Ant Colony Optimization with Policy Gradients and Replay

William Jardee
william.jardee@msu.montana.edu
Montana State University
Bozeman, Montana, USA

John W. Sheppard
john.sheppard@montana.edu
Montana State University
Bozeman, Montana, USA

ABSTRACT

Ant Colony Optimization (ACO) has served as a widely-utilized metaheuristic algorithm for decades for solving combinatorial optimization problems. Since its initial construction, ACO has seen a wide variety of modifications and connections to Reinforcement Learning (RL). Substantial parallels can be seen as early as 1995 with Ant-Q's relationship with Q-learning, through 2022 with ADACO's connection with Policy Gradient. In this work, we describe ACO, more specifically the Stochastic Gradient Descent ACO algorithm (ACOSGD), explicitly as an off-policy Policy Gradient (PG) method. We also incorporate experience replay into several ACO algorithm variants, including AS, MaxMin-ACO, ACOSGD, ADACO, and our two policy gradient-based versions: PGACO and PPOACO, drawing the connection to elitist ACO strategies. We show that our implementation of PG in ACO with experience replay and a baselined reward update strategy applied to eight TSP problems of varying sizes performs competitively with both fundamental ACO and SGD-based ACO versions. We also show that the replay buffer seems to unilaterally improve the performance of ACO algorithms through an ablation study.

CCS CONCEPTS

• **Computing methodologies** → **Heuristic function construction**; *Multi-agent reinforcement learning*; **Bio-inspired approaches**.

KEYWORDS

Ant Colony Optimization, Ant Algorithms, Metaheuristics, Reinforcement Learning, Replay Buffer, Policy Gradient

ACM Reference Format:
William Jardee and John W. Sheppard. 2025. Ant Colony Optimization with Policy Gradients and Replay. In *Genetic and Evolutionary Computation Conference (GECCO '25),July 14–18, 2025, Malaga, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3712256.3726452

1 INTRODUCTION

Ant Colony Optimization (ACO) has been a commonplace algorithm in the world of swarm intelligence and combinatorial optimization for multiple decades [6, 45]. In that time, the algorithm has seen various improvements that have mostly remained independently implemented, and few algorithms that have seen regular implementation. Like many other swarm intelligence models, parallels to the rest of the world of AI/ML/Stochastic optimization are often drawn to mixed success. Many of these models tout slight performance advantages and deserve consideration but face the struggle of pulling slight advantages at the cost of substantial implementation difficulty. To this end, commonly used algorithms see more application when bolstered by well known algorithms and common modifications.

To help unify the large number of ACO algorithm variants, we propose an explicit connection to the Policy Gradient (PG) paradigm. We claim that connecting ACO to the fundamentals of state-of-the-art Reinforcement Learning methods will allow for broader acceptance of the metaheuristic. Beyond stating this connection, which has been done by others before us [4, 22, 48], we argue that our extensions generalize the concepts behind the ACO elitist strategies and reinforcement functions. This is done, while also extending the gradient update to be an off-policy method, through the incorporation of importance sampling in the ACO update rule.

We hypothesize that the extension of ACO to account for the off-policy nature of previously sampled ants should provide an increase in performance, being more emphasized in difficult problems. Further, we expect that the incorporation of experience replay and a more sophisticated reward mechanism will yield significant improvements in the resulting ACO algorithm.

In this paper, we offer greater insight and improved performance of ACO by making the following contributions:

(1) We make the explicit connection between ACO and the Off-Policy, Policy Gradient method, implementing importance sampling and the Proximate Policy Optimization clipping function. To our knowledge, this is the first implementation of importance sampling to the update function of ACO and the PPO clipping function to ACO.
(2) We incorporate and evaluate experience replay as a generalization of the elitist strategies implemented by many ACO algorithms. We validate and explore the relationship between buffer size and problem size by an ablation study.
(3) We implement a variety of advantage functions in the ACO update rule and show that baselined reward seems to perform the best.
(4) We demonstrate how standard concepts in Reinforcement Learning (i.e., importance sampling, advantage functions, and experience replay) can be applied directly to ACO.

2 BACKGROUND

2.1 Combinatorial Optimization Problems

ACO is a commonly applied swarm intelligence model that tends to perform at its best when tackling shortest path problems [5, 6].

GECCO '25, July 14–18, 2025, Málaga, Spain

ACM ISBN 979-8-4007-1465-8/2025/07
https://doi.org/10.1145/3712256.3726452

The class of problems addressed by ACO, as introduced in [4, 40], involves, without loss of generality, a Combinatorial Optimization Problem (COP), denoted as (w, J, Ω), where the objective is to find a candidate solution, $w \in \mathcal{W}$, that minimizes an optimization function, $J(w)$, while following a set of constraints, Ω.

In the Traveling Salesperson Problem (TSP), we are given a graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of nodes and $\mathcal{E}$ the set of edges. The goal is to identify the shortest Hamiltonian tour within the graph. We define a set of feasible walks, $w \in \mathcal{W} \subset \mathcal{V}^n$, where $n = |\mathcal{V}|$, subject to the following constraints:

(1) We start at some start node, $v_0 \in \mathcal{V}$,
(2) Each walk, w, contains every node in $\mathcal{V}$ at most once, and
(3) The last node in w, v_f, has at most one successor, which is the start node; that is, for all $v_i \in \mathcal{N}(v_f)$, $v_i \in w$ and $v_0 \in \mathcal{N}(v_f)$, where $\mathcal{N}(v)$ is the neighborhood of v.

Our objective in TSP is to find the walk, w^*, that minimizes $J(w^*)$ and satisfies the constraints (1)–(3) and every node in $\mathcal{V}$ is in w^*. Here, J is the sum of all edge weights in w^* and in the case of a Euclidean graph problem, the edge weights are typically the Euclidean distance between nodes. This formulation follows the construction graph proposed in [25]. For a discussion of the connection between Markov Decision Processes and COP in the context of ACO, see [4, 25].

2.2 Reinforcement Learning

In Reinforcement Learning (RL), an agent receives some representation of the environment, $s_t \in \mathcal{S}$, and some set of available actions, $\mathcal{A}_t \subseteq \mathcal{A}$. The agent's objective is to learn a policy, $\pi : \mathcal{S} \to \mathcal{A}$, which maps the current state to the action that maximizes the expected return over time, G_t [44]. Alternatively, a stochastic policy maps states to a probability distribution over $\mathcal{A}$, $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. Several functions play a key role in RL:

- The quality function, $Q(s, a)$, measures the expected cumulative reward returned for each state-action pair, $(s, a) \in \mathcal{S} \times \mathcal{A}$. Methods that focus on Q are called Q-learning methods.
- The value function, $V(a)$, represents the expected cumulative reward for a given state, $s \in \mathcal{S}$. A common method for learning V is called Temporal Difference (TD) learning.
- The advantage, $A(s, a) = Q(s, a) - V(s)$, gives the difference between the expected reward for a given action, a, compared to the other actions at state s. The advantage shows up in many algorithms to reduce variance in the learning process.

A common assumption considered with RL problems is the Markov property, where the conditional dependence of all future steps only depends on the current state. The construction of a policy with the Markovian property addresses the Markov decision process (MDP), with nice optimality properties with regards to the Q- and V-functions.

Experience replay, a key component in modern reinforcement learning, boosts learning efficiency and stability [19, 29]. The method involves learning from samples stored and then drawn from a dataset, $\mathcal{D}$, called the replay buffer. This allows the learning process to be smoother by decoupling the samples used in learning from the current model by using historical data [44]. Experience replay can also be used to provide a set of examples to the model to bias training towards better traces known a priori. While popularized in Q-learning methods like Deep Q-Networks (DQN), the principles behind experience replay have been adapted to various RL algorithms, including some policy-based approaches [12, 33].

A foundational policy-based approach is the policy gradient algorithm (PG), which directly optimizes the parameters of the policy using gradient ascent. The parameters, θ, are updated as:

$$\theta_{t+1} = \theta_t + \rho \, \nabla_\theta J(\theta_t)$$
$$\nabla_\theta J(\theta_t) = \mathop{\mathbb{E}}_{(a_t, s_t) \sim \pi_{\theta_t}} \Big[G_t \, \nabla_\theta \log \pi(a_t \mid s_t \,;\, \theta_t) \Big], \tag{1}$$

where ρ is the learning rate. Alternative functions can be substituted in place of G_t (e.g., baselined reward, quality function, or advantage function), reducing variance while maintaining PG's theoretical guarantees [12, 24]. The REINFORCE algorithm [46] first derived Equation (1) and introduced a Monte Carlo algorithm to sample (a_t, s_t) from π_θ. [46] also demonstrated that subtracting a state-dependent baseline, b_{ij}, from G_t reduces update variance without introducing bias.

Two widely used policy gradient algorithms are Proximal Policy Optimization (PPO) [39] and Advantage Actor-Critic (A2C) [20, 32]. PPO, uses a clipping mechanism in its objective function to prevent excessive policy updates, leading to more stable learning. It is known for its robustness to hyperparameter variations and stability across a wide range of tasks [1, 20]. A2C uses an actor-critic architecture, where the actor network learns the policy via Equation (1) and a critic network estimates the value function to compute G_t for a given state. Although A2C can be computationally efficient and effective when properly tuned, it is often more sensitive to hyperparameters than both PPO and DQN [20].

2.3 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic often deployed to solving combinatorial optimization problems [6, 13]. Using the structure of the construction graph, we can relate a COP problem to a graph-based problem. Because of this, and for brevity, we will restrict our handling of ACO to TSP. There are various extensions that project this problem into the more broad COP [13, 23], continuous space problems [37], and a wide variety of applications [6, 9, 27].

The ACO algorithm's goal is to find a heuristic values, called "pheromones" and notated as T, to measure how preferable the edge is. Either the best solution found while searching for this pheromone level, or the whole table of pheromone levels can be returned. Ideally, greedily following this heuristic should return the optimal solution, meaning reporting of the heuristic trivially produces the optimal solution as well. Being a Population-Based Incremental Learning (PBIL) [3] method, ACO employs an iterative, stochastic, batch search over the problem space. In [4], this process is broken down into three steps:

(1) *The forward phase*: A Monte Carlo run over $\mathcal{G}$ that follows a stochastic policy parameterized by T.
(2) *The backward phase*: Each ant back-tracks over all the edges in their solution and proposes new values for T, T'.
(3) *The merge phase*: An update function combines all the T''s into the next generation's T.

Taking a closer look at the first, and most popular, version of ACO, Ant System (AS) (specifically the ant-cycle algorithm) [16] we define the steps as:

(1) *The forward phase*: Let m ants search the space, where each step is decided from the stochastic policy:

$$p_{i,j} = \begin{cases} \frac{\tau_{ij}^{\alpha} h_{ij}^{\beta}}{\sum_{l \in \mathcal{N}(i)} \tau_{il}^{\alpha} h_{il}^{\beta}} & \text{if } j \in \mathcal{N}(i) \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

where $\mathcal{N}(i)$ is the neighborhood of node i (i.e., all edges accessible from i and not in a tabu list), α and β are positive hyperparameters, $\tau_{ij} \in T$ are the learned heuristic for each edge, and h_{ij} is some heuristic prior passed to the algorithm, typically $1/d_{ij}$, where d_{ij} is the weight of the edge $\langle i, j \rangle$.

(2) *The backward phase*: Each ant creates a T' such that each edge in the ant's solution, w, is reinforced by $Q/L(w)$, where Q is some amount of pheromone and $L(w)$ is the path length.

(3) *The merge phase*: The pheromone table is reduced by a rate of $(1-\rho)$, a process called pheromone evaporation, and then increased by the sum of all proposed changes:

$$T \leftarrow (1-\rho)\, T + \sum_{k=1}^{m} \Delta T_k' \,.$$

ACO has seen many variants through the years. Elitist AS [17] and MMACO [41–43] were two of the first to restrict updates in the backward step to only the best performing ant(s). MMACO also included a clipping of the T values, preventing them from getting too large or small. Rank-AS [8], κ-best ACO [28], and Best-Worst Ant System (BWAS) [11] expand the Elitist strategy by keeping more than one best and adding weights according to how the ant's rank compare to the other peers in the generation. While Rank-AS and κ-best ACO add weights to better performing samples, BWAS aims to utilize positive and negative reinforcement learning to increment the best while decrementing the worst.

Some models, like, Hyper-Cube AS [7], and Graph Based AS (GBAS) [25, 26] focus on limiting the bounds of elements of T, redefining the problem space and ensuring limited bounds and more predictable behavior. These, along with the clipping from MMACO, relate to concepts in RL focused on maintaining exploration, such as entropy regularization and experience replay [44].

Some ACO models have drawn directly from the inspiration of RL algorithms. The first explicit connection between ACO and RL was Ant-Q [22] which employed the same update rule used in Q-learning [44]. Many models, such as Ant Colony System (ACS) [14, 15, 21], Entropy-based Dynamic Heterogeneous ACO (EDHACO) [10], and greedy-Levy ACO [30], attempt to tackle the exploration - exploitation problem by modifying the forward step search rule in ways also seen in RL. There are two models we are aware of, Ant Temporal Difference (AntTD) [36] and Label Section-Ant RL (LS-AntRL) [35], that directly invoke TD methods to update a value function.

ADaptive ACO (ADACO) [47, 48] addresses the AS update rule as Policy Gradient, building on the ACO with Stochastic Gradient Descent (ACOSGD) and ACO with Cross-Entropy (CEACO) algorithms [18, 31]. ADACO applied the ADADELTA algorithm to the update rule in AS to create a dynamic evaporation rate. While a connection between ACO and PG can be seen in [47, 48], they implement a gradient approximation in place of the exact calculation.

3 NOTATIONAL CONVENTIONS USED

Both RL and ACO have developed their distinct terminologies, creating ambiguities when integrating the two. For clarity, we present the notation used going forward, which aligns with the conventions in optimization and RL literature. We denote the parameterization of the learned pheromone table by θ, replacing T. For an edge $\langle i, j \rangle$, with pheromone, τ_{ij}, we define $\tau_{ij} \equiv \theta_{ij}$. Given a path through the graph, we define a trace $\tau \equiv \langle s_0, a_0, s_1, a_1, \cdots, a_{T-1}, s_T \rangle$, where each state, $s_i \in \mathcal{S}$, corresponds to a node of the construction graph and each action, $a_i \in \mathcal{A}$, to an edge. Since the full construction history, $\tau_{0:t}$, is embedded in the current state, s_t, the construction graph serves as a bridge between the TSP solutions and MDPs As such, we note that $\pi(a_t \mid \tau_{0:t}\,; \theta) = \pi(a_t \mid s_t\,; \theta)$ when we utilize the construction graph. We reuse the evaporation rate, ρ, to denote the learning rate, drawing an intentional analogy between the two terms.[1] The distance between two nodes connected by edge a is denoted d_a, and the path length of the path (trace) is $L(\tau)$. The neighborhood at state, s_t is the set of outgoing edges not yet in the current trace:

$$\mathcal{N}(s_t) \equiv \{a_i \mid a_i \text{ originates from } s_t \text{ and } a_i \notin \tau_{0:t}\}.$$

At iteration k, the set of samples collected in that iteration is $\mathcal{D}_k$, where the size of this set represents the number of ants, $|\mathcal{D}_k| = m$. The replay buffer, which stores persistent samples across iterations, is denoted $\mathcal{D}$, with size $|\mathcal{D}|$. A generalized measure of the utility of an action during the solution construction is $\Psi_t(a_t \mid \tau_{0:t})$, which is equivalent to $\Psi(a_t \mid s_t)$ in the homogeneous-Markov process case. To allow for the application of PG, we assert that Ψ is independent of θ.

To simplify notation, we define $\hbar_{ij} \equiv h_{ij}^{\beta}$, and $\pi_\theta \equiv \pi(\cdot \mid \cdot\,; \theta)$. The vector of parameters that contribute to the decision at step t, given $\tau_{0:t}$, is $\vec{\theta}_{\tau_{0:t}}$, where entries corresponding to edges already in $\tau_{0:t}$ or edges not originating from current state, s_t, are zero. When the history is clear from context, we omit the subscript for brevity. We define the operation $\vec{\theta} \cdot \vec{\hbar}$ as the element-wise (Hadamard) product of the vectors. Similarly, $\vec{\theta}^{\alpha}$ and $\vec{\theta}^{-1}$ refer to the vectors of component-wise exponentiation and reciprocal values, respectively. A stochastic decision made from the policy is denoted $a_t \sim \pi(s_t)$, and a trace drawn from the policy, starting at a random node, as $\tau \sim \pi$. The expectation of a quantity with respect to a trace randomly drawn from π is expressed as $\mathbb{E}_{\tau \sim \pi}[\cdot]$.

4 METHODS

Drawing on the inspiration of previous ACO and RL methods, we present an algorithm that incorporates a collection of select method from both. To address variance in parameter updates, we implement a robust Policy Gradient update implementing experience replay, importance sampling, an advantage reward function, and a PPO gradient clipping function.

[1] See the appendix in [47] for a derivation of pheromone evaporation from gradient ascent with an L2 regularizer.

4.1 Replay Buffer

Bringing previous generation's information into future generations is a common concept to PBIL and is especial familiar to ACO in the form of the Elitist Ant strategy [3, 17]. However, existing approaches often keep a static list either of the most recent or only the best samples. We draw on the success of experience replay and implement a stashing and resampling method for ACO [19, 32]. Instead of applying the current generation's sampled paths, they are added to a collection of paths, which is then sampled to calculate the batch gradient. The experiences in the replay buffer can be added from the model, as we will do here, or provided to the model to provide guided examples to learn from.

Experience replay should yield more diversity and, by extension, more exploration, while not substantially increasing the runtime of the algorithm [19, 44]. In the application of ACO in domains where collection is costly, experience replay permits more efficient usage of the collected samples by having the chance to revisit them before discarding. For the bulk of models, we can apply experience replay directly with no additional modifications to the backward step. Especially in the case that the backward step is dependent on the parameters themselves, off-policy methods need to be considered. We address this in Section 4.2.

For each generation, we generate the ants as is done in the typical forward step of AS. These samples are added to a collection of stored experiences, and then pruned. To help promote randomness in the search, samples used in the backwards step are drawn i.i.d. with replacement u times from a uniform distribution of the replay buffer, $e \sim \text{Unif}(\mathcal{D})$. We implemented two pruning rules to remove pruned samples from the replay buffer:

- Evict: We keep the $|\mathcal{D}|$ most recent samples by using a queue structure, adding m samples, and removing the oldest m samples at every step.
- Elitist: We keep only the $|\mathcal{D}|$ best ants in the buffer.

Throughout our preliminary testing, the Elitist strategy outperformed the Evict strategy, so we used the Elitist rule for our experiments.

The replay buffer can seen as a generalization of previous elitist-ACO algorithms. MMACO, Rank-AS, κ-best ACO, and BWAS can be seen as instances of replay buffers applied to ACO. In MMACO and BWAS the Elitist strategy is invoked with $|\mathcal{D}| = 1$, with BWAS including a Elitist update for both the best and the worst sampled. Rank-AS and κ-best ACO are also Elitist updates with a fixed buffer size and the whole buffer being used for updates. The weighting methods used in these two algorithms could be considered similar to the advantage function seen in the PG update rule, where better samples are given more weight. Finally, AS can be seen as a specific instance of the Evict strategy, where the population size is the same as the buffer size. By generalizing the idea of replay buffers for ACO, we open up the potential for rapid development of sampling methods to make better efficiency of the sampled data.

4.2 Importance Sampling

One of the assumptions made in deriving Equation (1), is that samples used to update the parameters were sampled from our policy [46]. By utilizing a replay buffer, we are now generating updates according to a different policy, $\mu \neq \pi_{\theta_t}$. This problem is called off-policy learning, as we are learning from experience that are not in line with our current policy (the alternative, where we learn from $\tau \sim \pi_{\theta_t}$, is called on-policy learning). In [12], the derivation for policy gradient for experiences sampled from μ is provided as:

$$\nabla_\theta J(\theta_t) = \mathop{\mathbb{E}}_{a_t \sim \mu}\left[\frac{\pi_{\theta_t}(a_t \mid s_t)}{\mu(a_t \mid s_t)} \Psi(a_t \mid s_t) \nabla_\theta \log \pi_{\theta_t}(a_t \mid s_t)\right]. \quad (3)$$

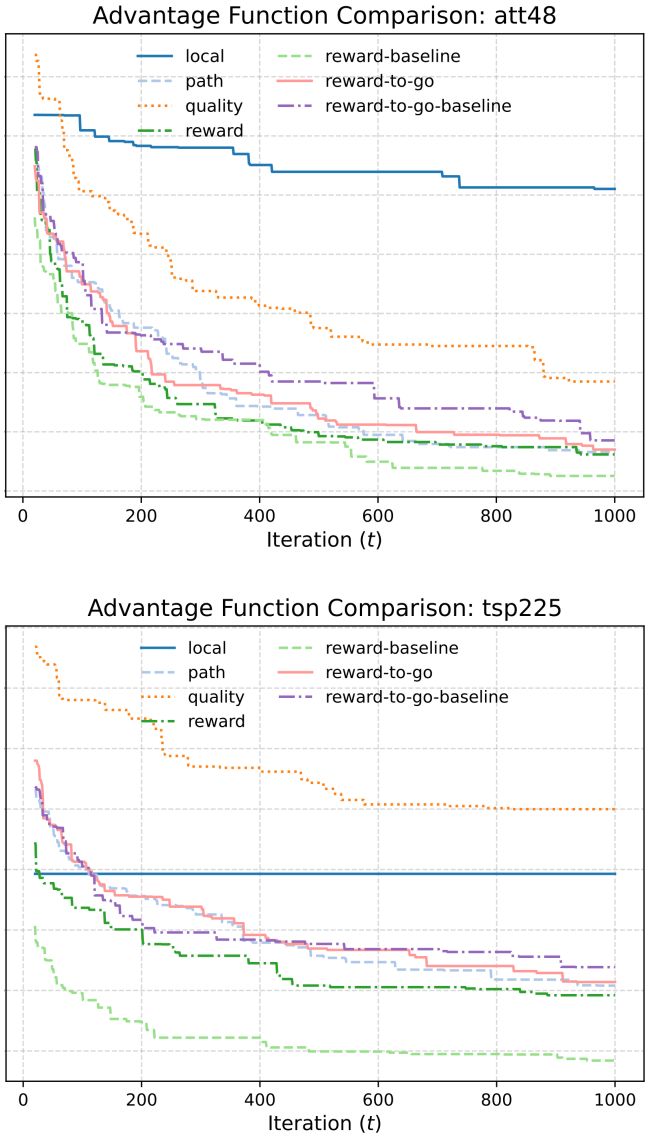

Figure 1: Average performance of different advantage functions over five runs with 1000 iterations. Only iterations after step 20 are shown to highlight the behavior after the initial burn-in period.

This added term, π/μ, is called importance sampling and serves to weight updates according to how relevant to the current policy they are. This yields the same update rule as seen in [18], but with an added importance sampling term. It is generally seen that on-policy methods learn more conservative policies and avoid negative rewards during training, while off-policy methods coverage faster and explore the space more, tending to include more costly moves [44].

4.3 Clipping

To mitigate extreme policy updates, SGD methods often constrain step magnitudes, typically through a tuned learning rate, ρ, or by allowing ρ to be adaptive. While this helps control overall stability, it does not prevent occasional poor updates due to stochasticity [39]. PPO address this by directly limiting the influence of policy updates that diverge significantly from the current policy. The PPO clipping function is:

$$J(\theta \mid s_t, a_t, \mu) = \min_{\theta} \left\{ \frac{\pi_\theta}{\mu} \Psi(a_t \mid s_t),\ \text{clip}\left(\frac{\pi_\theta}{\mu},\ \varepsilon\right) \Psi(a_t \mid s) \right\},$$

where clip restricts the importance sampling ratio to between $(1+\varepsilon)$ and $(1-\varepsilon)$, preventing overly large updates. This approach improves upon earlier methods by being simpler to implement and more robust in practice [20], making it a natural choice for our broader argument about synergy between ACO and modern RL. Given that ACO updates pheromones in batches, we apply clipping per sample rather than to the averaged gradient, ensuring stable reinforcement at the individual sample level.

4.4 Advantage

Finally, recall that Equation (3) refers to Ψ, and that we have a collection of choices on how to define this. In classic ACO, the backward step uses the aggregated cost of the path, $\Psi(\tau_k) = 1/L(\tau_k)$, for the ant's walk, τ_k. Drawing from RL, we consider a variety of alternative advantage algorithms.

- **Quality**: The pheromone weight of the action. This update is the term in the original derivation for Equation (1) [46].
- **Local advantage**: The difference between a pheromone and all pheromones in its neighborhood:

$$\Psi_{\text{local}}(a_t \mid s_t) = \frac{1}{\theta_{a_t}} - \frac{|\mathcal{N}(s_t)|}{\sum_{a' \in \mathcal{N}(s_t)} \theta_{a'}}.$$

- **Path advantage**: The difference between the whole path's cost and the average cost of all path's in the replay buffer:

$$\Psi_{\text{path}}(\tau_k) = \frac{1}{L(\tau_k)} - \frac{|\mathcal{D}|}{\sum_{\tau_e \in \mathcal{D}} L(\tau_e)}.$$

- **Reward**: The typical AS update of $1/L(\tau_k)$.
- **Baselined reward**: The reward from the edge minus the average of the neighborhood:

$$\Psi_{\text{base}}(a_t \mid s_t) = \frac{1}{d_{a_t}} - \frac{|\mathcal{N}(s_t)|}{\sum_{a' \in \mathcal{N}(s_t)} d_{a'}}.$$

  The baselined reward is argued to be a better function for Ψ_t and used in REINFORCE to help reduce the variance in policy updates [44, 46].

Algorithm 1 PPOACO

```
initialize_ants()
while not_terminate() do
    D_k ← forward_step()
    D ← Elitist_buffer_update(D_k)
    grad_list ← {}
    for 1..u do
        e ~ Unif(D)
        grad_e ← clipped_gradient(e)
        grad_list.add(grad_e)
    end for
    grad ← avg (grad_list)
    θ ← θ − ρ · grad
    MaxMin(T)
end while
```

- **Reward-to-go**: The average of the reward from the next n steps:

$$\Psi_{\text{togo}}(a_t \mid s_t) = \sum_{i=0}^{n} \frac{1}{d_{a_{t+i}}}.$$

  This reward takes into account not just the current step, but what the following finite trace rewards.
- **Baselined reward-to-go**: The application of both baseline and reward-to-go:

$$\Psi_{\text{btg}}(a_t \mid s_t) = \frac{1}{d_{a_t}} + G_{\text{togo}}(a_t \mid s_t) - G_{\text{base}}(a_t \mid s_t).$$

To determine which advantage function is the best choice, we ran preliminary tests over TSP problems with the different advantage functions. From Figure 1, the baselined reward showed the best performance, so we ran all experiments for the PG-based ACO algorithms with it.

4.5 Algorithm

The previous elements come together to form PPOACO, given in Algorithm 1. When we omit the clipping function, we call that algorithm PGACO. For the sampled action, $a_t \sim \mu(a_t \mid s_t)$, and the policy, π, from Equation (2), the gradient update becomes

$$\begin{aligned}
\nabla_\theta J &= \mathop{\mathbb{E}}_{\tau \sim \mu}\left[\frac{\pi_{\theta_t}(a_t \mid s_t)}{\mu(a_t \mid s_t)} \Psi(a_t \mid s_t) \nabla_\theta \log \pi_{\theta_t}(a_t \mid s_t) \right] \\
&\approx r\, \Psi \nabla_\theta \log \frac{\theta_{a_t}^\alpha \hbar_{a_t}}{\sum_{a' \in \mathcal{N}(s_t)} \theta_{a'}^\alpha \hbar_{a'}} \\
&= r\, \Psi \left[\nabla_\theta \log \theta_{a_t}^\alpha \hbar_{a_t} - \nabla_\theta \log \sum \theta_{a'}^\alpha \hbar_{a'} \right] \\
&= r\, \Psi \left[\nabla_\theta \log \theta_{a_t}^\alpha + \cancelto{0}{\nabla_\theta \hbar_{a_t}} - \nabla_\theta \log \sum \theta_{a'}^\alpha \hbar_{a'} \right] \\
&= r\, \Psi \left[\alpha\, \theta_{a_t}^{-1} \hat{e}_{a_t} - \alpha\, \vec{\pi} \cdot \vec{\theta}^{-1} \right] \\
&= -\alpha\, r\, \Psi \left[\vec{\pi} - \hat{e}_{a_t} \right] \cdot \vec{\theta}^{-1},
\end{aligned}$$

where $r \equiv \frac{\pi_{\theta_t}(a_t|s_t)}{\mu(a_t|s_t)}$ is the importance sampling ratio, and $\hat{e}_{a_t}$ is the unit vector for the action. We typically approximate the expectation by taking the average of a stochastic search over the space, which is why we dropping the expectation in the second line. This calculation closely resembles the results of previous SGD applications to ACO [18, 47]. For each element in the replay buffer, we only need to store the path, the advantage function, and the policy value of the observed action. If the query table for π is pre-calculated, the calculation of the gradient runs in $O(u \cdot n)$ time. The space required for the replay buffer is $O(|\mathcal{D}| \cdot n)$.

5 EXPERIMENTS

We ran two different experiments: an ablation study and a comparative performance study. All tests were run over a collection of 2D Euclidean TSP problems from the problem-set TSPLib, ranging from small to medium size [38]. The parameters of $\alpha = 1$, $\beta = 2$, $m = 10$, were selected following trends in previous literature [13, 47]. The parameters, ρ, $|\mathcal{D}|$, u, and ε were tuned with Bayesian optimization [2]. The algorithm selected $\rho \approx 0.2$ for AS and MaxMin-ACO, and $\rho \approx 3$ for PGACO and PPOACO. The replay size and update size were both tuned to 20. The clipping bound in PPOACO, ε, was tuned to 0.2. The performance of all algorithms was measured by the average best solution observed during training (measured by the length of the final path found).

5.1 Datasets

To test the performance of PGACO and PPOACO, we utilized the TSPLib dataset of TSP graphs. Developed by Gerhard Reinelt, the datasets serve as benchmarks for researchers to test and compare the performance of various optimization algorithms. The library encompasses a diverse collection of problem instances sourced from various origins and of different types, facilitating the evaluation of algorithmic efficiency across a broad spectrum of scenarios. At this point, all graphs in the TSP set have optimal values calculated.

The problem instances in TSPLIB vary in size and complexity, and we utilized eight small to medium sized problems. We chose the following graphs, each with known optimal tour lengths:

- **att48**: 48 U.S. state capitals, optimal tour length = 10,628.
- **pr76**: 76-city problem represented in a 2D Euclidean space, optimal tour length = 108,159.
- **rd100**: 100 random cities in 2D Euclidean space, optimal tour length = 7,910.
- **tsp225**: 225-city problem in 2D Euclidean space, optimal tour length = 3,916.
- **pcb442**: 442-node drilling problem in 2D Euclidean space, optimal tour length = 50,778.
- **att532**: 532-city problem, optimal tour length = 27,686.
- **ali535**: 535 airports worldwide represented as geographical locations, optimal tour length = 202,339.
- **pr2392**: 2,392-city problem in 2D Euclidean space, optimal tour length = 378,032.

All eight of these problems were used for a comparative study, the smaller four being used for an ablation study. We chose these graphs to observe both problem size and distance magnitude's effects on ACO.

Table 1: The average best solution over five runs, with att48, pr76, and rd100 run for 1000 iterations and tsp225 for 2000.

Problem	Buffer Size	AS	ADACO	PPOACO
att48	None	11825	11376	11054
	Small	11682	11153	10862
	Large	11769	11150	10741
pr76	None	110460	126981	120233
	Small	110272	124162	116578
	Large	109775	123438	116878
rd100	None	8950	8584	8418
	Small	8984	8391	8211
	Large	8890	8304	8225
tsp225	None	4436	4398	4257
	Small	4428	4300	4183
	Large	4407	4211	4136

Table 2: P-values of the two-sided Wilcoxon signed-rank tests. "None" is shorthanded to "N", likewise for "Small" being "S" and "Large being "L." Significant as possible values are shaded.

Problem	p-value	AS	ADACO	PPOACO
att48	N vs. S	0.062	0.063	0.063
	N vs. L	0.125	0.063	0.063
	S vs. L	0.063	1.0	0.813
pr76	N vs. S	0.813	0.063	0.063
	N vs. L	0.125	0.063	0.063
	S vs. L	0.063	0.313	0.625
rd100	N vs. S	0.313	0.063	0.063
	N vs. L	0.313	0.063	0.063
	S vs. L	0.125	0.063	0.313
tsp225	N vs. S	0.063	0.063	0.063
	N vs. L	0.125	0.063	0.063
	S vs. L	1.0	0.063	0.063

5.2 Ablation Study

To test whether the replay buffer affected the performance of the algorithm, we ran an ablation study with the replay size under test. We chose to focus on three algorithms, AS, ADACO, and PPOACO, to target the simplest algorithm, an on-policy update, and an off-policy update, respectively. The replay buffer size was allowed to take on three values

(1) **None**: The replay buffer was disabled; only the ants sampled that iteration updated the parameter values.
(2) **Small**: The replay buffer was set to 20. This sampled a constant, conservative number of ants.
(3) **Large**: The replay buffer was set to the dimension of the problem, n.

As argued in [34], ACO should not require an abundant number of ants, motivating a smaller replayer buffer. In the experience replay literature, a more extensive replay buffer tends to perform better [19] and ADACO sets $m = n$ to large success [47]. The number of sampled elements from the replay buffer was fixed to 20 for both replay rules, and the batch gradients were averaged at each

iteration to prevent the application of experience replay to be a proxy for ρ. Each algorithm was run for 1,000 iterations, five times. The percentage improvement from no experience replay is reported in Table 1.

Across all algorithms and problems, the replay size had a significant impact on the performance. The advantage was more evident in the two gradient-based methods. It appears that the replay buffer should be much larger in the case of AS and ADACO, with a growing importance on the size as problems get larger. For ADACO, the superiority of a larger replay buffer has a growing gap between the two sizes as the problem sizes get larger and show a larger ability to exploit the benefits of experience replay. PPOACO saw a unilateral advantage from the replay buffer but did not see a clear relationship between buffer size and performance conditioned on problem size. These results provide an affirmation that the inclusion of experience replay is largely beneficial to AS.

To validate these results, we ran a two-sided Wilcoxon signed-rank test to test the whether adding a replay buffer affected the performance of these algorithms to a statistically signifcant degree. To control for seeding in the tests, permutations in the order of tests were not allowed. The p-values can be seen in Table 1, and it should be noted that the lowest possible p-value for the test on this size is 0.063. We can see, with few exceptions, that including a replay buffer seemed to shift the average performance of the algorithms to a statistically significant degree, always improving the performance. The size of the replay buffer seems to have had a much smaller effect on the performance, opening up a possible choice as to what size to implement.

For the Elitist rule, at every time step the algorithm has to iterate over all elements of both $\mathcal{D}$ and the sampled data points to find the batch of worst performing samples; this process runs in $O(|\mathcal{D}|+m)$. The storage cost for the buffer is $O(|\mathcal{D}|\cdot n)$. In the case of the large buffer, $|\mathcal{D}| = n$ and $O(n^2)$, which is both the size of the problem and the driving term in the space complexity of ACO. In the case of runtime, this addition is not the driving term and matches the space complexity of the larger problem. Theoretically the best performing buffer size should be used until $|\mathcal{D}| \gg n$. Because of the comparative performance of the small and large buffers in PPOACO and with practical runtime considerations, we employed a small ($|\mathcal{D}| = 20$), Elitist replay buffer to PGACO and PPOACO.

5.3 Comparative Analysis

We tested both PGACO and PPOACO against the traditional ACO algorithms, AS and MMACO, and against its gradient peers, ACOSGD and ADACO. All alternative algorithms were tested as outlined in their respective sources, using their hyperparameters, outside of what was defined earlier, without modifying their update functions. For PGACO and PPOACO, we employed an elitist replay buffer strategy with $|\mathcal{D}| = 20$ and $u = 20$. The algorithms were run across the eight TSP problems from TSPLib described in Section 5.1 ten times. The average of the best path found can be seen in Table 3. Each run was seeded, ensuring seeds did not overlap with the ablation study's seeds, and controlling for random staring points across algorithms.

Across all the problems, PGACO and PPOACO found better solutions than all alternatives. This margin decreased as the problems got larger, with att532 showing no improvement in PGACO compared to ADACO. On this same dataset, however, PPOACO had a large improvement over all other algorithms. The number of iterations given to the algorithm did not appear to be the primary factor driving performance differences. The tsp225 instance, which was run with the same number of iterations as the larger problems, exhibited a substantial performance increase between PGACO and the other algorithms. In contrast, the larger problems showed only a modest improvement, suggesting that other factors contributed to the observed differences in performance.

To validate qualitative results, we ran a one-sided Wilcoxon signed-rank test to test the alternative hypothesis that our model score was less than the comparative model, on average. The p-values can be seen in Table 2. In every test, PGACO and PPOACO had a ~ 0.001 p-value to provide a lower performance average than the compared models. We also saw that PPOACO had a p-value of ~ 0.001 to perform better than PGACO on all datasets, except att48 and rd100, in which case the p-values were ~ 0.042 and ~ 0.053, respectively. Therefore, we assert that there is evidence that PGACO and PPOACO achieve lower objective values on 2D Euclidean TSP problems than the four alternatives tested. Further, there is evidence that the clipping function significantly improved the results as well.

6 DISCUSSION

As shown in Figure 1, the baselined reward function provided improvement compared to the typical reward function. This improvement may be due to parameters' magnitudes being dependent on the problem-specific distances. Algorithms such as Hypercube-ACO and GBAS mitigate this issue by constraining the parameter bounds, requiring substantial modifications to the backward and merge steps of ACO. The baseline function offsets the reward by a mean of the neighborhood, requiring only minor modifications to existing ACO functions. The PPO Clipping function helps stabilize variance during learning by preventing extreme parameter updates until they have reached a magnitude where large changes are less disruptive.

The ablation study (Table 1) indicates that a replay buffer improved the ACO performance, with statistical significance on many of the datasets. For ADACO, larger replay buffer sizes led to significant performance improvements, whereas PPOACO showed no statistically significant relationship between buffer size and performance. Our analysis of experience replay focused on quantitative behavior and a limited set of buffer sizes. Future work could explore a broader range of values, alternative sampling distributions over D, and employing more advanced replay buffer strategies. Our findings motivate further exploration of experience replay variations to enhance ACO performance.

In the larger comparison study, PPOACO, followed by PGACO to a lesser extent, outperformed all competitors examined. We attribute PGACO's strong performance to a careful consideration of its fundamental design principles. By casting the algorithm explicitly into the ACO forward/backward/merge framework with a PG update, modifications were straightforward. As a population-based RL algorithm, ACO benefits from stressing this unified framework that facilitates integration with modern RL methods and application-driven refinements.

Table 3: Average best solution of ACO models over ten runs.

Problem	Iters	AS	MaxMin-AS	ACOSGD	ADACO	PGACO	PPOACO
pr2392	9000	525362	527043	499671	500144	491681	**485854**
ali535	5000	272290	271228	259683	261126	257014	**253365**
att532	5000	35610	35410	33961	33500	33498	**33131**
pcb422	5000	56788	56918	54096	54539	53102	**52506**
tsp225	5000	4436	4296	4117	4147	4022	**4003**
rd100	2000	8967	8682	8321	8383	8131	**8092**
pr76	2000	120250	117219	111604	112986	109884	**109162**
att48	1000	11720	11434	10981	11089	10856	**10815**

In [47], the pheromone evaporation rule in AS is shown to be an L2 regularizer. Initially introduced to prevent parameter runaway, evaporation has since been adopted by most ACO models. In contrast, entropy regularization is a common RL technique for mitigating early convergence by preventing premature parameter reduction [44]. The effectiveness of entropy regularization in other RL models suggests that the L2 regularization should be analyzed alongside alternative parameter-bounding strategies.

To our knowledge, the theoretical guarantees for PG have not been formally established within the context of ACO. Numerous RL theorems apply universally to PG methods, and our connections in this paper motivate a more formal discussion of what PG can tell us about ACO behavior.

RL is a field with broad applications to real-world problems. With the connection between ACO and RL, we have opened up many practical solutions to problems that arise. Replay buffers with importance sampling allow algorithms to reuse data from previous samples reliably, providing both sample efficiency and more exploration in the search [19, 44]. We see practical application of this principle in the world of swarm robotics or network routing, where data collection may be costly and sparse, but computation may be plentiful. By stashing and resampling observations, these algorithms can continue to learn, adding irregular updates to the buffer whenever they come in. With the connection to SGD directly made through policy gradients, custom loss function, regularizers, and solvers can be applied according to domain-knowledge.

7 CONCLUSION

This work demonstrates that integrating reinforcement learning principles into Ant Colony Optimization can yield significant performance improvements. By incorporating policy gradient updates and experience replay, we have enhanced ACO's adaptability while maintaining its core strengths. Our results show that these modifications enable more effective solution search, particularly on medium-scale Traveling Salesperson Problem instances. More broadly, this study highlights the potential of unifying population-based optimization with reinforcement learning, paving the way for future advancements in learning-driven metaheuristics.

While our evaluation focused on medium-sized TSP instances, the success of our approach suggests promising directions for larger and more complex combinatorial optimization problems. Future work will explore scalability to high-dimensional problems, alternative problem domains beyond TSP, and a deeper sensitivity analysis to refine hyperparameter selection. Additionally, further investigation into the interplay between ACO enhancements and reinforcement learning techniques may yield even greater improvements. By building upon these foundations, we can push the boundaries of learning-augmented optimization and unlock new possibilities for adaptive, high-performance metaheuristics.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their comments that have lead to a tighter presentation of our work. We also thank members of the Numerical Intelligent Systems Laboratory at Montana State University for their support and guidance through this project. This material is based upon work supported in part by the National Science Foundation EPSCoR Cooperative Agreement OIA-2242802. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Jacob Adkins, Michael Bowling, and Adam White. 2024. A Method for Evaluating Hyperparameter Sensitivity in Reinforcement Learning. https://doi.org/10.48550/arXiv.2412.07165 arXiv:2412.07165

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-Generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (Kdd '19)*. Association for Computing Machinery, Anchorage, AK, USA, 2623–2631. https://doi.org/10.1145/3292500.3330701

[3] Shumeet Baluja and Rich Caruana. 1995. Removing the Genetics from the Standard Genetic Algorithm. In *Proceedings of the Twelfth International Conference on Machine Learning*, Armand Prieditis and Stuart Russell (Eds.). Morgan Kaufmann, Tahoe City, CA, 38–46. https://doi.org/10.1016/B978-1-55860-377-6.50014-1

[4] Mauro Birattari, Gianni Di Caro, and Marco Dorigo. 2002. Toward the Formal Foundation of Ant Programming. In *ANTS*. Springer Berlin Heidelberg, Berlin, Heidelberg, 188–201. https://doi.org/10.1007/3-540-45724-0_16

[5] Christian Blum. 2005. Ant Colony Optimization: Introduction and Recent Trends. *Physics of Life reviews* 2, 4 (2005), 353–373.

[6] Christian Blum. 2024. Ant Colony Optimization: A Bibliometric Review. *Physics of Life Reviews* 51 (Dec. 2024), 87–95. https://doi.org/10.1016/j.plrev.2024.09.014

[7] Christian Blum, Andrea Roli, and Marco Dorigo. 2001. HC–ACO: The Hyper-Cube Framework for Ant Colony Optimization. In *MIC'2001*, Vol. 4. IRIDIA, Porto, Portugal, 399–403.

[8] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. 1999. A New Rank Based Version of the Ant System–a Computational Study. *Central European Journal of Operations Research* 7, 1 (1999).

[9] M. Chandana and Sanjeev Thakur. 2016. Ant-Net: An Adaptive Routing Algorithm. In *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*. IEEE, 1–4. https://doi.org/10.1109/ICPEICES.2016.7853616

[10] Jia Chen, Xiao-Ming You, Sheng Liu, and Juan Li. 2019. Entropy-Based Dynamic Heterogeneous Ant Colony Optimization. *IEEE access : practical innovations, open solutions* 7 (2019), 56317–56328.

[11] Oscar Cordon, Iñaki Viana, Francisco Herrera, and Llanos Moreno. 2000. A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System. *Proceedings of Ants'2000* (Aug. 2000), 22–29.

[12] Thomas Degris, Martha White, and Richard S. Sutton. 2013. Off-Policy Actor-Critic. https://doi.org/10.48550/arXiv.1205.4839 arXiv:1205.4839

[13] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. 2006. Ant Colony Optimization. In *IEEE Computational Intelligence Magazine*, Vol. 1. IEEE, 28–39. https://doi.org/10.1109/MCI.2006.329691

[14] M. Dorigo and L.M. Gambardella. 1997. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 53–66. https://doi.org/10.1109/4235.585892

[15] Marco Dorigo and Luca Maria Gambardella. 1997-July. Ant Colonies for the Travelling Salesman Problem. *Bio Systems* 43, 2 (1997-July), 73–81. https://doi.org/10.1016/S0303-2647(97)01708-5

[16] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. 1991. Positive Feedback as a Search Strategy. *Technical Report* (June 1991).

[17] M. Dorigo, V. Maniezzo, and A. Colorni. 1996. Ant System: Optimization by a Colony of Cooperating Agents. In *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 26. IEEE, 29–41. https://doi.org/10.1109/3477.484436

[18] Marco Dorigo, Mark Zlochin, Nicolas Meuleau, and Mauro Birattari. 2002. Updating ACO Pheromones Using Stochastic Gradient Ascent and Cross-Entropy Methods. In *Applications of Evolutionary Computing: EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN Kinsale, Ireland, April 3–4, 2002 Proceedings*. Springer, 21–30.

[19] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. 2020. Revisiting Fundamentals of Experience Replay. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 3061–3071.

[20] Neil De La Fuente and Daniel A. Vidal Guerra. 2024. A Comparative Study of Deep Reinforcement Learning Models: DQN vs PPO vs A2C. https://doi.org/10.48550/arXiv.2407.14151 arXiv:2407.14151

[21] L.M. Gambardella and M. Dorigo. 1996. Solving Symmetric and Asymmetric TSPs by Ant Colonies. In *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE, 622–627. https://doi.org/10.1109/ICEC.1996.542672

[22] Luca M. Gambardella and Marco Dorigo. 1995-July. Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem. In *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann, San Francisco (CA), 252–260. https://doi.org/10.1016/B978-1-55860-377-6.50039-6

[23] Fred Glover and Gary A. Kochenberher. 2003. *Handbook of Meta-Heuristics*. INTERNATIONAL SERIES IN OPERATIONS RESEARCH & MANAGEMENT SCIENCE, Vol. 57. KLUWER ACADEMIC PUBLISHERS, Dordrecht.

[24] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. 2004. Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. *Journal of Machine Learning Research* 5, 9 (2004).

[25] Walter J. Gutjahr. 2000. A Graph-based Ant System and Its Convergence. *Future Generation Computer Systems* 16, 8 (June 2000), 873–888. https://doi.org/10.1016/S0167-739X(00)00044-3

[26] Walter J. Gutjahr. 2003. A Generalized Convergence Result for the Graph-Based Ant System Metaheuristic. *Probability in the Engineering and Informational Sciences* 17, 4 (Oct. 2003), 545–569. https://doi.org/10.1017/S0269964803174086

[27] Poul E. Heegaard and Werner Sandmann. 2007. Efficient Estimation of Loss Rates in Optical Packet Switched Networks with Wavelength Conversion. In *2007 Second International Conference on Systems and Networks Communications (ICSNC 2007)*. IEEE, 59–59. https://doi.org/10.1109/ICSNC.2007.32

[28] Nikola Ivković, Robert Kudelić, and Marin Golub. 2023. Adjustable Pheromone Reinforcement Strategies for Problems with Efficient Heuristic Information. *Algorithms* 16, 5 (2023). https://doi.org/10.3390/a16050251

[29] Long-Ji Lin. 1992. Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning* 8, 3 (May 1992), 293–321. https://doi.org/10.1007/BF00992699

[30] Yahui Liu, Buyang Cao, and Hehua Li. 2021. Improving Ant Colony Optimization Algorithm with Epsilon Greedy and Levy Flight. *Complex & Intelligent Systems* 7, 4 (2021), 1711–1722.

[31] Nicolas Meuleau and Marco Dorigo. 2002. Ant Colony Optimization and Stochastic Gradient Descent. *Artificial Life* 8, 2 (April 2002), 103–121. https://doi.org/10.1162/10645460232018420 2

[32] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. https://doi.org/10.48550/arXiv.1602.01783 arXiv:1602.01783

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-Level Control through Deep Reinforcement Learning. *Nature* 518, 7540 (Feb. 2015), 529–533. https://doi.org/10.1038/nature14236

[34] Frank Neumann, Dirk Sudholt, and Carsten Witt. 2010. A Few Ants Are Enough: ACO with Iteration-Best Update. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10)*. Association for Computing Machinery, New York, NY, USA, 63–70. https://doi.org/10.1145/1830483.1830493

[35] Yuchen Pan, Yulin Xue, Jun Li, and Jianhua Xu. 2024. Label Selection Algorithm Based on Ant Colony Optimization and Reinforcement Learning for Multi-label Classification. In *Neural Information Processing*, Biao Luo, Long Cheng, Zheng-Guang Wu, Hongyi Li, and Chaojie Li (Eds.). Springer Nature, Singapore, 509–521. https://doi.org/10.1007/978-981-99-8073-4_39

[36] Mohsen Paniri, Mohammad Bagher Dowlatshahi, and Hossein Nezamabadi-pour. 2021. Ant-TD: Ant Colony Optimization plus Temporal Difference Reinforcement Learning for Multi-Label Feature Selection. *Swarm and Evolutionary Computation* 64 (July 2021), 100892. https://doi.org/10.1016/j.swevo.2021.100892

[37] Seid H. Pourtakdoust and Hadi Nobahari. 2004. An Extension of Ant Colony System to Continuous Optimization Problems. In *Ant Colony Optimization and Swarm Intelligence*, Marco Dorigo, Mauro Birattari, Christian Blum, Luca Maria Gambardella, Francesco Mondada, and Thomas Stutzle (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 294–301.

[38] Gerhard Reinelt. 1991. TSPLIB—A Traveling Salesman Problem Library. *ORSA Journal on Computing* 3, 4 (Nov. 1991), 376–384. https://doi.org/10.1287/ijoc.3.4.376

[39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. https://doi.org/10.48550/arXiv.1707.06347 arXiv:1707.06347

[40] Thomas Stutzle and Marco Dorigo. 2002. A Short Convergence Proof for a Class of Ant Colony Optimization Algorithms. *IEEE Transactions on Evolutionary Computation* 6, 4 (Aug. 2002), 358–365. https://doi.org/10.1109/TEVC.2002.802444

[41] Thomas Stutzle and Holger H Hoos. 1996. *Improving the Ant System: A Detailed Report on the MAX–MIN Ant System*. Technical Report AIDA-96-12. FG Intellektik, FB Informatik, TU Darmstadt, Germany.

[42] Thomas Stützle and Holger H. Hoos. 2000. MAX–MIN Ant System. *Future Generation Computer Systems* 16, 8 (2000), 889–914. https://doi.org/10.1016/S0167-739X(00)00043-1

[43] Thomas G. Stutzle. 1998. *Local Search Algorithms for Combinatorial Problems - Analysis, Improvements, and New Applications*. Ph. D. Dissertation. Darmstadt University of Technology.

[44] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). MIT press, Cambridge, Massachusetts.

[45] El-Ghazali Talbi. 2021. Machine Learning into Metaheuristics: A Survey and Taxonomy. *ACM Comput. Surv.* 54, 6 (July 2021), 129:1–129:32. https://doi.org/10.1145/3459664

[46] Ronald J Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine learning* 8 (1992), 229–256.

[47] Yi Zhou, Weidong Li, Xiaomao Wang, Yimin Qiu, and Weiming Shen. 2022. Adaptive Gradient Descent Enabled Ant Colony Optimization for Routing Problems. *Swarm and Evolutionary Computation* 70 (April 2022), 101046. https://doi.org/10.1016/j.swevo.2022.101046

[48] Y. Zhou, W. D. Li, X. Wang, and Q. Qiu. 2021. Enhancing Ant Colony Optimization by Adaptive Gradient Descent. *Data Driven Smart Manufacturing Technologies and Applications* (Feb. 2021), 191–215. https://doi.org/10.1007/978-3-030-66849-5_9