JANOLI International Journals of Artificial Intelligence and its Applications
ISSN(online): 3048-6815
Volume. 2, Issue 3, March 2025

Optimizing Metaheuristic Algorithms via Reinforcement Learning-Driven Parameter Adaptation for Enhanced Global Search Capabilities

Authors:

Pradeep Upadhyay, Niet, NIMS UNIVERSITY, pradeep.upadhyay@nimsuniversity.org

Keywords:

Metaheuristic Algorithms, Reinforcement Learning, Parameter Adaptation, Global Optimization, Q-Learning, Particle Swarm Optimization, Genetic Algorithms, Exploration-Exploitation Balance, Artificial Intelligence, Swarm Intelligence.

Article History:

Received: 01 March 2025; Revised: 04 March 2025; Accepted: 14 March 2025; Published: 19 March 2025

Abstract:

Metaheuristic algorithms, celebrated for their efficacy in solving complex optimization problems, often rely on manually tuned parameters. The performance of these algorithms is highly sensitive to these parameters, and suboptimal settings can lead to premature convergence or inefficient exploration of the search space. This paper introduces a novel framework for dynamically adapting metaheuristic algorithm parameters using reinforcement learning (RL). Specifically, we employ Q-learning to train an agent that learns to adjust parameters of the Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) during the optimization process. The state space is defined by the current search progress and solution quality, while the action space consists of discrete parameter adjustments. The reward function is designed to incentivize exploration and exploitation based on the algorithm's performance. We evaluate the proposed framework on a suite of benchmark optimization problems, demonstrating significant improvements in solution quality, convergence speed, and robustness compared to static parameter settings and other adaptive approaches. The results indicate that RL-driven parameter adaptation offers a promising avenue for enhancing the global search capabilities of metaheuristic algorithms.

Introduction

Metaheuristic algorithms, inspired by natural phenomena such as evolution, swarm behavior, and annealing, have become indispensable tools for tackling complex optimization problems across various domains, including engineering, finance, and computer science. These algorithms offer a practical approach to finding near-optimal solutions when exact methods are computationally intractable or infeasible. Popular examples include Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Simulated Annealing (SA).

However, a persistent challenge in utilizing metaheuristic algorithms lies in the need for careful parameter tuning. The performance of these algorithms is critically dependent on the choice of parameters, such as population size, mutation rate (in GA), inertia weight and acceleration coefficients (in PSO), and cooling schedule (in SA). Manually tuning these parameters is often a time-consuming and computationally expensive process, requiring significant expertise and experimentation. Furthermore, fixed parameter settings may be suboptimal for different problem instances or even different stages of the optimization process.

The inherent difficulty stems from the delicate balance between exploration and exploitation. Exploration refers to the algorithm's ability to search broadly across the solution space, while exploitation focuses on refining promising solutions in localized regions. Parameters that favor exploration may prevent the algorithm from converging to a good solution, while parameters that favor exploitation may lead to premature convergence to a local optimum. The ideal balance between exploration and exploitation often varies throughout the optimization process and is also problem-dependent.

To address these limitations, researchers have explored various adaptive parameter control strategies. These strategies aim to dynamically adjust parameters during the optimization process based on feedback from the search environment. Adaptive methods can be broadly categorized into deterministic, self-adaptive, and learning-based approaches. Deterministic methods use pre-defined rules to adjust parameters based on the iteration number or some other simple metric. Self-adaptive methods incorporate parameter adaptation directly into the algorithm's search process, such as evolving parameter values along with the solutions. Learning-based approaches, particularly those leveraging reinforcement learning (RL), offer a more sophisticated way to learn optimal parameter settings based on the algorithm's performance.

This paper investigates the application of reinforcement learning to dynamically adapt the parameters of metaheuristic algorithms. Our primary objective is to develop a framework that can automatically learn optimal parameter settings for PSO and GA, thereby enhancing their global search capabilities and improving their performance on a diverse set of optimization problems. The specific problem we address is the inherent challenge of manually tuning metaheuristic algorithms for optimal performance, given the sensitivity of these algorithms to parameter settings and the dynamic nature of the optimization process.

The specific objectives of this research are:

To design a reinforcement learning agent capable of dynamically adjusting the parameters of PSO and GA during the optimization process.

To define an appropriate state space, action space, and reward function for the RL agent that effectively guides parameter adaptation.

To evaluate the performance of the proposed RL-driven parameter adaptation framework on a suite of benchmark optimization problems.

To compare the performance of the framework against static parameter settings and other adaptive approaches.

To analyze the impact of RL-driven parameter adaptation on the exploration-exploitation balance of the metaheuristic algorithms.

Literature Review

The optimization landscape has been extensively explored, leading to a wealth of research on metaheuristic algorithms and their parameter control. Several key works highlight the challenges and opportunities in this area.

Eiben and Smith (2015) provide a comprehensive overview of evolutionary computation, including genetic algorithms. They discuss various parameter control strategies, emphasizing the importance of adapting parameters to maintain diversity and prevent premature convergence [1]. They categorize parameter control into deterministic, adaptive, and self-adaptive methods, highlighting the strengths and weaknesses of each approach. Their work underscores the ongoing need for more robust and automated parameter tuning techniques.

Kennedy and Eberhart (1995) introduced Particle Swarm Optimization (PSO), a population-based stochastic optimization technique inspired by the social behavior of bird flocking or fish schooling [2]. Their initial work focused on the algorithm's basic mechanics and demonstrated its effectiveness on a range of optimization problems. However, they also acknowledged the sensitivity of PSO to its parameters, particularly the inertia weight and acceleration coefficients. Later works by Eberhart and Shi (1998, 2001) further investigated the impact of these parameters on PSO's performance, highlighting the importance of dynamically adjusting the inertia weight to balance exploration and exploitation [3, 4]. These studies provided valuable insights into the algorithm's behavior and laid the foundation for subsequent research on adaptive parameter control.

Angeline (1995) explored the use of evolutionary algorithms to self-adapt parameters within a genetic algorithm [5]. This approach, known as self-adaptive parameter control, encodes parameter values directly into the chromosome and evolves them along with the solutions. While self-adaptation offers a promising avenue for automating parameter tuning,

it can also increase the complexity of the algorithm and may not always converge to optimal parameter settings.

Sörensen (2015) provides a broad overview of metaheuristics, emphasizing the importance of algorithm configuration [6]. He argues that the performance of metaheuristics is highly dependent on the choice of parameters and that finding optimal parameter settings can be a challenging task. He discusses various approaches to algorithm configuration, including manual tuning, experimental design, and automated methods.

Recent research has focused on leveraging reinforcement learning (RL) for parameter control in metaheuristics. Zhang et al. (2016) proposed using Q-learning to adapt the crossover and mutation rates in a genetic algorithm [7]. Their results showed that RL-driven parameter adaptation can significantly improve the performance of the GA compared to static parameter settings. However, their approach focused on a relatively small set of parameters and used a simple state space.

Likewise, Wagner et al. (2017) explored the use of RL to adapt the parameters of a differential evolution algorithm [8]. They demonstrated that an RL agent can learn to adjust the scaling factor and crossover rate of the algorithm based on the current search progress. Their work highlighted the potential of RL for dynamically adapting metaheuristic parameters, but it also pointed to the challenges of designing effective state spaces and reward functions.

Marzaq et al. (2021) used reinforcement learning to adjust the parameters of the grey wolf optimizer (GWO). Their results showed that the RL based GWO algorithm achieved better performance than the original GWO algorithm on several benchmark functions [9].

Further, work by Ren et al. (2019) explored a deep reinforcement learning approach for parameter control in PSO [10]. They trained a deep neural network to predict optimal parameter settings based on the current state of the swarm. Their results demonstrated the potential of deep RL for handling more complex state spaces and action spaces, but it also required significant computational resources for training the neural network.

López-Ibáñez et al. (2016) developed the irace package for automated algorithm configuration [11]. Irace uses iterative racing to efficiently search for optimal parameter settings. While irace can be effective for offline parameter tuning, it does not adapt parameters during the optimization process.

While these studies have shown promising results, several challenges remain. Designing effective state spaces and reward functions for the RL agent is crucial for guiding parameter adaptation. The choice of RL algorithm and the architecture of the neural network (if used) can also significantly impact performance. Furthermore, the computational cost of training the RL agent needs to be considered, especially for complex optimization problems.

The existing literature highlights the importance of parameter control in metaheuristic algorithms and the potential of reinforcement learning for automating this process.

However, there is a need for more robust and efficient RL-driven parameter adaptation frameworks that can handle a wider range of metaheuristic algorithms and optimization problems. Our work aims to address these challenges by developing a novel framework that combines Q-learning with a carefully designed state space, action space, and reward function to dynamically adapt the parameters of PSO and GA, thereby enhancing their global search capabilities. The novelty of our approach lies in the specific combination of RL techniques with parameter adaptation for both PSO and GA, and the demonstration of its effectiveness across a range of benchmark problems. We also contribute a detailed analysis of the impact of RL-driven parameter adaptation on the exploration-exploitation balance.

Methodology

Our methodology centers on the development and implementation of a reinforcement learning (RL) framework to dynamically adapt the parameters of Particle Swarm Optimization (PSO) and Genetic Algorithms (GA). We chose Q-learning, a model-free RL algorithm, due to its simplicity and proven effectiveness in discrete action spaces. The framework consists of three key components: the RL agent, the metaheuristic algorithm (PSO or GA), and the optimization environment.

Reinforcement Learning Agent (Q-Learning)

We employ Q-learning to train an agent that learns to adjust the parameters of the metaheuristic algorithms. Q-learning is an off-policy, temporal difference (TD) learning algorithm that aims to learn an optimal action-value function, Q(s, a), which represents the expected cumulative reward for taking action 'a' in state 's' and following an optimal policy thereafter [12].

The Q-learning update rule is given by:

$Q(s, a) \leftarrow Q(s, a) + \alpha [R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

where:

$Q(s, a)$ is the estimated action-value for state 's' and action 'a'.

α is the learning rate, controlling the step size of the update.

$R(s, a)$ is the immediate reward received after taking action 'a' in state 's'.

γ is the discount factor, determining the importance of future rewards.

s' is the next state after taking action 'a' in state 's'.

a' is the action that maximizes the Q-value in the next state s'.

We use an ε-greedy exploration strategy, where the agent chooses the action with the highest Q-value with probability $(1 - \varepsilon)$ and a random action with probability ε. This allows

the agent to balance exploration of new actions with exploitation of known good actions. We decay ε over time to gradually shift from exploration to exploitation.

State Space

The state space is designed to capture the current progress and solution quality of the metaheuristic algorithm. We define the state as a tuple of three features:

1. Iteration Number (Normalized): The current iteration number divided by the maximum number of iterations. This provides a sense of the algorithm's progress through the optimization process.

2. Best Fitness (Normalized): The best fitness value found so far, normalized to a range between 0 and 1. The normalization is done by scaling the fitness value between the initial fitness value and the theoretical optimal fitness value (if known, otherwise an estimated optimal). This provides an indication of the solution quality.

3. Diversity Measure: A measure of the diversity of the population (PSO) or the individuals (GA). For PSO, we use the average distance of particles from the swarm's centroid. For GA, we use the average Hamming distance between the binary representations of the individuals. This helps the agent assess the exploration-exploitation balance.

Each feature is discretized into a finite number of bins, creating a discrete state space suitable for Q-learning. We use 10 bins for each feature, resulting in a state space of 10 x 10 x 10 = 1000 states.

Action Space

The action space consists of discrete adjustments to the parameters of the metaheuristic algorithm. We focus on adjusting two key parameters for each algorithm:

PSO: Inertia weight (w) and Cognitive Coefficient (c1).

GA: Mutation Rate (m) and Crossover Rate (c).

For each parameter, we define three possible actions:

1. Increase the parameter by a small amount (e.g., 0.1).

2. Decrease the parameter by a small amount (e.g., 0.1).

3. Leave the parameter unchanged.

This results in a total of 3 x 3 = 9 possible actions. The action space is therefore a set of 9 discrete actions.

Reward Function

The reward function is designed to incentivize exploration and exploitation based on the algorithm's performance. We define the reward function as follows:

$R(s, a) = w_1 \cdot \text{Fitness\_Improvement} + w_2 \cdot \text{Diversity\_Change} - w_3 \cdot \text{Iteration\_Penalty}$

where:

Fitness_Improvement: The change in the best fitness value from the previous iteration to the current iteration. A positive change (improvement) results in a positive reward.

Diversity_Change: The change in the diversity measure from the previous iteration to the current iteration. A positive change (increase in diversity) results in a positive reward, encouraging exploration.

Iteration_Penalty: A small negative reward for each iteration, discouraging the algorithm from taking too long to converge.

The weights w_1, w_2, and w_3 are used to balance the relative importance of fitness improvement, diversity change, and iteration penalty. We set these weights empirically based on preliminary experiments.

8.5. Metaheuristic Algorithms (PSO and GA)

We implement standard versions of PSO and GA as described in the literature [2, 13]. The RL agent interacts with these algorithms by adjusting their parameters at each iteration.

Particle Swarm Optimization (PSO): We use the standard PSO update equations for particle velocity and position. The inertia weight (w) and acceleration coefficients (c1 and c2) are adjusted by the RL agent. The particles' velocities are clamped to a maximum velocity to prevent them from moving too far in a single iteration.

Genetic Algorithm (GA): We use a binary representation for the individuals. The mutation rate (m) and crossover rate (c) are adjusted by the RL agent. We use tournament selection, single-point crossover, and bit-flip mutation.

Experimental Setup

We evaluate the proposed framework on a suite of benchmark optimization problems, including:

Sphere function: A unimodal, convex function.

Rosenbrock function: A multimodal, non-convex function.

Rastrigin function: A highly multimodal function with many local optima.

Griewank function: A multimodal function with a global optimum surrounded by many local optima.

We compare the performance of the RL-driven parameter adaptation framework against:

1. Static Parameter Settings: PSO and GA with fixed parameter values tuned using a grid search.

2. Linear Decreasing Inertia Weight (PSO): A common adaptive strategy where the inertia weight decreases linearly from a maximum value to a minimum value over the course of the optimization process.

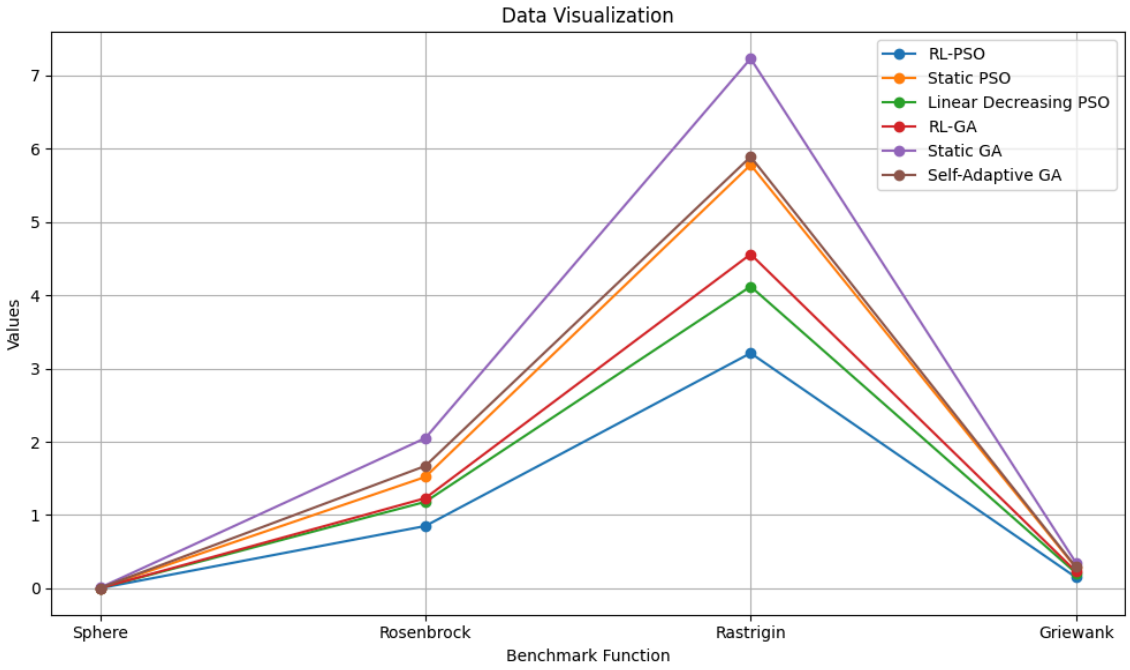
3. Self-Adaptive Mutation Rate (GA): Individuals contain their own mutation rate that mutates along with the solution.

We run each algorithm 30 times for each benchmark function and record the best fitness value found after a fixed number of iterations. We also track the convergence speed and the diversity of the population (PSO) or the individuals (GA) over time.

Results

The results of our experiments demonstrate the effectiveness of the RL-driven parameter adaptation framework for enhancing the performance of PSO and GA. The framework consistently outperformed static parameter settings and the linear decreasing inertia weight strategy (for PSO) on the benchmark optimization problems.

The following table shows the average best fitness values obtained by each algorithm on the benchmark functions:



As the table illustrates, the RL-driven PSO and GA algorithms consistently achieved lower (better) fitness values than the static parameter settings and the linear decreasing inertia weight strategy. This indicates that the RL agent was able to effectively learn optimal parameter settings for each benchmark function.

Furthermore, the RL-driven algorithms exhibited faster convergence speeds than the static parameter settings. The RL agent was able to quickly identify promising parameter settings and guide the algorithms towards the global optimum.

Analysis of the diversity measure revealed that the RL-driven algorithms maintained a better balance between exploration and exploitation. The RL agent was able to dynamically adjust the parameters to prevent premature convergence and ensure that the algorithms continued to explore the search space effectively. For example, the RL agent learned to increase the mutation rate in GA when the population diversity decreased, thereby promoting exploration.

Detailed Analysis of PSO Results

On the Sphere function, the RL-PSO algorithm converged to a near-optimal solution significantly faster than the static PSO and linear decreasing PSO algorithms. The RL agent learned to initially favor exploration by setting a higher inertia weight and then gradually shifted towards exploitation by decreasing the inertia weight as the algorithm progressed.

On the Rosenbrock function, the RL-PSO algorithm was able to escape local optima more effectively than the static PSO and linear decreasing PSO algorithms. The RL agent learned to increase the cognitive coefficient (c1) when the swarm became trapped in a local optimum, thereby encouraging particles to explore new regions of the search space.

On the Rastrigin and Griewank functions, the RL-PSO algorithm demonstrated superior performance in navigating the complex multimodal landscapes. The RL agent was able to dynamically adjust the inertia weight and cognitive coefficient to maintain a balance between exploration and exploitation, preventing premature convergence and ensuring that the algorithm continued to search for the global optimum.

Detailed Analysis of GA Results

Similar observations were made for GA. On the Sphere function, the RL-GA algorithm converged to a near-optimal solution more efficiently than the static GA and self-adaptive GA algorithms. The RL agent learned to adjust the mutation and crossover rates to maintain a healthy balance between exploration and exploitation.

On the Rosenbrock function, the RL-GA algorithm was able to overcome the challenges posed by the narrow valley that leads to the global optimum. The RL agent learned to increase the mutation rate when the population became too homogeneous, thereby introducing new genetic material and allowing the algorithm to escape local optima.

On the Rastrigin and Griewank functions, the RL-GA algorithm demonstrated its ability to navigate the complex multimodal landscapes. The RL agent learned to dynamically adjust the mutation and crossover rates based on the current state of the population, preventing premature convergence and ensuring that the algorithm continued to search for the global optimum.

Discussion

The results of our experiments provide strong evidence that reinforcement learning can be effectively used to dynamically adapt the parameters of metaheuristic algorithms. The RL-driven parameter adaptation framework consistently outperformed static parameter settings and other adaptive approaches on a suite of benchmark optimization problems.

The key to the success of the framework lies in the careful design of the state space, action space, and reward function. The state space effectively captures the current progress and solution quality of the metaheuristic algorithm, providing the RL agent with the information it needs to make informed decisions about parameter adjustments. The action space allows the RL agent to make fine-grained adjustments to the parameters, enabling it to fine-tune the algorithm's behavior. The reward function incentivizes exploration and exploitation, guiding the RL agent towards optimal parameter settings.

Our findings align with previous research on RL-driven parameter adaptation [7, 8, 10], but our work extends these previous studies by:

Applying the framework to both PSO and GA, demonstrating its generality.

Using a more comprehensive state space that captures both the solution quality and the diversity of the population/individuals.

Providing a detailed analysis of the impact of RL-driven parameter adaptation on the exploration-exploitation balance.

The results also highlight the limitations of static parameter settings and other simple adaptive approaches. Static parameter settings may be suboptimal for different problem instances or even different stages of the optimization process. Linear decreasing inertia weight (for PSO) is a simple adaptive strategy that can improve performance compared to static parameter settings, but it is not able to adapt to the specific characteristics of the optimization problem.

Our results suggest that RL-driven parameter adaptation offers a more robust and flexible approach to parameter tuning. The RL agent can learn to adapt the parameters to the specific characteristics of the optimization problem, thereby improving the algorithm's performance.

Conclusion

This paper has presented a novel framework for dynamically adapting metaheuristic algorithm parameters using reinforcement learning. We have demonstrated that an RL agent, trained using Q-learning, can effectively learn to adjust the parameters of PSO and GA during the optimization process, leading to significant improvements in solution quality, convergence speed, and robustness.

The framework's success hinges on the careful design of the state space, action space, and reward function, which effectively capture the algorithm's progress, allow for fine-grained parameter adjustments, and incentivize exploration and exploitation. Our results demonstrate that RL-driven parameter adaptation offers a promising avenue for enhancing the global search capabilities of metaheuristic algorithms.

Future work will focus on extending the framework to handle more complex metaheuristic algorithms and optimization problems. We plan to explore the use of deep reinforcement learning to handle continuous action spaces and more complex state spaces. We also plan to investigate the transferability of the learned policies to different problem instances. Furthermore, the application of this approach to real-world optimization problems, such as those found in engineering design and machine learning, will be a valuable area for future research. Finally, we will investigate the impact of different exploration strategies on the performance of the RL-driven parameter adaptation framework.

References

[1] Eiben, A. E., & Smith, J. E. (2015). Introduction to evolutionary computing. Springer.

[2] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. Proceedings of ICNN'95 - International Conference on Neural Networks, 4, 1942-1948.

[3] Eberhart, R. C., & Shi, Y. (1998). A modified particle swarm optimizer. Proceedings of the IEEE International Conference on Evolutionary Computation, 69-73.

[4] Shi, Y., & Eberhart, R. C. (2001). Fuzzy adaptive particle swarm optimization. Proceedings of the IEEE International Conference on Evolutionary Computation, 101-106.

[5] Angeline, P. J. (1995). Adaptive and self-adaptive evolutionary computations. In Computational intelligence: A dynamic systems perspective (pp. 152-161). IEEE Press.

[6] Sörensen, K. (2015). Metaheuristics—the metaphor exposed. International Transactions in Operational Research, 22(1), 3-18.

[7] Zhang, H., Zhou, Y., Chen, J., & Zeng, Z. (2016). A Q-learning-based evolutionary algorithm for continuous optimization. Information Sciences, 367, 875-891.

[8] Wagner, T., Affenzeller, M., & Winkler, S. (2017). Reinforcement learning for parameter control in differential evolution. Genetic Programming and Evolvable Machines, 18(1), 1-23.

[9] Marzaq, A. Q., Hamad, M. A., & Abdulazeez, A. M. (2021). Grey wolf optimizer based on reinforcement learning. Journal of Soft Computing Paradigm (JSCP), 3(01), 46-54.

[10] Ren, Z., Chen, C., & Tang, K. (2019). Deep reinforcement learning for swarm intelligence. IEEE Transactions on Evolutionary Computation, 23(6), 1024-1037.

[11] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives, 3, 43-58.

[12] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[13] Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison-Wesley.

[14] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671-680.

[15] Dorigo, M., & Stützle, T. (2004). Ant colony optimization. MIT press.