WoLF Ant

Gilbert L. Peterson*, Christopher B. Mayer†, and Kevin Cousin*
*Department of Electrical and Computer Engineering
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45431
e-mail: {gilbert.peterson, kevin.cousin}@afit.edu
†Department of Electrical and Computer Engineering
United States Naval Academy
105 Maryland Avenue
Annapolis, MD 21402
e-mail: cmayer@usna.edu

***Abstract*—Ant colony optimization (ACO) algorithms can generate quality solutions to combinatorial optimization problems. However, like many stochastic algorithms, the quality of solutions worsen as problem sizes grow. In an effort to increase performance, we added the variable step size off-policy hill-climbing algorithm called PDWoLF (Policy Dynamics Win or Learn Fast) to several ant colony algorithms: Ant System, Ant Colony System, Elitist-Ant System, Rank-based Ant System, and Max-Min Ant System. Easily integrated into each ACO algorithm, the PDWoLF component maintains a set of policies separate from the ant colony's pheromone. Similar to pheromone but with different update rules, the PDWoLF policies provide a second estimation of solution quality and guide the construction of solutions. Experiments on large traveling salesman problems (TSPs) show that incorporating PDWoLF with the aforementioned ACO algorithms that do not make use of local optimizations produces shorter tours than the ACO algorithms alone.**

***Index Terms*—Ant Colony Optimization, Ant System, Traveling Salesman Problem, Win or Learn Fast, Gradient Descent.**

I. Introduction

Ant Colony Optimization (ACO) is a meta-heuristic based on the natural behavior of real ants and their seeming ability to solve minimization problems efficiently obeying a few simple rules. ACO algorithms have produced world-class solutions to many NP-Hard optimization problems including the Traveling Salesman Problem (TSP) [1], [2], the Quadratic Assignment Problem [3]–[5], and job-shop scheduling problems [6], [7].

Although variations exist, the majority of ACO implementations employ a graph structure for representing problems. Loosely speaking, vertices represent problem states. Edges run between vertices and contain a substance called *pheromone*. The level of an edge's pheromone represents the learned importance of the edge in regards to connecting its vertices as part of a desirable solution.

The views expressed in this article are those of the authors and do not reflect the official policy or position of the U.S. Air Force, Dept. of Defense, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation here on.

This paper was supported by the Air Force Office of Scientific Research, project number 2311/FX.

Ants construct a solution by traversing the graph's edges as they link the graph's vertices together. To move from one vertex to another, an ant computes a value for each outgoing edge of the current vertex by combining the edge's pheromone concentration and a heuristic desirability specific to the problem being solved. The ant then normalizes the probability of each outgoing edge and picks an edge to traverse using a uniformly generated random number.

After connecting enough vertices together to form a solution, ants deposit pheromone on the edges traversed during solution construction in proportion to the solution's quality. That is, edges critical to the best solutions receive more pheromone than other edges. Periodic pheromone evaporation makes non-reinforced edges less attractive over time, and acts to weed out undesirable solutions.

Making weighted random decisions using both pheromone (representing past good solutions) and heuristics (which guide ants in the absence of pheromone) encourages exploration in the neighborhood of good solutions and allows for enough variability that distant solutions are unlikely to go unnoticed. Excellent explanations of the ACO meta-heuristic can be found in [1], [2], [8]

Unfortunately, ACO has the drawback that solution quality degrades in proportion to the problem size [1]. While this is true of all algorithms, it is particularly troublesome for ACO because interesting problems (non-toy problems) consist of larger search spaces. This means that ACO solutions become generally less competitive compared to other search algorithms as problem sizes increase.

This work improves ACO solutions for larger problems by enhancing ACO with the off-policy learning reinforcement learning technique Policy Dynamics Win or Learn Fast (PDWoLF) [9]. PDWoLF speeds learning by using a variable step size gradient which when the agent is winning, is small and large when it is losing. Five legacy ACO algorithms are updated to include a PDWoLF policy gradient: Ant System (AS) [1], [2], [10], [11], Ant Colony System (ACS) [1], [12], [13], Elitist-Ant System [14] (EAS), Rank-based Ant System (RAS) [15], and Max-Min Ant System (MMAS) [16]. Each legacy algorithm implements the ACO meta-heuristic

The Traveling Salesman Problem

Given: A set of cities, C, a set of edges that connects each city in C to every other city, $E = C \times C$, and, for each edge $(i, j) \in E$ where $i, j \in C$, a weight w_{ij} that represents the cost of traversing the edge.
Find: An ordering of edges, T, (called a tour) that starts and ends with city s such that a walk of T visits each city in C once, except for city s, and that the tour length, L = $\sum_{i,j \in T} w_{ij}$, is minimal among all tours over C.

Fig. 1. A formulation of the Traveling Salesman Problem (TSP).

in form, but each one tweaks or adds something to make itself unique. Due to their individual differences, the five legacy algorithms serve as a varied testbed. With minimal modification of ACOTSP source code available at [17], we integrated PDWoLF into each ACO algorithm.

Experimental evaluation shows that the PDWoLF ACO algorithms generate Traveling Salesman Problem (TSP) solutions with costs that are in many cases statistically lower the legacy ACO algorithms alone. However, because each legacy ACO algorithm differs in how it selects edges for traversal, deposits pheromone, and evaporates pheromone, the results show that the PDWoLF addition functions best when the ACO algorithm updates the trail of all of the ants and does not make use of local optimizations.

A. Paper Focus and Outline

This paper is organized in the following manner. Section II provides background information on the TSP, introduces ACS-TSP, and finally ends with a summary of the policy-based, variable-step size hill-climbing technique called PDWoLF. We point out the algorithmic features and modifications made to the five legacy ACO algorithms in Section III. Section IV analyzes the performance of the modified ACO algorithms. A survey of other hybrid ACO algorithms is contained in V. Finally, section VI concludes by summarizing the significance of these algorithms.

II. Background

A. The Traveling Salesman Problem

Primarily for comparative purposes, we have chosen to implement our two WoLF Ant algorithms in the context of the TSP. The objective of the classic NP-Complete TSP [18] is to find the shortest tour that connects all cities in a set of cities, C. A tour is a closed path that visits every city in C once. The problem's name comes from the idea that a salesman who must visit a number of cities would like to do so while minimizing some travel-related cost such as distance or time. TSP is formally defined in Fig. 1.

B. An Archetypal ACO Algorithm for TSP

Ant System (AS) [1], [2], [10], [11] was the first ACO algorithm for solving the TSP. We consider AS to be the archetypal ACO algorithm for TSP; the ACO algorithms we consider in this paper can be considered variants of AS. This section presents an overview of AS as presented in [1] but modified slightly to match the notation used in this paper.

AS represents the TSP as a graph where vertices are cities and edges connect cities together. In addition to the weight of each edge, w_{ij}, (the distance between cities i and j), each edge also has pheromone level, τ_{ij}, that is the learned desirability that city i should follow city j in a TSP tour.

The algorithm iteratively solves the TSP in computational rounds called *timesteps*. At the start of each timestep AS places m ant agents (the "colony") on the graph, each ant in a random city. The number of ants, m, equals the number of cities in the TSP, n. Each ant, k, walks the graph, constructing a tour, T_k (a solution to the TSP), as it does. The length of ant k's tour, L_k, is the sum of the weight of the edges connecting the cities in the tour.

To select the next city to add to its tour, the ant employs the proportional transition rule of equation (1). The transition rule computes the normalized probability that ant k at city i selects city j as the next city to visit.

$$p_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}, \tag{1}$$

In (1), J_i^k is the set of cities connected by an edge to city i that ant k has not visited so far this tour. η_{ij} is the heuristic desirability of connecting city i to city j. Heuristic desirability helps with edge selection early in the solving process when pheromone is nearly uniform. Usually $\eta_{ij} = 1/w_{ij}$; cities that are closer together are more desirable. The parameters α and β weight pheromone relative to the heuristic desirability. The p_{ij} values for the current city i form a weighted distribution from which the ant randomly picks. The ant then moves from city i to the newly selected city y along edge (i, y) and adds the edge (i, y) to the tour. This process repeats until the ant has visited all cities just once. After visiting the last city, the tour is finalized by adding the edge that connects the last city visited to the first city.

After all m ants have constructed their tours, pheromone is reinforced and evaporated using the pheromone update rule shown in equation (2) below.

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij} \tag{2}$$

where

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{3}$$

and

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{if edge } (i,j) \in T_k \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

In the above equations ρ ($\rho \in \{0, 1\}$) is the pheromone decay factor and Q is a constant set to 100. Note that in AS, each ant in the colony plays a part in updating pheromone.

The deposition and evaporation of pheromone marks the end of one timestep. If a stopping condition has been met at the end of a timestep, AS outputs the best tour found during the trial and that tour's length. The stopping condition for AS was a set number of timesteps. Otherwise, another timestep commences.

Because ACO algorithms are stochastic, many trials, each trial consisting of many timesteps, must execute to have high confidence of producing at least one good solution.

As mentioned earlier, the other four ACO algorithms considered in this paper (and several other ACO algorithms) are variants, to some degree or another, of AS. Minor variations include values for constants and parameters such as the constant Q and the number of ants in the colony, m. Major variations include such things as the number of solutions used to update pheromone, pheromone update rules, and edge selection rules. Section III covers the major differences between AS and the other four legacy algorithms.

C. Policy Dynamics Win or Learn Fast (PDWoLF)

In an effort to overcome the drawback of declining solution quality with large problem instances, we have integrated the variable step-size hill-climbing algorithm PDWoLF [9] into several legacy ACO algorithms. The following reviews off policy-based learning, variable-step hill-climbing, and PDWoLF.

1) Policies and Hill-Climbing: Some agent-based learning algorithms involve the notion of state-action pairs, represented by (s, a) where $s \in S$ and $a \in A$. The set of states, S, is the particular situations/locations/places in which an agent can be (e.g., the states of the TSP are the cities). Actions, A, are the things an agent is allowed to do when in a certain state (e.g., move to another city). The agent must learn what action to perform when it finds itself in some state so as to maximize a reward (or alternately minimize the cost) of the problem being solved or game being played. Generally, this manifests as the assignment of probabilities to each (s, a) pair called a *policy*, π_{sa}. The higher the probability, the more important the action is to the best outcome. A *strategy* is the collective effect of an agent's policies.

An agent employing a policy hill-climbing (PHC) algorithm iteratively adjusts policy values through a value function in response to the reward (or lack of it) received by performing action a when in state s. Picturing the maximum reward as sitting at the top of a hill, the algorithm "climbs the hill" toward the reward. The amount by which policies are adjusted is called the *learning rate* or *step size*, δ.

2) Variable Step Sizes: In general, a PHC algorithm operates using a fixed step size, though doing so is not ideal in several respects. First, it throttles the algorithm. In other words, a fixed step size prevents the algorithm from increasing policies by a large amount (big step size) and, conversely, from making small adjustments (small steps) when warranted. Second, it has been shown that fixed step size hill-climbing algorithms are not convergent in two-player, two-action, iterated matrix games (so-called 2×2 games) [19]. Convergent means that the agent's policies eventually reach a steady state. Singh et al. attempted to overcome this problem through the use of an infinitesimally small step size [19]. While an improvement, it was not convergent in all 2×2 games.

Bowling and Veloso introduced a completely convergent hill-climbing approach called Win or Learn Fat (WoLF) [20]. WoLF uses two step sizes to update policies. Using two step sizes enables WoLF to converge in 2×2 games.

The first step size, δ_w, is associated with the concept of "winning". A policy, or a set of policies, is said to be "winning" when the strategy indicated by the policy or policies by a player gives the player a decided advantage over his or her opponent. The concept of winning is not to be confused with the actual value of the policy itself. Indeed, an (s, a) pair with a low policy value can be winning in the sense that the player has learned not to perform action a when in state s. In WoLF, winning policies are updated using a small step size since they are already desirable and need only minor adjustment.

The other step size, δ_l, is associated with the idea of "losing". Losing is simply the opposite of winning. An agent with a losing policy seeks to improve the policy (change it to a winning policy) as quickly as possible. Hence, using a large step size, where $\delta_l > \delta_w$, rapidly changes the policy value of a losing (s, a) pair. This gives rise to the "learn fast" aspect of WoLF.

3) PDWoLF: The desirable properties of WoLF, led Bowling and Veloso to propose a general purpose algorithm that combined the WoLF concept with a policy hill-climbing variant of Q-learning [21]. They called the resulting algorithm WoLF-PHC (WoLF Policy Hill-Climbing) [20]. In Q-learning, an agent in state s performs an action a which results in a reward. The agent updates the Q-value for the state-action pair, Q_{sa}, based on the reward received. Q-values are similar to, but not exactly the same, as ACO pheromone; both are utility-based performance metrics, but their updating and use in their respective algorithms are slightly different. In addition to the Q-values, WoLF-PHC maintains a policy for each (s, a) pair, π_{sa}, which is adjusted according to a variable learning rate, δ. Modified from [20] to conform with notation used in this paper, the value of δ depends on whether or not the (s, a) pair is winning or losing:

$$\delta = \begin{cases} \delta_w, & \sum_{a'} \pi_{sa'} Q_{sa'} > \sum_{a'} \overline{\pi}_{sa'} Q_{sa'} \\ \delta_l, & \text{otherwise} \end{cases} \tag{5}$$

where a' are the actions available from state s, Q_{sa} is an (s, a) pair's Q-value, and $\overline{\pi}_{sa}$ is the average policy. According to (5), an agent is winning if his current set of policies for state s have a greater benefit ratio than using the average policy of state s [20].

In the Policy Dynamics Win or Learn Fast Policy Hill-Climbing (PDWoLF-PHC) algorithm [9] Banerjee and Peng replaced the notion of average policy contained in equation (5) with one that uses the gradient of the policy. Otherwise, WoLF-PHC and PDWoLF-PHC are identical. The new definition relies on tracking the policy change rate policy velocity), Δ_{sa}, and policy acceleration, Δ^2_{sa}:

$$\delta = \begin{cases} \delta_w, & \Delta_{sa} \cdot \Delta^2_{sa} < 0 \\ \delta_l, & \text{otherwise.} \end{cases} \tag{6}$$

According to (6), the policy for an (s, a) pair is "winning" if the policy value is increasing (positive Δ_{sa}) while the rate of increase is slowing down (negative Δ^2_{sa}) or the policy value is decreasing (negative Δ_{sa}) when the rate is slowing down (positive Δ^2_{sa}). This definition of "winning" captures the notion that policy value change rates should slow down as they near their optimums. Otherwise, the pair is seen as "losing" in the sense that the optimal policy value is still some distance away and requires a large learning rate (large step size) to close the gap.

The PDWoLF definition of winning and losing is based solely on the behavior of the policy values themselves and does not require the extra computation of an average policy. Moreover, PDWoLF-PHC converged faster than WoLF-PHC. For these two reasons, we demonstrate and show results of ACO algorithms that incorporate the PDWoLF approach.

In both WoLF-PHC and PDWoLF-PHC, actions are chosen based on policies "with suitable exploration"; policies replace Q-values in the action selection mechanism [9]. However, Q-values are computed, maintained, and used to update policy values. Indeed, both WoLF-PHC and PDWoLF-PHC use the same technique for updating policy values. The appropriate step size δ is computed by (6) and normalized in (7) to compute the change to the policy, $\Delta\pi_{sa}$ for (s,a) shown in (8). Note that (8) is the only place that Q-values and policies are linked

$$\delta_{sa} = min\left(\pi_{sa}, \frac{\delta}{|A|-1}\right) \tag{7}$$

$$\Delta\pi_{sa} = \begin{cases} -\delta_{sa} & a \neq \text{argmax}_{a' \in A} Q(s,a') \\ \sum_{a' \neq a} \delta_{sa'} & \text{otherwise} \end{cases} \tag{8}$$

Finally, the policy for (s,a) is updated by $\pi_{sa} \leftarrow \pi_{sa} + \Delta\pi_{sa}$ from (8). The changes in policy are similarly updated following the policy update in the order $\Delta^2_{sa} \leftarrow \Delta\pi_{sa} - \Delta_{sa}$ and then $\Delta_{sa} \leftarrow \Delta\pi_{sa}$.

III. PDWoLF Modifications to the Five Legacy ACO Algorithms

Inspired by the success of the WoLF-PHC [20] and PDWoLF-PHC [9] algorithms, we inserted the PDWoLF technique into five ACO algorithms for solving the TSP. This section details the modifications common to all the ACO algorithms in order to incorporate PDWoLF. Then using Ant System (AS) as a baseline (Section II-B), we present the differences between AS and the other four ACO algorithms.

A. Modifications Common to All ACO Algorithms

Changes to each ACO algorithm were limited to the following.

1) The addition of policy values for each edge. For example, π_{ij} is the policy value for the edge connecting city i with city j. Policy values are initialized to $1/n$.
2) The transition rule. Rules vary between algorithms. We give particulars for each algorithm below.
3) The mechanism to update policies. In each modified algorithm, pheromone is updated then policies are updated. Policies are updated according to Algorithm 1. Note that the policy updating process uses equations (9)–(11) which are just the PDWoLF policy modification equations updated to refer to pheromone and a TSP problem in which the set of cities, C, is fully connected.

Algorithm 1 Policy updating in the modified ACO algorithms.

```
1: // $T$ is the best tour found this timestep.
2: for each $(i,j) \in T$ do
3:     calculate policy change $\Delta\pi_{ij}$ by (11)
4:     $\pi_{ij} \leftarrow \pi_{ij} + \Delta\pi_{ij}$
5:     // normalize $\pi_i$
6:     for each $l \in C$ do
7:         $\pi_{il} \leftarrow \frac{\pi_{il}}{\sum_{k \in C} \pi_{ik}}$
8:     $\Delta^2_{ij} \leftarrow \Delta\pi_{ij} - \Delta_{ij}$
9:     $\Delta_{ij} \leftarrow \Delta\pi_{ij}$
```

$$\delta = \begin{cases} \delta_w, & \Delta_{ij} \cdot \Delta^2_{ij} < 0 \\ \delta_l, & \text{otherwise.} \end{cases} \tag{9}$$

$$\delta_{ij} = min\left(\pi_{ij}, \frac{\delta}{|C|-1}\right) \tag{10}$$

$$\Delta\pi_{ij} = \begin{cases} -\delta_{ij} & j \neq \text{argmax}_{l \in C} T^+_{il} \\ \delta_{ij} & \text{otherwise} \end{cases} \tag{11}$$

Finally, the transition rule of equation (1) was changed to:

$$p_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta \cdot [\pi_{ij}]^\phi}{\sum_{l \in J^k_i} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta \cdot [\pi_{il}]^\phi}, \tag{12}$$

where π_{ij} is the policy associated with edge (i,j) and ϕ is a weighting factor for policies. This transition rule is common to all the ACO algorithms except for ACS. ACS's rule is explained below.

B. Elitist Ant System

The elitist ant system proposed in [14] is identical to AS except that in addition to the regular ants a set of e "elitist" ants deposit pheromone on the edges of the best tour found since the start of the trial, T^+. Edges in T^+ receive $e \cdot Q/L^+$ pheromone, where L^+ is the length of T^+.

The proportional selection rule for AS with PDWoLF also applies to Elitist Ant System with PDWoLF.

C. Ant Colony System

Ant Colony System (ACS) [1], [12], [13] is an extension of Ant System (AS). In ACS, the transition rule is split into two parts. An ant in city i selects a random number $q \in [0,1]$ and then selects the next city j according to equation (13) where j is the city selected by the proportional transition rule of equation (1).

$$j = \begin{cases} \text{argmax}_{u \in J^k_i} \{[\tau_{iu}(t)]^\alpha \cdot [\eta_{iu}]^\beta\} & \text{if } q \leq q_0, \\ J & \text{if } q > q_0. \end{cases} \tag{13}$$

We updated the ACS transition rule for policies by changing equation (13) to

$$j = \begin{cases} \text{argmax}_{u \in J^k_i} \{[\tau_{iu}(t)]^\alpha \cdot [\eta_{iu}]^\beta \cdot [\pi_{iu}]^\phi\} & \text{if } q \leq q_0, \\ J' & \text{if } q > q_0. \end{cases} \tag{14}$$

where J' is the city selected using the PDWoLF proportional transition rule of equation (12).

The remaining differences between AS and ACS are not affected by PDWoLF. Briefly, those differences are:

- Only the best solution, T^+, found since the beginning of the trial is reinforced with pheromone.
- Pheromone is locally updated as ants move about constructing tours. If an ant uses edge (i, j), pheromone on that edge is updated by $\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \rho\cdot\tau_0$ where τ_0 is the initial amount of deposited on edges at the start of a trial.
- Instead of considering all unvisited cities reachable from city i ants are at first restricted to choosing from a candidate list of the c closest cities to city i. Only when the candidate list for a city is exhausted can an ant chose from other cities.
- For larger problems a non-reordering 3-opt local search algorithm is run on each ant's tour in each timestep. This particular version of ACS is known as ACS-3-opt.

D. Rank-based Ant System

Rank-based Ant System (RAS) [15] uses weighted elitist strategy for depositing pheromone. After each timestep, one of the elitist ants deposits $\sigma \cdot Q/L^+$ on the edges of the best tour found in the current trial. The remaining $\sigma - 1$ ants deposit pheromone based on the set of r best-ranked (shortest) solutions of the current timestep:

$$\Delta\tau_{ij}^{r} = \sum_{\mu=1}^{\sigma-1} \Delta\tau_{ij}^{\mu} \tag{15}$$

$$\Delta\tau_{ij}^{\mu} = (\sigma-\mu)\cdot Q/L^{\mu} \text{ iff ant } \mu \text{ uses edge } (i,j). \tag{16}$$

The pheromone evaporation equation for RAS is

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \sigma\cdot\Delta\tau_{ij}^{+} + \Delta\tau_{ij}^{r}. \tag{17}$$

E. Max-Min Ant System

Of all the legacy ACO algorithms, Max-Min Ant System (MMAS) [16] differs the most from AS and does so in five ways. Each of the five differences can be parameterized and activated as desired.

1) Elitist Update Strategy: An elitist strategy with $e = 1$ is used (see III-B). As an added twist, the edges updated by the elitist ant can be either those of the global best solution or the best solution from the current timestep, t. For larger TSPs, the creators of MMAS found it best to alternate between timestep-best and global-best updates according to a schedule that increases the frequency of global-best updates. Specifically, we employ the same update schedule used in [16] for symmetric TSPs as shown in Table I.

2) Bounded Pheromone Trail Intensity: Pheromone trail intensities are bounded by $[\tau_{min}, \tau_{max}]$ (hence the name Max-Min Ant System). τ_{max} is updated each time a new global-best solution is found in such a way that τ_{max} is an estimate of the highest possible pheromone level on an edge given the cost

TABLE I
UPDATE SCHEDULE FOR GLOBAL-BEST SOLUTIONS IN MMAS.

Timestep	Global-best to Timestep-best ratio
1 to 25	0:1 (timestep-best only)
26 to 75	1:4
76 to 125	1:2
126 to 250	1:1
> 250	1:0 (global-best only)

of the global-best solution (see [16] for details). τ_{min} is set according to the following equation

$$\tau_{min} = \frac{\tau_{max}\cdot(1-\sqrt[n]{p_{best}})}{(\frac{n}{2}-1)\cdot\sqrt[n]{p_{best}}}, \tag{18}$$

where p_{best} is a constant reflecting the probability that an ant constructs the best tour possible and n is the number of cities in the TSP. The choice of p_{best} influences the amount of exploration done by MMAS; the smaller p_{best} is, the closer τ_{min} is to τ_{max} [16].

3) Pheromone Initialization: Instead of intializing edges to some arbitrarily low value for τ_0 (initial pheromone level of an edge), MMAS initializes all edges to some arbitrarily high value. After the first timestep, a global-best solution will have been found and trails will be bounded by computed values of τ_{min} and τ_{max}.

4) Pheromone Re-initialization: MMAS reinitializes pheromone trails when solution values stall (solution not improved in 250 timesteps) or when the λ-branching factor falls below a set threshold. After reinitializing, the update schedule for global-best solutions (Table I) is reset.

5) Local optimization: Like ACS, a 3-opt local search algorithm improves each ant's tour. The recommended execution of MMAS includes all of these enhancements and the use of the 3-opt local search algorithm [16].

IV. ANALYSIS

For comparison testing, the code used for the results in [2] (available for download at [17]) is modified with the PDWoLF changes discussed in Section III. The settings and initialization parameters for AS, EAS, RAS, ACS, and MMAS shown in Table II are those found to give the best overall performance in [2]. For all algorithms, initial pheromone τ_0, was based on the tour lenght produced by a nearest neighbor heuristic: C^{nn}. The ACS and MMAS algorithms make use of a 3-opt local search. Also, MMAS performs a pheromone reinitialization if the λ-branching factor ≤ 2.0 and there has been no improvement in the last 250 iterations. Additionally, the MMAS algorithm alternates between updating the timestep-best and global-best tours according to the schedule in Table I. The PDWoLF specific parameters are the policy weight ($\phi = 1.0$), the winning size ($\delta_w = 0.05$), and the losing step size ($\delta_l = 0.15$), and remain consistent in all tests. The values of δ_l and δ_w are set based on the values identified in [20]. The policy values are initialized to $1/n$.

The experiment consists of 30 trials of each algorithm for ten symmetric TSP problems. The TSP instances tested include eil51, kroA100, d198, lin318, pcb442, att532, rat783, pcb1173,

TABLE II
ALGORITHM-SPECIFIC PARAMETERS

	AS	EAS	RAS	ACS	MMAS
α	1	1	1	–	1
β	2	2	2	2	2
ρ	0.5	0.5	0.1	0.1	0.2
m	n	n	n	10	25
τ	$1/(\rho C^{nn})$	$1/(\rho C^{nn})$	$1/(\rho C^{nn})$	$1/(nC^{nn})$	$1/(\rho C^{nn})$
e	–	n	–	–	–
σ	–	–	6	–	–
ϵ	–	–	–	0.1	–
q_0	–	–	–	0.98	–
local search	–	–	–	3-opt	3-opt

d1291, and pr2392. The best solution per trial is the shortest tour found over 1,000 timesteps.

Tables III, IV, and V list the best and average tour lengths along with standard deviations resulting from 30 trials for each algorithm. Results were compared for statistical difference using a paired two-tailed t-test with $\alpha = 0.05$. Comparing each algorithm with and without PDWoLF, the statistically better results are in bold. If the results are not bold, then they were not statistically different.

As can be seen in Table III, the addition of PDWoLF to Ant System (AS) results in a significant reduction in tour length for all problems. For Elitist Ant System (EAS), PDWoLF improves the solutions as the problem size increases.

However, for Rank-based Ant System (RAS) (Table IV), the addition of PDWoLF results in significantly worse performance on all but the largest problem. The distinguishing element between RAS and AS and EAS is that in RAS only the top σ best ants lay pheromone, while in AS and EAS, every ant does. This exploitative nature may explain why the PDWoLF version of RAS performs poorly. This situation exists for ACS and MMAS as well, but the inclusion of the 3-opt local search overrides this as we discuss in the next paragraph. We conducted several tests to try to identify why this occurs. As best we could determine, the rank update (updating the σ best ants) and the PDWoLF policy update (timestep-best-ant) provide competing information. As the rank size increases, RAS begins to perform more like EAS and PDWoLF shows an improvement. Similarly, as the policy weighting is reduced, the algorithm performs more like RAS.

For the MMAS and ACS algorithms that use the 3-opt local search, no significant benefit of adding PDWoLF appears (Table V). ACS and ACS with PDWoLF have no significant differences in solution quality, and for MMAS and MMAS with PDWoLF, the few statistically significant differences do not indicate a consistent trend. PDWoLF cannot help because the tour improvements provided by 3-opt overwhelms it. In fact, without 3-opt, the tour lengths of MMAS and ACS are dramatically worse than the local search augmented results. For example, without 3-opt had a solution average with a standard deviation of 10267.4 ± 56.7 and ACS came in at 13268.1 ± 113.5. Both results are significantly worse than all but the AS and EAS results. Consequently, any benefit provided by PDWoLF to these algorithms is swamped by local search.

The addition of PDWoLF to ant based algorithms can result in a significant improvement in results as shown with the AS and EAS results. However, PDWoLF does not make sense for all of the ant colony optimization algorithms, specifically RAS. Specifically since PDWoLF helps direct agents to previous good solutions (exploitative information) in a way already similar to RAS (and MMAS and ACS, which predominantly update the best-so-far or iteration-best tours).

V. RELATED WORK

Since its introduction, the Ant Colony Optimization heuristic has been adapted to a wide range of combinatorial optimization problems despite the known drawbacks of a slow convergence rate and lengthy run times, especially on large problems. Papers on ACO fall into three broad categories:

1) adapting ACO to a new problem domain including continuous solution spaces;
2) investigations into behavior and properties; and
3) efforts to improve the performance of ACO in terms of convergence rate, solution quality, or both.

This paper falls into the third category. Work within this category has primarily progressed on six fronts: local search, trail stirring, parameter tuning, search restrictions, memory systems, and parallelization. Since this paper presents a hybrid ACO algorithm, we relate work on other hybrid ACO approaches within the performance improvement category. Parallel ACO techniques are omitted because they primarily focus on splitting the workload and reconciling solutions and pheromone trails instead of hybridization. The combination of ACO with PDWoLF is unique; no other hybrid algorithm is so tightly integrated.

A. Local Search

Local search has been used to improve ACO performance for some time. Indeed two of the legacy ant algorithms examined in this paper, namely ACS [1], [12], [13] and MMAS [16], use 3-opt to improve ant tours. More recent efforts include using simulated annealing [23], random walks [24], tabu searches [24], *H*-method [25], minimum spanning trees [26], forward checking and back-tracking [27]. Repairing of infeasible solutions can also be considered a form of local search [28]. To the best of our knowledge, all ant algorithms implementing some kind of local search are only lightly coupled; local search is used to seed solutions for ants or to improve or clean up ant-generated solutions and not to guide ants as they build solutions.

TABLE III
MINIMUM TOUR LENGTHS (AS AND EAS)

Problem	AS		AS with WoLF		EAS		EAS with WoLF	
	Best	Avg±Stdev	Best	Avg±Stdev	Best	Avg±Stdev	Best	Avg±Stdev
eil51	436	442.5±4.2	428	**437.7±7.5**	427	**430.8±2.19**	432	452.3±10.16
kroA100	22324	22619.0±133.7	21566	**22132.0±349.3**	21363	**21905.6±262.1**	22158	22900.9±435.37
d198	16860	17201.5±139.9	16212	**16844.0±297.39**	16025	**16278.2±205.8**	16242	16783.4±330.9
lin318	45858	46688.7±320.5	43861	**44876.9±759.9**	42680	**43876.5±512.3**	43527	44759.9±567.0
pcb442	59091	60015.3±450.1	53998	**56450.1±111.7**	57314	58524.0±492.4	54026	**55361.9±698.8**
att532	32049	32628.6±251.3	29617	**30970.7±637.2**	30568	31444.7±374.6	29343	**30243.9±511.8**
rat783	10563	10677.3±45.9	9647	**9935.9±198.0**	10171	10438.2±102.6	9499	**9787.7±147.9**
pcb1173	69612	70731.5±423.4	65315	**66808.8±763.2**	69724	70473.1±307.2	64703	**66362.3±1095.1**
d1291	57300	57978.9±327.9	54845	**56779.4±731.8**	57013	57678.0±405.5	54493	**56378.5±1057.5**
pr2392	477437	485131±2428.3	443522	**458501.6±5671.2**	481091	485345.9±1964.6	446384	**458530.3±4913.1**

TABLE IV
MINIMUM TOUR LENGTHS (RAS)

Problem	RAS		RAS with WoLF	
	Best	Avg±Stdev	Best	Avg±Stdev
eil51	427	**430.8±2.2**	427	436.4±5.9
kroA100	21321	**21550.5±165.5**	21609	22152.4±394.2
d198	15962	**16166.3±91.5**	16289	16519.8±152.6
lin318	42753	**43290.8±312.3**	43674	44542.4±521.2
pcb442	51841	**52380.0±411.1**	52761	54040.7±645.1
att532	28588	**28834.5±221.5**	29146	29517.3±224.4
rat783	9031	**9120.4±48.0**	9354	9489.9±95.4
pcb1173	59423	**61980.8±1110.2**	61087	63023.9±913.7
d1291	52077	**53160.7±671.6**	53045	54514.9±740.72
pr2392	443131	464018.9±13862.0	426285	**441632.3±9672.9**

TABLE V
MINIMUM TOUR LENGTHS (MMAS AND ACS)

Problem	MMAS		MMAS with WoLF		ACS		ACS with WoLF	
	Best	Avg±Stdev	Best	Avg±Stdev	Best	Avg±Stdev	Best	Avg±Stdev
eil51	426	426.0±0.0	426	426.0±0.0	426	426.0±0.0	426	426.0±0.0
kroA100	21282	21282.0±0.0	21282	21282.0±0.0	21282	21282.0±0.0	21282	21282.0±0.0
d198	15780	15780.5±0.5	15780	15780.5±0.5	15780	15780.4±0.5	15780	15780.7±0.5
lin318	42029	**42031.1±11.3**	42029	42080.0±57.3	42029	42097.2±59.1	42029	42133.8±67.9
pcb442	50778	50882.03±55.21	50778	**50850.0±66.7**	50785	50923.3±72.5	50785	50934±81.5
att532	27686	27702.6±8.67	27686	27702.2±9.11	27693	27721.2±25.4	27690	27733.7±41.6
rat783	8806	8811.9±6.2	8804	8814.7±6.7	8809	8833.3±14.78	8808	8832.3±15.2
pcb1173	56892	56968.3±60.9	56893	57001.1±101.5	56904	57151.17±131.98	56923	57172.2±132.6
d1291	50801	**50828.8±26.9**	50801	50868.7±82.5	50801	50893.27±112.2	50801	50898.17±80.0
pr2392	378414	**379257.4±354.5**	379368	380632.9±935.63	378826	380144.4±630.7	378816	380042.4±579.1

B. Trail Stirring

Improving performance via stirring of the pheromone trails was introduced first in ACS [1], [12], [13]. In ACS it took the form of slightly evaporating pheromone on edges traversed by ants during solution construction. The popularity of trail stirring remains high [29].

C. Parameter Tuning

Ant algorithm parameters such as the weighting factors α and β, colony size m, and pheromone evaporation rate ρ have long been the subject of study. Early on, parameters were tuned by hand on a problem-by-problem basis. More recently researchers have looked at formalizing the tuning process [30], [31] and automated tuning [32]–[35].

D. Search Restrictions

Because of the large search spaces frequently encountered, techniques to either reduce the number of decisions facing the ants or pruning of the search space are often employed. An early example of this include the candidate lists employed by ACO [1], [12], [13]. Recent approaches include beam searches [36], [37] and constraint checking [27]. Alternatively, using hierarchies, the search space is limited and approximated [22].

E. Memory Systems

Pheromone and policies act as a memory of past good solutions that ants refer to when constructing new solutions. Several researchers have added additional specialized memories to ACO in order to avoid recomputing past solutions [38] or reusing parts of past solutions [39], [40].

VI. CONCLUSION

This paper presents the tight integration of the variable step size off-policy hill-climbing algorithm PDWoLF with several ant colony algorithms: AS, EAS, RAS, ACS, and MMAS. Experiments on large traveling salesman problems (TSPs) show that incorporating the off-policy variable step size mechanism into an ant colony optimization algorithm can result in shorter tours than the ACO algorithms alone. As mentioned in the results, further investigation is needed to determine why adding PDWoLF to ant algorithms with local search optimizations (ACS and MMAS) results in little improvement, and in the case of RAS results in significantly worse solutions. One aspect of this is the greedy nature of these variations of ant colony optimization. A second is that no parameter tuning was done on the algorithms that included PDWoLF. This was done to demonstrate the generalness of the approach, but also impacts the best performance available.

References

[1] E. Bonabeau, M. Dorigo, and T. Théraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, 1999.

[2] Marco Dorigo and Thomas Stützle, *Ant Colony Optimization*, The MIT Press, 2004.

[3] L. M. Gambardella, E. D. Taillard, and M. Dorigo, "Ant colonies for the qap", *Journal of the Operational Research Society*, vol. 50, no. 2, pp. 167–176, 1999.

[4] V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment pproblem", *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 5, pp. 769–778, 1998.

[5] T. Stützle and H. Hoos, "Max-min ant system and local search for combinatorial optimization problems", in *Second International Conference on Metaheuristics, MIC'97*. 1998, Kluwer Academic.

[6] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian, "Ant system for job-shop scheduling", *Belgian Journal of Operations Research, Statistics and Computer Science*, vol. 34, no. 1, pp. 39–53, 1994.

[7] Christian Blum and Michael Sampels, "An ant colony optimization algorithm for shop scheduling problems", *J. of Mathematical Modelling and Algorithms*, vol. 34, no. 3, pp. 285–308, 2004.

[8] Marco Dorigo and Thomas Stützle, *Handbook of Metaheuristics*, vol. 57 of *International Series in Operations Research and Management Science*, chapter The ant colony optimization metaheuristic: Algorithms, applications, and advances, pp. 251–285, Kluwer Academic Publishers, 2002.

[9] Bikramjit Banerjee and Jing Peng, "Adaptive policy gradient in multi-agent learning", in *2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2003, pp. 686–692, ACM Press.

[10] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by an colonies", in *Proceedings of the First European Conference on Artificial Life*, F. Varela and P. Bourgine, Eds. 1991, pp. 134–142, MIT Press.

[11] A. Colorni, M. Dorigo, and V. Maniezzo, "An investigation of some properties on an ant algorithm", in *1992 Parallel Problem Solving from Nature Conference*. 1992, pp. 509–520, Elsevier.

[12] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the travelling salesman problem", *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[13] M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling salesman probelm", *BioSystems*, vol. 43, pp. 73–81, 1997.

[14] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents", *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 26, 1996.

[15] Bernd Bullnheimer, Richard F. Hartl, and Christine Strauß, "A new rank based version of the ant system — a computational study", Tech. Rep., University of Viena, Institute of Management Science, 1997.

[16] T. Stützle and H. H. Hoos, "Max-min ant system", *Future Generation Computer Systems*, vol. 16, no. 8, pp. 889–914, 2000.

[17] Thomas Stützle, "ACOTSP V1.0.1", iridia.ulb.ac.be/∼mdorigo/ACO/downloads/ACOTSP.V1.01.tar.gz.

[18] Richard M. Karp, *In Complexity of Computer Computations*, chapter Reducibility among combinatorial problems, pp. 85–103, Plenum Press, 1972.

[19] Satinder P. Singh, Michael J. Kearns, and Yishay Mansour, "Nash convergence of gradient dynamics in general-sum games", in *16th Conference on Uncertainty in Artificial Intelligence*. 2000, pp. 541–548, Morgan Kaufmann.

[20] Michael H. Bowling and Manuela M. Veloso, "Multiagent learning using a variable learning rate", *Artificial Intelligence*, vol. 136, no. 2, pp. 215–250, 2002.

[21] Christopher J.C.H. Watkins and Peter Dayan, "Q-learning", *Machine Learning*, vol. 8, pp. 279–292, 1992.

[22] Erik J. Dries and Gilbert L. Peterson, "Scaling ant colony optimization with hierarchical reinforcement learning partitioning", in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, New York, NY, USA, 2008, GECCO '08, pp. 25–32, ACM.

[23] B. Dengiz, F. Altiparmak, and O. Belgin, "A hybrid ant colony optimization approach for the design of reliable networks", in *IEEE Congress on Evolutionary Computation*, 2007, pp. 1118–1125.

[24] M. Mouhoub and Z. Wang, "Improving the ant colony optimization algorithm for the quadratic assignment problem", in *IEEE Congress on Evolutionary Computation*, 2008, pp. 250–257.

[25] Leonid Hulianytskyi and Sergii Sirenko, "Hybrid metaheuristic combining ant colony optimization and *h*-method", in *ANTS Conference*. 2010, vol. 6234 of *Lecture Notes in Computer Science*, pp. 568–569, Springer.

[26] M. Kheirkhahzadeh and A. A. Barforoush, "A hybrid algorithm for the vehicle routing problem", in *IEEE Congress on Evolutionary Computation*, 2009, pp. 1791–1798.

[27] David C. Uthus, Patricia J. Riddle, and Hans W. Guesgen, "An ant colony optimization approach to the traveling tournament problem", in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, New York, NY, USA, 2009, GECCO '09, pp. 81–88, ACM.

[28] Zhigang Ren and Zuren Feng, "An ant colony optimization approach to the multiple-choice multidimensional knapsack problem", in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, New York, NY, USA, 2010, GECCO '10, pp. 281–288, ACM.

[29] Xiangyin Zhang, Haibin Duan, and Jiqiang Jin, "Deaco: Hybrid ant colony optimization with differential evolution", in *IEEE Congress on Evolutionary Computation*, 2008, pp. 921–927.

[30] Haibin Duan, Guanjun Ma, and Senqi Liu, "Experimental study of the adjustable parameters in basic ant colony optimization algorithm", in *IEEE Congress on Evolutionary Computation*, 2007, pp. 149–156.

[31] Paola Pellegrini, Daniela Favaretto, and Elena Moretti, "On max-min ant system's parameters", in *ANTS Workshop*. 2006, vol. 4150 of *Lecture Notes in Computer Science*, pp. 203–214, Springer.

[32] M. Maur, M. López-Ibáñez, and T. Stützle, "Pre-scheduled and adaptive parameter variation in max-min ant system", in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[33] Paola Pellegrini, Thomas Stützle, and Mauro Birattari, "Off-line vs on-line tuning: A study on max-min ant system for the tsp", in *ANTS Conference*. 2010, vol. 6234 of *Lecture Notes in Computer Science*, pp. 239–250, Springer.

[34] Wei jie Yu and Jun Zhang, "Pheromone-distribution-based adaptive ant colony system", in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, New York, NY, USA, 2010, GECCO '10, pp. 31–38, ACM.

[35] Adrian A. de Freitas and Christopher B. Mayer, "The effectiveness of dynamic ant colony tuning", in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 2007, pp. 170–170.

[36] Christian Blum, Joaquín Bautista, and Jordi Pereira, "Beam-aco applied to assembly line balancing", in *ANTS Workshop*. 2006, vol. 4150 of *Lecture Notes in Computer Science*, pp. 96–107, Springer.

[37] Christian Blum, "Beam-aco for the longest common subsequence problem", in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[38] Shih-Pang Tseng, Chun-Wei Tsai, Ming-Chao Chiang, and Chu-Sing Yang, "A fast ant colony optimization for traveling salesman problem", in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–6.

[39] Shigeyoshi Tsutsui, "Cunning ant system for quadratic assignment problem with local search and parallelization", in *Second International Conference on Pattern Recognition and Machine Intelligence (PReMI)*. 2007, vol. 4815 of *Lecture Notes in Computer Science*, pp. 269–278, Springer.

[40] A. A. Acan and A. Unveren, "A shared-memory aco+ga hybrid for combinatorial optimization", in *IEEE Congress on Evolutionary Computation*, 2007, pp. 2078–2085.