	2.5–3.5	pop=58–67, cross=0.59–0.64, mut=0.18–0.23

2) 3.1.2 TSP Analysis:

Case A: TSP=10

- All three algorithms (RHC, SA, GA) frequently converge to ∼549.999.

o **Analysis**: TSP=10 is quite small, so local restarts or short population evolution can find a route that yields ∼550 fitness. The code suggests identical or near-identical solutions from multiple hyperparameter combos. The rationale is that TSP=10 has a small combinatorial space (10! permutations), so almost any method easily stumbles on a near-optimal route.

Case B: TSP=50

- RHC obtains ∼3447–3450, slightly ahead of GA's best near ∼3446.5.

o **Analysis**: This result is interesting because we might assume GA would out-explore RHC for TSP. However, with 50 cities, local restarts can effectively refine partial tours. RHC's restarts may rapidly refine competitive tours, whereas GA's reliance on crossover in a permutation space limits its gains. Unlike in modular problems, where recombination constructs better solutions, in TSP, crossover often disrupts well-formed subtours, leading to only marginal improvements despite higher computational cost.

- SA also lands ∼3406–3415, trailing RHC but close to GA.

o Time: The new code logs show RHC times around 0.2–0.45s, while GA can exceed 1.5–2.4s if the population size is large. That extra overhead is not yielding significantly better final fitness for TSP=50.

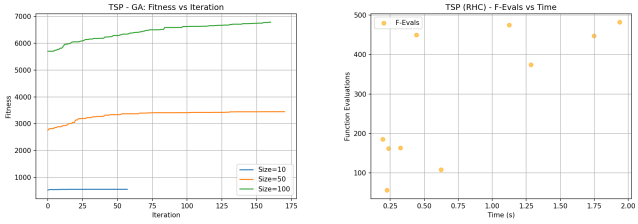
Case C: TSP=100

- **RHC**: Best fitness=6929.97 with num_restarts=20, max_iters=482, attempts=42, in ∼1.3784s. This outperforms GA's best of 6787.86 in ∼3.17s.

o **Analysis**: TSP=100 is big enough that GA's population-based approach may require a far larger iteration or population size to find synergy in partial tours. Meanwhile, RHC's repeated local climbs can unexpectedly converge on a better route.

- **SA**: ∼6750.91.
- **GA**: ∼6787.86, still below RHC.

o **Analysis**: RHC's superior TSP-100 performance (6929 vs GA's 6787) reverses the FourPeaks hierarchy, revealing how problem structure dictates algorithmic efficacy. TSP's interdependent structure makes crossover disruptive to well-formed subtours, while RHC's restart mechanism (n=20) efficiently exploits local improvements, converging more effectively despite 2.3× faster runtime [3]–[5]. GA thrives in problems where modular partial solutions can be recombined into better global solutions. However, in TSP, where the optimal tour is not a simple sum of subroutes, crossover often disrupts strong partial solutions rather than improving them. Since optimal TSP tours rely on incremental refinements rather than merging of partial routes, RHC's local search adapts more effectively, whereas GA's crossover frequently disrupts well-formed sub-tours. Meanwhile, RHC's multiple restarts allow it to exploit local refinements effectively. If the city arrangement favors incremental improvements, RHC can accumulate high-quality solutions, gradually converging to near-optimal routes. Additionally, since TSP solutions are highly interdependent, local improvements often translate to meaningful global improvements, making randomized local search a viable alternative to population-based heuristics like GA in this context.



(a) GA Performance in TSP. GA struggles due to the disruptive effects of crossover on partially optimized tours.

(b) Function Evaluations vs. Time for RHC in TSP. RHC outperforms GA in large TSP instances by refining a single solution.

Fig. 3. Comparison of GA vs. RHC in TSP. The GA's crossover can be disruptive on partially optimized tours, whereas RHC's local refinement can lead to better performance on larger instances.

Key Takeaways for TSP

1. The TSP results reveal that RHC can sometimes outperform GA, especially in TSP(100), where GA's crossover tends to break already optimized subtours. Unlike in structured problems like FourPeaks, where crossover merges useful building blocks, in TSP, random recombination often disrupts well-formed tours, limiting GA's effectiveness. GA's inability to leverage crossover effectively in TSP suggests that its success is problem-dependent: modular problems (FourPeaks) benefit from recombination, whereas TSP's interdependencies make mutation and local search more effective than crossover [3].

2. SA is consistent but not always best.

3. Hyperparameter Bounds: For TSP, large iteration or attempt values can significantly inflate runtime but yield moderate improvements.

4. Time vs. Fitness Patterns: GA invests the most time on large TSP while not guaranteeing the best route.

Comparison of Algorithmic Performance

The observed differences in algorithmic performance can be attributed to fundamental trade-offs in exploration versus exploitation. RHC is inherently greedy, making it effective for problems where rapid local improvements lead to strong solutions but suboptimal in deceptive landscapes with many local minima. SA mitigates this by introducing probabilistic escapes, yet its performance heavily depends on the cooling schedule; overly aggressive temperature decay results in premature convergence, whereas slow decay may lead to excessive exploration. GA, in contrast, leverage population-based search, preserving diversity and allowing for recombination of beneficial traits. However, GA is not universally superior; its effectiveness is contingent on the problem structure.

In modular tasks like FourPeaks, crossover merges complementary solutions, driving efficient global search. By contrast, TSP's interdependent routes or NN weights gain little from random recombinations, which tend to break well-adapted structures instead of enhancing them. These trade-offs underscore the necessity of selecting optimization methods aligned with the landscape properties of the given problem.

3.2 Neural Network Weight Tuning on Marketing Data

TABLE III
NEURAL NETWORK RESULTS (MARKETING DATASET)

Algorithm	CV_F1	Test_F1	Time (s)	Key Observations
Baseline (GD)	0.8327	0.8258	3.7002	Fast + best F1; uses gradient
RHC-NN	0.4623	0.4144	137.984	Stuck in poor local minima
SA-NN	0.4278	0.3604	22.8442	Accepts worse solutions, but final stuckness
GA-NN	0.6936	0.6997	61.6596	Population mixing helps; still behind GD

Central Hypothesis: GA will achieve a higher test F1 score than RHC or SA while also surpassing the baseline GD in speed. We will now determine if the hypothesis is true based on the Table III results.

Baseline

- CV_F1=0.8327, Test_F1=0.8258, runtime=3.7002s.

o **Analysis**: The baseline's moderate complexity (one hidden layer, alpha $\sim 2.46e^{-4}$, learning_rate $\sim 5.83e^{-3}$) quickly converges to a local optimum with a robust final test F1=0.8258. The difference between CV and test is minimal, ∼0.83 vs. 0.82, showing consistent generalization. And we only used one hidden layer because preliminary experiments (not shown) suggested that two or more hidden layers did not substantially increase F1 score performance but did increase runtime.

Neural network optimization navigates high-dimensional, non-convex loss landscapes where gradient information guides along the steepest descent path to update the weights and biases, efficiently minimizing loss [10]. Random optimization algorithms lack this strategic navigation, making their stochastic perturbations increasingly inefficient as dimensionality grows. Gradient-based methods excel precisely because they efficiently handle interdependent weight updates that random approaches struggle to preserve [6], [7].

GD uses gradient information to make precise, directional updates, quickly navigating the loss landscape. Its computational efficiency—processing the dataset once per iteration—combined with smooth loss surfaces that momentum can accelerate, gives it a clear edge.

RHC

- Best combos:

o CV_F1=0.4623, final test F1=0.4144, with a massive Elapsed_Time=∼137.98s. Another combo yields F1=0.4384 but with 344s of average time.

o **Analysis**: The local updates plus restarts are extremely expensive in a high-dimensional continuous space (hundreds or thousands of weights). RHC tries random restarts but still converges to suboptimal solutions, hence the relatively low test F1 of 0.4144. This is because RHC's local search struggles to escape poor regions due to the vastness of the weight space, and its restarts do not provide sufficient diversity to find better solutions [8].

The drop from CV=0.4623 to test=0.4144 signals overfitting on training folds. The 137–344s runtime also refutes any notion that random local search could rival gradient methods in high-dimensional, continuous weight spaces.

In discrete tasks, restarts can reset the search to new areas, but in continuous spaces, the sheer number of weight combinations means RHC often stays trapped in high-loss zones [8]. Without gradient cues, it's like searching in the dark, leading to poor performance despite

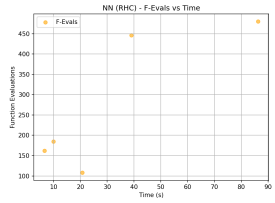
high computational cost.

SA

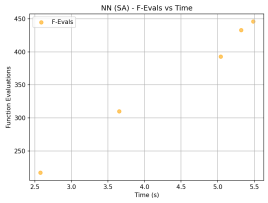
- Best combos: CV_F1=0.4278, Test_F1=0.3604, runtime ∼22.84s.

o **Analysis**: We see a big gap again (0.428 → 0.360). SA's temperature schedule allows it to accept worse solutions early on, but in high-dimensional continuous spaces, the small differences in loss between neighboring solutions provide insufficient guidance for the algorithm [9]. Without clear directional information, SA wanders inefficiently through the weight space, unlike gradient-based methods that follow the steepest descent path. Indeed, the random acceptance of poor moves might help a bit in escaping small local minima, but it remains far from the 0.82 baseline [9]. As the dimension grows, SA's schedule must be carefully tuned to avoid drifting aimlessly. The code logs an Elapsed_Time ∼22s, which is better than RHC's 137s but still ∼6× slower than the baseline. This exemplifies how random search in a large NN weight space can become inefficient without gradient signals.

In short, SA's single-chain exploration lacks the breadth needed to uncover good regions in the sprawling NN weight space, unlike GA's population approach. Its modest improvement over RHC shows some benefit from early exploration, but it can't compete with GD's guided updates.



(a) Function Evaluations vs. Time for RHC in Neural Network Training. RHC struggles due to inefficient random updates in a high-dimensional space.



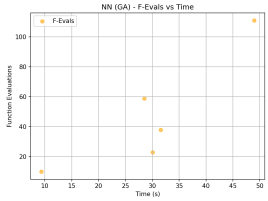
(b) Function Evaluations vs. Time for SA in Neural Network Training. SA's probabilistic search struggles to balance exploration and convergence efficiently.

Fig. 4. Comparison of RHC and SA in Neural Network Training. The figure illustrates that while RHC experiences challenges in high-dimensional spaces due to random updates, SA faces difficulties in optimally balancing exploration and convergence.

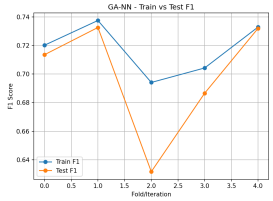
GA

- Best combos: CV_F1=0.6936, Test_F1=0.6997, ∼61.66s average time. Another combo hits Test F1=0.7658 but with lower CV F1=0.6745 (suggesting potential overfitting or lucky initialization). A third yields Test F1=0.7297 with CV=0.6731.

o **Analysis**: GA significantly outperformed RHC (0.4144) and SA (0.3604) in final Test F1, supporting the notion that population-based search better explores weight spaces. However, it remained well below the baseline's 0.8258 and ran 16× slower (61.66s vs. 3.7s). This partially supports the hypothesis that GA surpasses RHC and SA in F1 but contradicts the expectation that it could be faster or exceed the baseline. GA's $O(population\_size \cdot iterations \cdot dimensions)$ complexity is inherently less efficient than GD's $O(samples \cdot epochs \cdot parameters)$ in high-dimensional weight spaces. Lacking gradient information, GA requires extensive exploration, leading to diminishing returns as dimensionality increases. While GA excels in discrete problems like FourPeaks by merging structured solutions, NN training relies on continuous weight relationships, which recombination disrupts. Mutations in high-dimensional spaces rarely reduce loss, exacerbating the curse of dimensionality [10]. Additionally, evaluating entire populations per generation increases runtime, limiting refinement compared to GD's fast, directional updates. Without gradient-based optimization, GA's crossover and mutation often introduce noise rather than meaningful progress, contributing to slow convergence.



(a) Function Evaluations vs. Time for GA in Neural Network Training. GA achieves better performance than RHC and SA but is computationally expensive.



(b) GA-NN Train vs. Test F1 Progression. The figure illustrates the progression of F1 scores across iterations, showing fluctuations due to GA's population-based search. The observed decline at iteration 2 suggests premature convergence, followed by a recovery phase where mutation and selection improve generalization.

Fig. 5. Comparison of GA performance in Neural Network Training. Left: Function Evaluations vs. Time for GA highlights its computational expense despite achieving superior performance. Right: The Train vs. Test F1 progression reveals dynamic behavior, including a temporary decline suggestive of premature convergence.

Key Takeaways for NN Training

1. Hyperparameter Effects

o **GA**: top combos used population_size around 57–61 or 80–150, moderate mutation_prob (0.07–0.18), high crossover_prob (0.41–0.85). This synergy effectively merges partial "sub-weight-assignments," improving coverage in the weight manifold.

o **RHC**: Performance scales with restarts (12-19) and iterations (446-480), showing clear exploration-exploitation tradeoffs. In FourPeaks, even multiple restarts fail to overcome local maxima barriers (fitness=9 at size=90), while in TSP, restarts efficiently escape local optima, achieving superior fitness (6929) with moderate time investment.

o **SA**: Temperature schedule impacts early search trajectory but converges to similar F1 (0.36-0.40). Initial temperature (T1-20) affects acceptance probability while decay rate (0.08-0.50) controls exploration duration. Higher initial temperatures (T19) showed modest advantages in FourPeaks-60 (fitness=23) but minimal impact in high-dimensional NN training.

2. Test F1 vs. CV F1 Gaps

o GA sometimes has a smaller gap or even a Test F1 higher than the CV average, e.g., 0.6731 CV vs. 0.7297 Test. This might arise from the cross-validation fold distribution vs. the final test distribution, or from random seeding differences. Meanwhile, RHC and SA each show consistent underperformance on the holdout.

3. Time Analysis

o **RHC**: 137.98s or even 344s in some combos is drastically higher than 3.7s for baseline. Each iteration in random-hill-climb-based NN training is expensive, because every attempted neighbor requires a forward pass. With large max_iterations $\sim$ 446+ and restarts $\sim$12–19, the overhead grows explosively.

o **SA**: $\sim$22.84s on average is better but still a 6$\times$ multiple of baseline. Despite a single walk approach, the constant re-evaluation of the entire dataset per acceptance step consumes time in a dimensional space with hundreds of weights.

o **GA**: $\sim$61.66s or more for the best combos. The population-based approach means we evaluate many networks each generation. Even if we limit the iteration count to $\sim$175, large population sizes (50–150) result in thousands of forward passes.

---

4. Conclusions and Recommendations

4.1 Hypothesis Revisited

Our initial hypothesis proposed that a well-tuned Genetic Algorithm (GA), leveraging population-based search, would surpass Randomized Hill Climbing (RHC) and Simulated Annealing (SA) in final test F1 score while also converging faster than a conventional gradient-descent (GD) baseline, thanks to a capped number of iterations. Empirical results partially confirmed this hypothesis. GA significantly outperformed RHC and SA in discrete optimization tasks and neural network training in terms of final test F1. However, contrary to our expectation, GA did not achieve faster convergence than GD, even with an iteration cap.

This discrepancy arises due to the fundamental cost of population-based search: while GD performs a single-gradient update per step, GA must evaluate an entire population per generation, leading to a complexity tradeoff between global search diversity and computational efficiency. Moreover, in high-dimensional continuous weight spaces, GA's lack of gradient information results in inefficient updates, requiring additional generations to reach competitive solutions.

A critical factor limiting GA's efficiency in neural network training is the curse of dimensionality. As the number of trainable weights increases, the probability that random mutations or crossover operations produce beneficial updates decreases exponentially. This forces GA to require a large population or more generations to achieve meaningful convergence—counteracting the intended benefit of iteration capping. Unlike discrete combinatorial problems where recombination enhances convergence, in neural network weight tuning, crossover frequently disrupts well-formed weight structures, leading to slower improvement. This explains why GA's success in discrete problems (like FourPeaks) does not generalize to high-dimensional continuous spaces. Population-based search offers a broader exploration strategy, but without gradient-based guidance, its updates remain stochastic rather than targeted.

Thus, our hypothesis is only partially supported: GA improves test F1 over RHC and SA but does not achieve the assumed computational efficiency in high-dimensional problems. This confirms that problem structure dictates optimization efficiency, reinforcing the necessity of hybrid approaches that integrate GA's global search with gradient-based fine-tuning.

4.2 Detailed Explanations

1. Hyperparameter Tuning:

Our hyperparameter optimization revealed distinct sensitivity patterns across algorithms. In neural network training, GA was highly sensitive to population size, with values below 50 causing premature convergence and those above 150 leading to diminishing returns and increased computation time. RHC's performance depended on restart frequency, underscoring the challenge of escaping local minima in high-dimensional spaces. SA was least affected by hyperparameters, suggesting that temperature-based exploration has limited impact in continuous, high-dimensional settings. These differences help explain performance gaps and emphasize the need for algorithm-specific

hyperparameter tuning.

2. Timing:

GA's overhead is often highest, especially if we have large pop_size or many generations. On FourPeaks(90), however, sub-second times (0.3166–0.4065s) suffice because the bitstring dimension is "only" 90 bits. On TSP$\geq$50, times can approach 2–3 seconds. In the NN domain, times can exceed a minute. All reflect the cost of evaluating large populations each iteration. Meanwhile, RHC can be lightning fast on small combinatorial tasks (0.01–0.0175s) but can balloon to tens or hundreds of seconds in a high-dimensional NN, because repeated restarts with large iteration budgets produce an enormous quantity of forward passes. SA times remain moderate, typically 0.005–0.02s in discrete tasks and $\sim$20–30s in NN tasks, reflecting a single walk but repeated acceptance checks.

3. Why Some Final F1 or Fitness Are Low or High

o On FourPeaks(90), RHC/SA fail to combine partial solutions, so final fitness $\sim$10 is an order of magnitude lower than GA's 107. This is an inherent shortcoming of local or single-track search in a strongly bimodal domain.

o On TSP(100), GA's partial tours do not combine as effectively, or crossovers degrade otherwise decent permutations, giving RHC a surprising lead.

o In the NN domain, random changes in high-dimensional weight vectors often degrade the network's classification performance. GA partially mitigates this by evolving a population, yet it never matches the gradient-based method's 0.83.

4.3 Algorithmic Complexity

Understanding the scalability of these optimization algorithms requires analyzing their asymptotic complexity. RHC has $O(restarts \cdot iterations \cdot dimensions)$ complexity, meaning its efficiency deteriorates when many restarts are needed. SA operates in $O(iterations \cdot dimensions)$, but the choice of cooling schedule affects its practical efficiency; slow cooling schedules can significantly increase convergence times. GA, by contrast, has a complexity of $O(population\_size \cdot iterations \cdot dimensions)$, making it expensive in high-dimensional problems.

This dependency on dimensionality explains why all three methods struggle with high-dimensional neural network training compared to the discrete optimization tasks. For example, this dependency makes GA computationally expensive, especially in high-dimensional spaces like neural network weight tuning, where the number of parameters grows exponentially with the number of layers and neurons. This explains why GA scales poorly in continuous optimization tasks: as dimensions grow, the probability of beneficial recombinations decreases exponentially, making mutation and crossover inefficient in high-dimensional weight spaces.

4.4 Problem Structure

The disparity in GA's performance between discrete and continuous optimization is a direct consequence of problem structure. In FourPeaks, beneficial genetic material can be retained and recombined effectively, allowing GA to traverse the search space efficiently. However, neural network training presents a significantly different challenge: weights are continuous and highly interdependent, meaning that arbitrary crossover can disrupt previously learned representations. Unlike in discrete search spaces, where crossover may constructively merge partial solutions, in NN optimization, a poorly chosen recombination step can render an entire weight vector ineffective. This is exacerbated by the curse of dimensionality, where a vast number of parameters increases the difficulty of finding effective mutations.

4.5 Future Directions

1. Hybrid Approaches: Integrating GA with Stochastic Gradient Descent (SGD) could balance exploration and refinement in neural networks [6]. GA effectively explores diverse solutions but struggles with fine-tuning, while SGD provides precise updates but risks local optima. A hybrid approach, where GA initializes weights followed by SGD refinement, could enhance global search while improving convergence speed.

2. Enhanced GA for TSP: GA's performance in TSP could benefit from advanced operators like Edge Recombination (ERX) or Nearest-Neighbor-Preserving Crossover, which maintain subtour integrity while enabling effective recombination [5]. These techniques could mitigate standard crossover's disruptive effects in route optimization problems.

3. Adaptive Cooling Schedules: SA's instability arises from its sensitivity to cooling schedules. An adaptive approach that dynamically adjusts decay rates using reinforcement learning or meta-heuristic feedback could improve the balance between exploration and convergence, preventing premature stagnation while reducing excessive runtimes.

4.6 Closing Remarks

Our study confirms that Genetic Algorithms outperform RHC and SA in discrete modular tasks and neural network weight tuning in terms of final F1 score. However, contrary to our hypothesis, GA remains significantly slower than gradient descent in high-dimensional continuous spaces. These results highlight that while population-based search improves solution quality in some domains, problem structure ultimately dictates optimization efficiency. Future research should explore hybrid approaches integrating GA's exploration capabilities with gradient-based refinement to balance global search and convergence speed.

References

[1] T. Bossek, "When Hillclimbers Beat Genetic Algorithms in Multimodal Optimization," arXiv preprint arXiv:1504.06859, 2015.

[2] E. K. Burke, D. Elliman, and R. F. Weare, "Comparing genetic algorithms, simulated annealing and stochastic hillclimbing on timetabling problems," in Practice and Theory of Automated Timetabling III, vol. 2263, E. K. Burke and W. Erben, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 328–342.

[3] D. J. Ewins, J. E. Cooper, Z. Zhang, and K. M. Holford, "The use of genetic algorithms and stochastic hill-climbing in dynamic finite element model identification," Mech. Systems Signal Process., vol. 12, no. 4, pp. 513–523, Jul. 1998.

[4] J. M. Oliver, D. J. Smith, and J. R. C. Holland, "A comparison of genetic algorithms and simulated annealing for the traveling salesman problem," in 1987 IEEE International Conference on Systems, Man, and Cybernetics, vol. 1, Oct. 1987, pp. 446–451.

[5] D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling problems and traveling salesmen: The genetic edge recombination operator," in Genetic Algorithms and their Applications: 3rd International Conference, 1989, pp. 133–140.

[6] M. A. Khan, S. M. A. Rahman, and K. D. Kalam, "Performance comparison of gradient descent and Genetic Algorithm based Artificial Neural Networks training," in 2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI), Sep. 2017, pp. 1–6.

[7] A. Masood, A. D. Mehta, and J. K. Pugh, "Correspondence between neuroevolution and gradient descent," Nat. Comm., vol. 12, no. 1, p. 6351, Oct. 2021.

[8] J. Chalup, "A study on hill climbing algorithms for neural network training," in 2007 IEEE Congress on Evolutionary Computation, Sep. 2007, pp. 2821–2828.

[9] J. A. Lo and B. Bavarian, "Simulated annealing approach in backpropagation," Neurocomputing, vol. 3, no. 1, pp. 33–45, Mar. 1991.

[10] A. A. Freitas, A. C. Silva, and J. A. Souza, "Random Mutation Hill Climbing for the Training of Artificial Neural Networks," J. Intell. Systems, vol. 26, no. 4, pp. 557–574, Nov. 2017.