Received December 1, 2018, accepted January 11, 2019, date of publication January 30, 2019, date of current version February 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2896104

Parameter Self-Adaptation in an Ant Colony Algorithm for Continuous Optimization

ASHRAF M. ABDELBAR[1] AND KHALID M. SALAMA[2]
[1]Department of Mathematics and Computer Science, Brandon University, Brandon, MB R7A 6A9, Canada
[2]School of Computing, University of Kent, Canterbury CT2 7NZ, U.K.

Corresponding author: Ashraf M. Abdelbar (abdelbara@brandonu.ca)

This work was supported in part by the Brandon University Research Council.

ABSTRACT ACO$_\mathbb{R}$ is a well-established ant colony optimization algorithm for continuous-domain optimization. We present an approach for the dynamic adaptation of the ACO$_\mathbb{R}$ algorithm's controlling parameters, focusing on the search width parameter, based on using several pre-specified parameter configurations, which we call personalities. Before an ant starts to generate a candidate solution, it stochastically adopts a personality based on the relative past success of different personalities. The success of a personality is measured, in turn, by the survival rate of the previous solutions generated by the ants adopting that personality. The premise of our approach is that some personalities will be more appropriate than others for different phases of the search. In addition, our adaptive approach can accommodate solution recombination, the use of which within ACO$_\mathbb{R}$ was recently explored in the previous work. It allows the frequency of applying recombination and the type of recombination operator to be dynamically adapted, by having one or more recombination personalities among the competing personalities. We evaluate these proposals experimentally on two applications: 1) training feedforward neural networks for classification using 65 benchmark datasets from the University of California Irvine repository and 2) optimizing several popular synthetic benchmark continuous-domain functions. Our experimental results indicate that our proposals perform better than the standard ACO$_\mathbb{R}$ on both applications, to a statistically significant extent.

INDEX TERMS Collective intelligence, evolutionary computation, adaptive algorithm, optimization methods, ant colony optimization.

I. INTRODUCTION

ACO$_\mathbb{R}$ [1] is a well-established Ant Colony Optimization (ACO) [2] algorithm for continuous-domain optimization. In this paper, we present an approach for the dynamic self-adaptation of the algorithm's controlling parameters, focusing on the search width parameter ξ. Our proposal falls in the general area of automated parameter tuning [3]–[8], a category in which there has been considerable interest in the literature. Such techniques can be broadly classified into offline and online approaches. Offline approaches optimize an algorithm's parameters before the algorithm is deployed, and include methods such as iterated local search [4], [5], iterated race algorithms [9], [10], evolutionary computation [11]–[13], and even ACO algorithms [14]–[16].

On the other hand, online approaches operate during an algorithm's execution and modify the algorithm's parameters while it is running. Stützle *et al.* [8] note that online approaches "may also be useful to reach the best performance in dependence of the stage of the search," and that "allowing parameters to change online may increase an algorithm's robustness."

One taxonomy of online parameter tuning techniques was presented by Eiben *et al.* [3], and Stützle *et al.* [8] have presented an excellent survey, organized around Eiben's taxonomy, of online parameter tuning methods that have been applied to (discrete) ACO. Eiben's taxonomy is based on three categories: pre-scheduled, adaptive, and self-adaptive approaches.

Pre-scheduled approaches allow an algorithm's parameters to be modified, deterministically or stochastically, based on the number of iterations or the amount of CPU-time elapsed. The Incremental ACO$_\mathbb{R}$ [17] algorithm's strategy of gradually increasing archive size can be considered to fall within this category, as the increase in the size of the search archive follows a specific schedule based on the iteration number.

The associate editor coordinating the review of this manuscript and approving it for publication was Sabah Mohammed.

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Other examples of pre-scheduled approaches that have been applied to discrete ACO can be found in [18]–[20].

Adaptive methods modify an algorithm's parameters based on statistical measures of the algorithm's dynamic behavior. Examples of adaptive methods that have been applied to discrete ACO can be found in [21]–[27].

Self-adaptive approaches are those in which the algorithm modifies its own parameters at run-time. Examples of self-adaptive approaches applied to discrete ACO include several different methods [28]–[31] with a common theme of associating pheromone traces with values of discrete ACO parameters, such as α, β, and ρ, and using the usual ACO roulette wheel probabilistic-action rule to choose between parameter values based on the associated pheromone traces. Our $\text{ACO}_{\mathbb{R}}$-**P** approach (see Section III) falls within this category. In $\text{ACO}_{\mathbb{R}}$, the role of pheromone is played by the solution archive, and our approach uses information extracted from the archive to choose between parameter settings (personalities) based on roulette wheel selection.

Other self-adaptive approaches that have been applied to discrete ACO include several methods that have used different flavors of Evolutionary Algorithms (EA) to adapt ACO parameters [32]–[34]. A Particle Swarm Optimization (PSO) approach [35] and an artificial fish-swarm approach [36] have also been considered, as have approaches based on local search in parameter space [37], and those based on multiple cooperating colonies with different parameter configurations for each colony [38]. The reader is referred to [8] for a more comprehensive discussion.

To our knowledge, our work is the first self-adaptive approach to be proposed in the context of $\text{ACO}_{\mathbb{R}}$. Our proposed approach uses several pre-specified parameter configurations, which we call *personalities*. Before an ant starts to generate a candidate solution, it first adopts a personality, or parameter configuration. The choice of which personality to adopt is stochastic and based on how successful each personality has been in the past. The success of a personality p is measured, in turn, by the success rate with which solutions generated by previous p-personality ants have succeeded in securing a place in the archive. The premise of our approach is that some personalities will be more appropriate than others for different phases of the search process.

Our approach allows personalities to compete. Personalities that are more effective for a particular phase of the search will be deployed more often. As the search drifts towards a different phase, previously effective personalities will become less effective as others emerge and dominate. This adaptation and competition among personalities will make the algorithm more responsive to different phases of the search process.

In [39], we explored the incorporation of recombination within the $\text{ACO}_{\mathbb{R}}$ algorithm. In the present work, we propose an approach for dynamically adapting the frequency of applying recombination and the type of recombination operator by having one or more recombination personalities among the competing personalities.

In [40], we proposed an approach, which we now call $\text{ACO}_{\mathbb{R}}$-**D**, in which the $\text{ACO}_{\mathbb{R}}$ algorithm's search width parameter ξ is reduced over time according to a pre-determined exponential decay schedule. At the start of the algorithm, or after a stagnation-reset (see Section II), the search width ξ is set to an initial value ξ_0, and then is reduced by a fixed factor in each iteration. It can be argued that $\text{ACO}_{\mathbb{R}}$-**D** falls into the category of pre-scheduled approaches because of the fixed decay schedule. On the other hand, it can also be argued that the mechanism of resetting the value of ξ to ξ_0 in the case of stagnation is an example of an adaptive strategy.

Our experimental evaluation is based on two applications: 1) training feedforward neural networks for classification, using 65 benchmark datasets from the UCI repository; 2) optimizing several synthetic continuous-domain benchmark functions that are popular in the evolutionary computation community.

In summary, the contributions of the present work can be categorized as follows:

1) We introduce an approach, called $\text{ACO}_{\mathbb{R}}$-**P**, for the self-adaptation of the $\text{ACO}_{\mathbb{R}}$ algorithm's controlling parameters. The present work focuses on the search width parameter ξ, but our approach can be generalized to other parameters as well.
2) In [39], we presented the $\text{ACO}_{\mathbb{R}}$-**R** algorithm, which augments $\text{ACO}_{\mathbb{R}}$ with a mechanism for the recombination of solutions from the archived population, with the probability of deployment of recombination being fixed and specified by a user-supplied parameter. We follow up, in the present work, by considering how the probability of recombination can be dynamically adapted by incorporating one or more recombination operators within the framework of $\text{ACO}_{\mathbb{R}}$-**P**. We allow the frequency of recombination and the type of recombination operator to be dynamically adapted based on the past performance of each operator. We present the $\text{ACO}_{\mathbb{R}}$-**PR** algorithm, which includes a single recombination operator, and the $\text{ACO}_{\mathbb{R}}$-**PR2** algorithm, which includes two competing recombination operators corresponding to uniform crossover and single-point crossover.
3) We present an extensive experimental evaluation on two applications: a) neural network training using 65 datasets; and b) optimizing several synthetic continuous-domain benchmark functions using 9 functions and 10 values for the number of dimensions for each function. Our experimental evaluation compares the following four algorithms to standard $\text{ACO}_{\mathbb{R}}$: $\text{ACO}_{\mathbb{R}}$-**P**, $\text{ACO}_{\mathbb{R}}$-**PR**, $\text{ACO}_{\mathbb{R}}$-**PR2**, and $\text{ACO}_{\mathbb{R}}$-**D**. The first three are introduced in the present work, while $\text{ACO}_{\mathbb{R}}$-**D** was introduced in [40]. In that work, $\text{ACO}_{\mathbb{R}}$-**D** was evaluated on neural network training using 36 datasets. The experimental evaluation in the present work increases the number of neural network datasets from 36 to 65, and also evaluates

ACO$_{\mathbb{R}}$-**D** on the benchmark continuous optimization functions.

The paper is organized as follows. We begin in Section II with a broad overview of Ant Colony Optimization (ACO) and a review of the ACO$_{\mathbb{R}}$ algorithm. Section III presents our competing personalities approach, while Section IV describes how personalities encoding recombination operators can be accommodated within the competing personalities approach. We then describe our experimental methodology in Section V, followed by our experimental results in Section VI. Finally, Section VII presents some final discussion and describes some potential future work.

II. REVIEW OF THE ACO$_{\mathbb{R}}$ ALGORITHM

Ant Colony Optimization (ACO) [2] is a general-purpose, biologically motivated, population-based optimization metaheuristic, based on a collection (called a *colony*) of fairly primitive processing elements (called *ants*), each operating autonomously and communicating with the others indirectly through shared data structures. ACO algorithms usually have a central data structure, analogous to pheromone information in natural ant colonies, that represents the time-evolving collective wisdom of the group. In each iteration, each ant typically generates a candidate solution, making use of the central pheromone data structure in some way in its solution construction. After all ants have generated their solutions, a subset of those solutions is then used to update the central data structure in some way.

ACO has been applied to a wide variety of domains [41]–[44], including supervised learning using various learning models, such as classification rules [29], [45]–[50], decision trees [51], [52], and various types of Bayesian network classifiers [53]–[57].

While the majority of research on ACO has focused on discrete (combinatorial) optimization problems [58], ACO methods for continuous problem domains have also been investigated [59]–[61]. In this paper, we focus on the ACO$_{\mathbb{R}}$ algorithm [59], which has been applied to a number of continuous optimization problems [1], [59], including neural network training [60], [62]–[64].

Suppose the ACO$_{\mathbb{R}}$ algorithm is to be applied to an optimization problem over n real-valued variables $V_1, V_2, \ldots, V_n$. The central data structure, analogous to pheromone information in natural ants, that is maintained by ACO$_{\mathbb{R}}$ is an archive A of L previously-generated candidate solutions. Each element s_a in the archive, for $a = 1, 2, \ldots, L$, is an n-dimensional, real-valued vector, $s_a = (s_{a,1}, s_{a,2}, \ldots, s_{a,n})$. For example, $s_{a,j}$ refers to the value of the j-th variable in the a-th solution in the archive. The archive is sorted by solution quality, so that $Q(s_1) \geq Q(s_2) \geq \ldots \geq Q(s_L)$. Each solution s_a in the archive has an associated weight ω_a that is related to $Q(s_a)$, so that $\omega_1 \geq \omega_2 \geq \ldots \geq \omega_L$.

Each iteration of the ACO$_{\mathbb{R}}$ algorithm has the following two phases: solution construction and pheromone update. In the solution construction phase, each ant probabilistically constructs a solution based on the solution archive A (representing pheromone information). The solution archive A is initialized with L randomly generated solutions, where the size L is a user-supplied parameter of the ACO$_{\mathbb{R}}$ algorithm. Then, in the pheromone update phase, the m constructed solutions (where m is the number of ants) are added to A, resulting in the size of A temporarily being $L + m$. The archive A is then sorted by solution quality, and the m worst solutions are discarded, so that the size of A returns to being L.

The heart of the algorithm is the solution construction phase. In this phase, each ant i generates a candidate solution s_i, where s_i is an n-dimensional vector, and $s_{i,j}$ represents an assignment to the j-th variable V_j. In constructing its solution s_i, ant i is influenced by one of the L solutions in the archive A. The ant first probabilistically selects one of the L solutions in the archive according to:

$$\Pr(\text{select } s_a) = \frac{\omega_a}{\sum_{r=1}^{L} \omega_r} \tag{1}$$

Thus, the probability of selecting the a-th solution is proportional to its weight ω_a. Recall that the archive A is sorted by quality, so that solution s_a has rank a, with the best solution having a rank of 1. The weights ω_a that are used in Eq. (1) are constructed in each iteration as:

$$\omega_a = g(a; 1, qL) \tag{2}$$

where g is the Gaussian function:

$$g(y; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} \tag{3}$$

Thus, Eq. (2) assigns the weight ω_a to be the value of the Gaussian function with argument a, mean 1.0, and standard deviation (qL). The value of q is a user-supplied parameter of the algorithm, where smaller values of q cause the better-ranked solutions to have higher weights ω (and thus make the algorithm more exploitative), while larger values of q result in a more uniform distribution.

Let s_a be the solution of A that is selected by ant i according to Eq. (1) in a given iteration. Ant i then generates each solution element $s_{i,j}$ by sampling the Gaussian probability density function (PDF):

$$s_{i,j} \sim N(s_{a,j}, \sigma_{a,j}) \tag{4}$$

where $N(\mu, \sigma)$ represents the Gaussian PDF with mean μ and standard deviation σ.

In Eq. (4), $s_{a,j}$ represents the value that the solution s_a assigns to variable V_j, and the standard deviation $\sigma_{a,j}$ is computed according to:

$$\sigma_{a,j} = \xi \sum_{r=1}^{L} \frac{\mid s_{a,j} - s_{r,j} \mid}{L - 1} \tag{5}$$

where ξ is a user-supplied parameter of the algorithm. The effect of Eq. (5) is that the average distance from s_a to other solutions in the archive, for the j-th dimension, is computed and then multiplied by ξ. The parameter ξ plays a role in

Algorithm 1 Pseudo-code of ACO$_{\mathbb{R}}$.

```
algorithm ACO$_{\mathbb{R}}(n, Q)$    ▷ $n$ : number of dimensions; $Q$ : fitness function ($Q : R^n \mapsto R$)
    for $t = 1 \to I_{max}$ do
        if $t = 1$ OR Stagnation () then
            for $r = 1 \to L$ do    ▷ $L = |A|$
                Initialize $s_r$ ($r$-th element of archive $A$) with a randomly-generated $n$-dimensional vector
                Compute $Q(s_r)$
            end for
            Sort Archive $A$ according to $Q$
        end if
        for $i = 1 \to m$ do    ▷ $m$ : number of ants
            Select $s_a$ according to Eq. (1)
            for $j = 1 \to n$ do
                Compute $\sigma_{a,j}$ according to Eq. (5)    ▷ Eq. (5) uses the search width parameter $\xi$
                Generate $s_{i,j}$ according to Eq. (4)
            end for
            Compute $Q(s_i)$
        end for
        Add $s_1, \ldots, s_m$ to archive $A$    ▷ $A$ temporarily grows to size $L + m$
        Sort Archive $A$ according to $Q$
        Remove bottom $m$ elements from $A$    ▷ $A$ shrinks back to size $L$
    end for
    return best solution in Archive $A$
end algorithm
```

ACO$_{\mathbb{R}}$ similar to that of the evaporation rate in other ACO algorithms. The higher the value of ξ, the less the search is biased towards the area of the search space around the solutions stored in the archive, and the slower the algorithm will converge. Once each ant has constructed its solution, the archive A is updated as described previously, and the process repeats.

If the top solution in the archive remains unchanged for I_{stag} iterations, then the algorithm is said to be *stagnated*. When this happens, the archive is re-initialized with random solutions, although a record is maintained of the best solution encountered thus far. The algorithm terminates when the total number of iterations reaches I_{max}.

In all, besides I_{stag} and I_{max}, the ACO$_{\mathbb{R}}$ algorithm has the following four user-supplied parameters: m, L, q, and ξ. The parameter m determines the number of ants; the parameter L determines the number of solutions stored in the archive A; the parameter q controls the extent to which the top solutions in the archive will dominate solution construction (Eq. 2); and the search width parameter ξ influences the degree of diversity in solution construction (Eq. 5). Table 1 shows the ACO$_{\mathbb{R}}$ parameter settings that we use in our experimental evaluation, which follow the parameter settings of [1]. Algorithm 1 summarizes the ACO$_{\mathbb{R}}$ algorithm in pseudocode.

Liao *et al.* [17] have considered a variant of ACO$_{\mathbb{R}}$, called Incremental ACO$_{\mathbb{R}}$ (IACO$_{\mathbb{R}}$), which allows the archive size L to gradually increase over time. In the same work [17], they presented another variation, IACO$_{\mathbb{R}}$-LS, that incorporates local search within the IACO$_{\mathbb{R}}$ framework. Kumar *et al.* [65] have investigated and evaluated a wide array of optimizers for IACO$_{\mathbb{R}}$-LS's local search step.

TABLE 1. Parameter settings used in experimental evaluation.

Algorithm	Parameter	Setting
ACO$_{\mathbb{R}}$	m	5
	L	90
	q	0.05
	I_{stag}	650
	I_{max}	5000
	ξ	0.68
ACO$_{\mathbb{R}}$-**P**	θ	2.5
	$\lvert\Psi\rvert$	14
	$\hat{\psi}$	$\xi = 0.68$
ACO$_{\mathbb{R}}$-**PR**	$\lvert\Psi\rvert$	15
ACO$_{\mathbb{R}}$-**PR2**	$\lvert\Psi\rvert$	16
ACO$_{\mathbb{R}}$-**D**	ξ_0	0.68
	ξ'	0.28

III. MULTIPLE COMPETING PERSONALITIES

Our proposal, called ACO$_{\mathbb{R}}$-**P**, aims to allow the ACO$_{\mathbb{R}}$ algorithm to respond and be more closely coupled to the progress of the search. We specify a set of personalities $\Psi = \{\psi_1, \psi_2, \ldots, \psi_{|\Psi|}\}$, where each personality is a full or partial specification of algorithm parameter values. In this section, each personality simply specifies a value of the search width parameter ξ. In our experimental evaluation, we use 14 personalities corresponding to the following evenly spaced values of ξ:

$$\{0.93, 0.88, \ldots, 0.68, \ldots, 0.28\} \tag{6}$$

Each element of the archive will now have an associated field ϕ that records the personality that was used to generate it. Let $\varsigma(\psi)$ denote the number of elements of the archive that were generated by personality ψ. Note, of course, that

$$\sum_{\psi \in \Psi} \varsigma(\psi) = L \tag{7}$$

When an ant starts to generate a candidate solution, it first probabilistically adopts a personality according to the roulette wheel probabilistic-action rule:

$$\Pr(\text{select } \psi) = \frac{u(\psi)}{\sum_{\psi' \in \Psi} u(\psi')} \tag{8}$$

where u denotes the *utility* of personality ψ, which is a measure of the past performance of personality ψ. We define the function u as:

$$u(\psi) = \varsigma(\psi) + \theta \tag{9}$$

where θ is a small constant, similar in role to a Laplace correction, that prevents a personality from ever having a zero probability of adoption.

One of the elements of Ψ is distinguished as the default personality, denoted $\hat{\psi}$. When the archive is randomly initialized, either at the start of the computation or after a stagnation-reset, all the elements of the archive are considered as having been generated by the default personality $\hat{\psi}$. Thus, immediately after initialization, the adoption probability of $\hat{\psi}$ is:

$$\Pr(\text{select } \hat{\psi}) = \frac{L + \theta}{L + \theta|\Psi|} \tag{10}$$

and the adoption probability of any other personality $\psi \neq \hat{\psi}$ is:

$$\Pr(\text{select } \psi) = \frac{\theta}{L + \theta|\Psi|} \tag{11}$$

Table 1 summarizes the parameter settings that we use in our experimental evaluation for ACO$_{\mathbb{R}}$-**P**. Note that the default personality for ACO$_{\mathbb{R}}$-**P** corresponds to the search width setting ($\xi = 0.68$) that is used with standard ACO$_{\mathbb{R}}$ in our experimental evaluation. With these parameter settings ($\theta = 2.5$ and $|\Psi| = 14$), the adoption probabilities immediately after population initialization would be 74% for the default personality and 2% for each of the other 13 personalities. Algorithm 2 summarizes ACO$_{\mathbb{R}}$-**P** in pseudocode.

The premise of ACO$_{\mathbb{R}}$-**P** is that a personality that is effective at a particular point in the search will produce solutions of relatively high quality, which will be more likely to find a place in the archive and to survive in the archive longer before being displaced. As the algorithm progresses, search conditions may change, and a previously less-effective personality can become more effective. As the relative quality of solutions produced by that personality improves, its representation in the archive will increase, in turn improving its probability of adoption. The Laplace constant (θ in Eq. 9) ensures that no personality will ever die out completely.

The process of constructing a candidate solution in ACO$_{\mathbb{R}}$ can be viewed as sampling a probability distribution that is indirectly specified by the solution archive and the algorithm parameters. The m solutions constructed in each iteration are all produced by sampling the exact same probability distribution. In ACO$_{\mathbb{R}}$-**P**, solutions constructed by different personalities are produced by sampling slightly different distributions, thus promoting search diversity. Because of the Laplace constant θ, even poorly-performing personalities will still be adopted occasionally, thus promoting search diversity even further.

It is worth emphasizing that the run-time per iteration of each of the three algorithms (ACO$_{\mathbb{R}}$, ACO$_{\mathbb{R}}$-**D**, and ACO$_{\mathbb{R}}$-**P**) is roughly the same. The run-time of ACO$_{\mathbb{R}}$ is dominated by fitness function evaluations, and each of these three algorithms constructs and evaluates the fitness of exactly m solutions per iteration.

IV. MULTIPLE PERSONALITIES WITH RECOMBINATION

A. INCORPORATING RECOMBINATION WITHIN MULTIPLE PERSONALITY FRAMEWORK

In [39], we presented an approach called ACO$_{\mathbb{R}}$-**R**, which incorporates recombination within the ACO$_{\mathbb{R}}$ framework. In ACO$_{\mathbb{R}}$-**R**, recombination is deployed with a probability that is fixed and specified by a user-supplied parameter. In the present work, we allow the probability of applying recombination to be dynamically adapted by generalizing the ACO$_{\mathbb{R}}$-**P** multiple competing personalities approach to produce an approach we call ACO$_{\mathbb{R}}$-**PR**, which incorporates recombination within the multiple personality approach. Here, as in Section III, we apply multiple personalities, corresponding to the values of ξ indicated in Eq. (6). We add an additional personality that corresponds to the application of recombination, for a total of 15 personalities. If the personality corresponding to recombination is adopted, then a recombination operator is used to construct a candidate solution instead of ACO$_{\mathbb{R}}$'s usual solution construction mechanism. In the present work, we use uniform crossover as the recombination operator in ACO$_{\mathbb{R}}$-**PR**.

When the recombination personality is adopted, two parents are selected from the ACO$_{\mathbb{R}}$ solution archive. One parent s_a is selected by applying Eq. (1) in the usual ACO$_{\mathbb{R}}$ way (i.e. rank-proportionate selection). The second parent s_b is randomly selected from the archive with uniform distribution. A single offspring s_c is then generated by uniform crossover; each solution element $s_{c,j}$ is set equal to $s_{a,j}$ with 50% probability, and to $s_{b,j}$ with 50% probability, for $j = 1, \ldots, n$. The generated offspring s_c then becomes one of the m constructed

Algorithm 2 Pseudo-code of ACO$_\mathbb{R}$-**P**. This algorithm builds on Algorithm 1; lines that were added are marked with a plus sign next to the line number.

```
1: algorithm ACO$_\mathbb{R}$-P$(n, Q)$                     ▷ $n$ : number of dimensions, $Q$: fitness function ($Q : R^n \mapsto R$)
2:     for $t = 1 \to I_{max}$ do
3:         if $t = 1$ OR Stagnation () then
4:             for $r = 1 \to L$ do
5:                 Initialize $s_r$ with a randomly-generated $n$-dimensional vector
6:                 Compute $Q(s_r)$
+7:                Set $\phi(s_r) = \hat{\psi}$                 ▷ $\hat{\psi}$ : default personality
8:             end for
9:             Sort Archive $A$ according to $Q$
+10:           Compute $u(\psi)$ according to Eq. (9) for all $\psi \in \Psi$
11:        end if
12:        for $i = 1 \to m$ do
+13:           Select a personality $\psi$ according to Eq. (8)
14:            Select $s_a$ according to Eq. (1)
+15:           Set $\xi$ as specified by personality $\psi$    ▷ $\xi$ is used in Eq. (5) in line 17
16:            for $j = 1 \to n$ do
17:                Compute $\sigma_{a,j}$ according to Eq. (5)
18:                Generate $s_{i,j}$ according to Eq. (4)
19:            end for
20:            Compute $Q(s_i)$
+21:           Set $\phi(s_i) = \psi$
22:        end for
23:        Add $s_1, \ldots, s_m$ to archive $A$
24:        Sort Archive $A$ according to $Q$
25:        Remove bottom $m$ elements from $A$
+26:       Compute $u(\psi)$ according to Eq. (9) for all $\psi \in \Psi$
27:    end for
28: return best solution in Archive $A$
29: end algorithm
```

solutions that compete with each other and with the existing L solutions for a place in the archive. If the generated offspring succeeds in finding a place in the population, it is considered to have been constructed by the recombination personality.

Note that when the recombination personality is selected, ACO$_\mathbb{R}$'s standard solution generation mechanism (Eqs. 1-5) is not applied, and the parameter ξ does not play a role in solution generation; thus, no value of ξ is needed when the recombination personality is selected.

Based on our parameter settings (summarized in Table 1), the initial adoption probabilities would be 71.15% for the default personality and 1.92% for each of the other 14 personalities, including the recombination personality. Algorithm 3 summarizes ACO$_\mathbb{R}$-**PR** in pseudocode.

In the present work, we select a specific recombination operator, uniform crossover with the second parent chosen by uniform distribution. But, of course, any other recombination operator could be used in its place within the ACO$_\mathbb{R}$-**PR** algorithm.

Of course, some applications will benefit from the use of recombination more than others. ACO$_\mathbb{R}$-**PR** allows the frequency of applying recombination to adapt dynamically based on the relative quality of past solutions constructed by recombination, without the user having to externally specify a fixed probability of applying recombination.

Recombination is a fundamentally different approach to solution construction than ACO$_\mathbb{R}$'s usual solution construction mechanism, so the use of recombination promotes search diversity. The selection of the second parent by uniform distribution further promotes diversity.

It is again worth emphasizing that ACO$_\mathbb{R}$-**PR** performs the same number of fitness function evaluations per iteration as ACO$_\mathbb{R}$-**P** and ACO$_\mathbb{R}$. In each iteration, m solutions are constructed and evaluated; each solution may be constructed either by recombination with a randomly-selected population element or by ACO$_\mathbb{R}$'s usual solution construction mechanism.

B. MULTIPLE RECOMBINATION PERSONALITIES

The multiple personality approach can accommodate more than one recombination personality corresponding to different flavors of recombination. Here, we consider an approach, called ACO$_\mathbb{R}$-**PR2**, which incorporates:

Algorithm 3 Pseudo-code of $ACO_{\mathbb{R}}$-**PR**. This algorithm builds on Algorithm 2; lines that were added are marked with a plus sign next to the line number.

```
1:   algorithm ACO_R-PR($n$, $Q$)        ▷ $n$ : number of dimensions, $Q$: fitness function ($Q : R^n \mapsto R$)
2:       for $t = 1 \to I_{max}$ do
3:           if $t = 1$ OR Stagnation () then
4:               for $r = 1 \to L$ do
5:                   Initialize $s_r$ with a randomly-generated $n$-dimensional vector
6:                   Compute $Q(s_r)$
7:                   Set $\phi(s_r) = \hat{\psi}$
8:               end for
9:               Sort Archive $A$ according to $Q$
10:              Compute $u(\psi)$ according to Eq. (9) for all $\psi \in \Psi$
11:          end if
12:          for $i = 1 \to m$ do
13:              Select a personality $\psi$ according to Eq. (8)
14:              Select $s_a$ according to Eq. (1)
+15:             if $\psi$ specifies recombination then
+16:                 Select $s_b$ with uniform distribution
+17:                 Generate $s_i$ by applying recombination operator to $s_a$ and $s_b$
+18:             else                                         ▷ else: $\psi$ specifies a value of $\xi$
19:                  Set $\xi$ as specified by personality $\psi$
20:                  for $j = 1 \to n$ do
21:                      Compute $\sigma_{a,j}$ according to Eq. (5)
22:                      Generate $s_{i,j}$ according to Eq. (4)
23:                  end for
+24:             end if
25:              Compute $Q(s_i)$
26:              Set $\phi(s_i) = \psi$
27:          end for
28:          Add $s_1, \ldots, s_m$ to archive $A$
29:          Sort Archive $A$ according to $Q$
30:          Remove bottom $m$ elements from $A$
31:          Compute $u(\psi)$ according to Eq. (9) for all $\psi \in \Psi$
32:      end for
33:  return best solution in Archive $A$
34:  end algorithm
```

- the 14 ξ personalities of Eq. (6),
- a personality corresponding to recombination with uniform crossover, as described in Section IV-A,
- a personality corresponding to recombination with classical single-point crossover.

In the case of single-point crossover, the two parents are selected as in Section IV-A, one by rank-proportionate selection according to Eq. (1), and the other by uniform distribution. A crossover point is then randomly selected with a uniform distribution, and a single offspring is produced by single-point crossover. Based on our parameter settings (summarized in Table 1), the initial adoption probabilities would be 69.81% for the default personality and 1.89% for each of the other 15 personalities, including the two recombination personalities.

In the present work, we use single-point crossover as the second competing operator, but, in general, any other recombination operator could have been used.

A drawback of $ACO_{\mathbb{R}}$-**PR** is the need to specify a specific recombination operator. $ACO_{\mathbb{R}}$-**PR2** somewhat alleviates this situation by allowing two recombination operators to be specified, which then compete, with the better-suited operator being deployed more often. This can, of course, be generalized to having more than two recombination personalities specifying different competing recombination operators.

The availability of different recombination operators further promotes search diversity because each operator constructs solutions in a different way.

V. EXPERIMENTAL METHODOLOGY

Our experimental evaluation compares standard $ACO_{\mathbb{R}}$ against our proposed extensions in the context of the following two problem domains: training feedforward neural networks for classification, and optimizing several synthetic, continuous-domain benchmark functions popular in the evolutionary computation community.

A. NEURAL NETWORK TRAINING

Feedforward neural networks (FFNN) are a popular and well-established method for pattern classification. The most common FFNN architecture is a three-layer topology with full connectivity between layers. The external input to the network feeds into the first layer, called the input layer. The input layer consists of fan-out units which feed into the next layer, called the hidden layer. Finally, the output of the hidden layer feeds into the final layer, called the output layer, whose output becomes the external output of the network.

Each neuron i is a simple circuit, which receives r inputs $o_1, \ldots, o_r$, and produces a single output o_i:

$$o_i = h\left(\sum_{j=1}^{r} w_{ij} o_j + b_i\right) \tag{12}$$

where each input o_j is the output of a unit in the previous layer, w_{ij} denotes a real-valued weight between unit j and neuron i, b_i denotes a weight associated with neuron i itself, called the neuron's self-bias, and h is a nonlinear *activation function*, often chosen to be the sigmoid function:

$$h(x) = \frac{1}{1 + e^{-x}} \tag{13}$$

Note that input units do not have self-biases. After an input pattern x is presented to the network, the output of the network is observed and is referred to as the actual output vector y'. A discrepancy function E is used to compare the target output y to the actual output y', resulting in a scalar error value. A common discrepancy function is the simple sum of squared error:

$$E = \sum_{p \in P} E_p \tag{14}$$

where P is the set of patterns, and

$$E_p = \frac{1}{2} \sum_{i=1}^{k} (y_i - y_i')^2 \tag{15}$$

where k is the number of classes.

In pattern classification applications, the target vector y is k-dimensional, where k is the number of classes. For a pattern with class label $\hat{k}$:

$$y_i = \begin{cases} 1 & \text{if } i = \hat{k} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

The weights and self-biases of a given FFNN are collectively referred to as the network's *weight vector* w. For example, a FFNN with 4 neurons in the input layer, 5 neurons in the hidden layer, and 3 neurons in the output layer would have a weight vector of 43 real numbers. If the weight vector for a given network is fixed, then the output of the network is a function of its input, and the total error E of the network is a mathematical function of the dataset. On the other hand, if the dataset is fixed, then the error E is a function of the weight vector w. When training a neural network, two distinct datasets are used: a training set and a test set. While a training algorithm (such as ACO$_\mathbb{R}$ or any of our proposed variants) is running, it only has access to the training set, and seeks to minimize the error E of the network on the training set. Once the algorithm terminates, its accuracy on the previously-unseen test set is computed, and reported as the algorithm's *predictive accuracy*. ACO$_\mathbb{R}$ has previously been applied to the training of three-layer feedforward networks [60].

When applying ACO$_\mathbb{R}$ (or any of our proposed variants) to neural network training, a candidate solution consists of a value of the neural network's weight vector. In the pseudocode presented in Algorithms 1-3, n would be the length of the network's weight vector, and the fitness function Q would consist of instantiating a neural network with the weight vector under evaluation, and obtaining the training set error. Once the algorithm terminates, the best weight vector is then evaluated on the test set to obtain the algorithm's predictive accuracy.

Ideally, the number of neurons in the hidden layer should be tuned for each dataset individually. However, for convenience and standardization, we set the number of hidden neurons, in our experiments, to be the sum of the number of input neurons and output neurons.

Before being presented to the network, the dataset undergoes some preprocessing. Any duplicate instances are removed from the dataset before the partitioning into cross-validation folds (see below). Continuous attributes are scaled to the range [0, 1], and any missing values are set to the mean value for that attribute. Each categorical attribute, with c category labels, is converted to c numeric attributes, where one of the numeric attributes has a value of 1, and each of the other $(c - 1)$ attributes has a value of 0. Any missing values for a categorical attribute are set to the mode for that attribute. If the dataset has k possible classes, then the network will have k external outputs, whose target values are set according to Eq. (16).

We use 65 datasets from the University of California Irvine (UCI) Machine Learning Repository. Table 2 shows some important characteristics of these datasets.

Our experiments employ the stratified 4-fold cross-validation procedure. This means that a dataset is divided into four mutually exclusive partitions (folds), with approximately the same number of instances and roughly the same class distribution in each fold. Each algorithm under evaluation is run four times; each time, a different fold is used as the test set, and the other three are used as the training set. Because the algorithms under evaluation in this work are stochastic, the entire process is then repeated 10 times. Performance on each of the test set folds is recorded, and the average test set performance, aggregated over the four folds and 10 repetitions, is reported as representative of the performance of each algorithm under evaluation.

B. SYNTHETIC BENCHMARK FUNCTIONS

In addition to neural network training, we also evaluate our proposed algorithms on nine synthetic benchmark continuous-domain functions which are popular

TABLE 2. Characteristics of the datasets used in experimental evaluation.

Dataset	Instances	Classes	Attributes		
			Total	Numeric	Categorical
abalone	4,177	28	8	7	1
adult	48,787	2	14	6	8
annealing	886	6	38	9	29
audiology	200	24	70	0	70
automobile	205	7	25	15	10
balance	625	3	4	0	4
bcancer	272	2	9	0	9
bcancer-wisc-diag	569	2	30	30	0
bcancer-wisc-orig	397	2	8	8	0
bcancer-wisc-prog	198	2	32	32	0
biology	1,052	2	41	41	0
breast-tissue	105	6	9	9	0
car	1,728	4	6	0	6
chess	3,196	2	36	0	36
cmc	1,425	3	9	1	8
credit-australian	690	2	14	6	8
credit-german	1,000	2	20	7	13
cylinder	539	2	35	19	16
dermatology	366	6	34	1	33
ecoli	336	8	7	7	0
EEG	14,980	2	14	14	0
gesture	9,873	5	32	32	0
glass	213	7	9	9	0
GTC	2,113	3	21	21	0
haberman	289	2	3	3	0
hay	86	3	4	0	4
heart-c	303	5	13	7	6
heart-h	293	5	13	7	6
hepatitis	155	2	19	6	13
horse	356	2	22	7	15
ionosphere	350	2	34	34	0
iris	147	3	4	4	0
lenses	24	3	4	4	0
letter-r	18,668	26	16	16	0
libras	330	15	90	90	0
liver-disorders	341	2	6	6	0
lung-cancer	32	3	56	56	0
lymphography	148	4	18	3	15
mammographic	689	2	5	5	0
monks	512	2	6	0	6
mushrooms	8,124	2	22	0	22
musk	476	2	166	166	0
nursery	12,960	5	8	0	8
ozone	2,526	2	72	72	0
page-blocks	5,406	5	10	10	0
parkinsons	195	2	22	22	0
pima	768	2	8	8	0
pop	83	3	8	0	8
s-heart	270	2	13	6	7
seeds	210	3	7	7	0
segmentation	2,086	7	19	19	0
sensorless	58,509	11	48	48	0
sonar	208	2	60	60	0
soybean	305	19	35	0	35
spam	4,210	2	57	57	0
thyroid	215	3	5	5	0
transfusion	534	2	4	4	0
ttt	958	2	9	0	9
vehicle	846	4	18	18	0
vertebral-column-2c	310	2	6	6	0
vertebral-column-3c	310	3	6	6	0
voting	342	2	16	0	16
wave	5,000	3	40	40	0
wine	178	3	13	13	0
zoo	65	7	16	0	16

TABLE 3. Synthetic continuous-optimization benchmark functions used in experimental evaluation.

Function	Mathematical Representation	Search Range	Initialization Range
sphere	$f(x) = \sum_{i=1}^{d} x_i^2$	$(-100, 100)^d$	$(50, 100)^d$
Rosenbrock	$f(x) = \sum_{i=1}^{d-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$	$(-100, 100)^d$	$(15, 30)^d$
Rastrigin	$f(x) = \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$(-10, 10)^d$	$(2.56, 5.12)^d$
Griewank	$f(x) = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$(-600, 600)^d$	$(300, 600)^d$
ellipsoid	$f(x) = \sum_{i=1}^{d} \left(10^6\right)^{\frac{i-1}{d-1}} x_i^2$	$(-100, 100)^d$	$(-100, 100)^d$
Ackley	$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + e$	$(-32, 32)^d$	$(-32, 32)^d$
Weierstrass	$f(x) = \sum_{i=1}^{d}\left[\sum_{k=0}^{k_{max}} a^k \cos(2\pi b^k(x_i + 0.5))\right] - d\sum_{k=0}^{k_{max}} a^k\cos(\pi b^k)$ where $a = 0.5, b = 3, k_{max} = 20$	$(-100, 100)^d$	$(-100, 100)^d$
expanded-Schaffer	$f(x) = \sum_{i=1}^{d-1} g(x_i, x_{i+1}) + g(x_d, x_1)$ where $g(a, b) = 0.5 + \left(\sin^2(\sqrt{a^2 + b^2}) - 0.5\right) / \left(\left(1 + 0.001(a^2 + b^2)\right)^2\right)$	$(-100, 100)^d$	$(-100, 100)^d$
happycat	$f(x) = \left\lvert \sum_{i=1}^{d} x_i^2 - d\right\rvert^{1/4} + \left(0.5\sum_{i=1}^{d} x_i^2 + \sum_{i=1}^{d} x_i\right)/d + 0.5$	$(-100, 100)^d$	$(-100, 100)^d$

in the evolutionary computation community, specifically, the sphere, Rosenbrock, Rastrigin, Griewank, ellipsoid, Ackley, Weierstrass, expanded-Schaffer, and happy cat functions. Table 3 shows the definition of each of these functions, as well as each function's initialization range and search range. In our experiments, we use 10 different settings for the number of dimensions d, varying from 10 to 10000. Specifically, we use the following settings:

$$d \in \{10, 50, 100, 200, 400, 600, 800, 1000, 5000, 10000\} \tag{17}$$

Each algorithm under evaluation is applied to each of the nine functions for each of the 10 dimensionalities. In each case, the algorithm is run 100 times, and the average over the 100 runs is taken as representative of the algorithm's performance. When applying ACO$_{\mathbb{R}}$ to a benchmark function f, a candidate solution would simply consist of an assignment to a d-dimensional vector. In Algorithms 1-3, n would be equal to d, and the fitness function Q would consist of simply evaluating the formula for the function f.

C. ALGORITHMS UNDER EVALUATION

Our experimental evaluation compares the following algorithms using the parameter settings shown in Table 1:

1) Standard ACO$_{\mathbb{R}}$, treated as the control method,
2) ACO$_{\mathbb{R}}$-**P**, the multiple competing personalities approach,
3) ACO$_{\mathbb{R}}$-**PR**, which includes a single recombination personality corresponding to uniform crossover,
4) ACO$_{\mathbb{R}}$-**PR2**, which includes two recombination personalities corresponding to uniform crossover and single-point crossover,
5) ACO$_{\mathbb{R}}$-**D**, the approach presented in [40] and mentioned earlier in Section I, in which ξ is gradually reduced over time according to a fixed exponential decay schedule.

The premise of ACO$_{\mathbb{R}}$-**D** is that different values of the search width parameter ξ will be best-suited for different phases of the search process. Early in the search process, when the archive is likely populated with lower-quality solutions, larger values of ξ will favor exploration and are more appropriate, as we do not wish the search to remain confined to the vicinity of the randomly initialized candidate solutions used to seed the archive. Later, as the quality of the archived solution improves, we might wish for the search to be intensified in the vicinity of the archived solutions; thus, a smaller search width will favor exploitation and would be more appropriate. At the start of the algorithm, or after a stagnation-reset, ξ is set to an initial value ξ_0. Then, in each iteration t, ξ is reduced according to:

$$\xi_t = \xi_{t-1} \cdot e \tag{18}$$

where $0 < e < 1$ is a constant that is determined based on ξ_0, ξ', and I_{max}, as follows:

$$e = \left(\frac{\xi'}{\xi_0}\right)^{\frac{1}{I_{max}}} \tag{19}$$

TABLE 4. **Predictive accuracy (%) results for neural network training.**

	ACO$_{\mathbb{R}}$	ACO$_{\mathbb{R}}$-**P**	ACO$_{\mathbb{R}}$-**PR**	ACO$_{\mathbb{R}}$-**PR2**	ACO$_{\mathbb{R}}$-**D**
`abalone`	13.65	**17.93**	17.42	16.69	12.86
`adult`	83.01	84.58	84.64	84.64	**84.73**
`annealing`	71.73	**75.79**	74.72	75.45	66.89
`audiology`	28.82	51.47	**52.01**	46.40	37.60
`automobile`	59.14	60.22	61.94	61.22	**63.45**
`balance`	91.11	91.25	**91.39**	91.35	91.20
`bcancer`	73.20	73.46	**73.76**	73.64	73.13
`bcancer-wisc-diag`	88.37	**90.74**	89.95	90.67	87.47
`bcancer-wisc-orig`	90.41	92.22	92.48	**92.58**	91.37
`bcancer-wisc-prog`	71.98	73.13	72.91	**74.49**	71.23
`biology`	83.01	84.30	**84.71**	83.92	84.63
`breast-tissue`	**62.17**	59.82	59.12	57.62	61.87
`car`	**92.99**	88.91	88.81	88.62	92.06
`chess`	94.05	95.29	95.12	95.08	**96.57**
`cmc`	53.81	54.34	**55.20**	54.73	54.95
`credit-australian`	86.29	86.52	**86.55**	86.37	86.19
`credit-german`	77.54	77.77	**77.86**	77.54	77.64
`cylinder`	68.80	71.04	**71.62**	70.58	71.46
`dermatology`	88.31	95.22	**95.41**	94.54	92.59
`ecoli`	81.66	78.39	78.46	75.63	**85.35**
`EEG`	51.28	52.00	52.25	52.12	**52.40**
`gesture`	40.90	**44.80**	44.31	44.12	41.87
`glass`	54.61	53.23	53.54	54.11	**55.61**
`GTC`	88.83	88.84	88.73	88.73	**89.42**
`haberman`	70.33	72.83	**73.17**	73.10	70.88
`hay`	76.78	**80.23**	79.75	79.67	78.94
`heart-c`	62.39	62.37	**63.13**	62.20	62.30
`heart-h`	64.92	64.55	**65.17**	62.79	62.92
`hepatitis`	85.79	**86.43**	85.15	85.01	86.30
`horse`	79.50	79.74	79.69	**80.21**	79.72
`ionosphere`	90.98	91.22	90.79	91.05	**92.25**
`iris`	94.41	93.48	94.00	93.79	**94.67**
`lenses`	76.29	78.83	**79.69**	79.49	78.30
`letter-r`	9.30	14.85	14.26	13.23	**15.03**
`libras`	23.40	47.86	**48.79**	45.69	46.43
`liver-disorders`	67.65	**69.02**	68.82	68.28	68.74
`lung-cancer`	**47.09**	45.46	41.46	45.45	44.33
`lymphography`	83.05	86.16	85.74	85.96	**86.71**
`mammographic`	51.42	52.79	52.73	**54.41**	50.83
`monks`	76.69	66.41	66.82	65.22	**76.94**
`mushrooms`	99.58	**99.84**	99.80	99.74	99.79
`musk`	72.26	79.08	78.30	78.41	**79.69**
`nursery`	93.11	92.10	91.94	91.62	**93.22**
`ozone`	93.63	93.75	**93.76**	**93.76**	93.57
`page-blocks`	92.42	90.99	**93.42**	93.12	93.18
`parkinsons`	84.03	83.72	83.79	**84.30**	82.76
`pima`	74.11	**74.57**	74.14	74.54	73.63
`pop`	54.99	**62.27**	60.40	60.53	59.33
`s-heart`	82.26	83.41	83.07	**83.93**	82.75
`seeds`	**93.39**	92.33	91.35	91.79	92.89
`segmentation`	83.10	79.31	80.38	77.78	**87.72**
`sensorless`	14.88	18.27	**18.29**	18.21	17.56
`sonar`	77.47	78.72	79.39	79.38	**80.21**
`soybean`	42.62	**63.82**	63.73	60.52	55.58
`spam`	89.75	91.83	**92.32**	92.15	91.88
`thyroid`	**95.29**	92.10	92.77	91.16	94.93
`transfusion`	71.48	**72.90**	72.41	72.71	72.21
`ttt`	92.52	86.20	86.13	85.67	**92.56**
`vehicle`	62.56	59.98	59.14	58.67	**65.58**
`vertebral-column-2c`	74.35	76.48	77.00	**77.60**	75.32
`vertebral-column-3c`	57.92	58.38	59.78	**59.96**	57.08
`voting`	95.14	95.67	95.58	**95.81**	95.12
`wave`	76.55	80.70	81.13	80.54	**82.84**
`wine`	96.74	**96.98**	**96.98**	96.82	96.52
`zoo`	86.90	89.86	**92.05**	91.96	86.83
#wins	5	13	20	10	19
rank (avg)	3.88	2.59	2.46	3.09	2.97

TABLE 5. Results for the synthetic continuous optimization benchmark functions.

function	d	ACO$_\mathbb{R}$	ACO$_\mathbb{R}$-**P**	ACO$_\mathbb{R}$-**PR**	ACO$_\mathbb{R}$-**PR2**	ACO$_\mathbb{R}$-**D**
sphere	10	4.25E+03	9.40E-108	1.17E-104	6.54E-104	**4.27E-112**
	50	2.51E+04	874.851	906.616	1.74E+03	**0.008**
	100	5.57E+04	3.43E+04	3.31E+04	4.09E+04	**4.59E+03**
	200	1.81E+05	2.42E+05	2.45E+05	2.61E+05	**1.09E+05**
	400	**5.83E+05**	9.29E+05	9.24E+05	9.56E+05	6.57E+05
	600	**1.35E+06**	1.76E+06	1.77E+06	1.81E+06	1.43E+06
	800	**2.21E+06**	2.69E+06	2.68E+06	2.74E+06	2.29E+06
	1,000	3.42E+06	3.63E+06	3.63E+06	3.67E+06	**3.23E+06**
	5,000	2.48E+07	2.46E+07	2.46E+07	2.47E+07	**2.45E+07**
	10,000	5.26E+07	5.19E+07	**5.17E+07**	5.18E+07	5.20E+07
rosenbrock	10	1.19E+07	53.824	58.939	59.512	**22.416**
	50	1.10E+08	1.05E+06	1.96E+06	2.37E+06	**450.405**
	100	2.41E+08	8.68E+07	9.07E+07	1.16E+08	**1.34E+06**
	200	6.07E+08	9.05E+08	8.80E+08	1.01E+09	**2.43E+08**
	400	**2.34E+09**	4.15E+09	4.01E+09	4.25E+09	2.58E+09
	600	6.56E+09	8.38E+09	7.97E+09	8.34E+09	**6.50E+09**
	800	1.31E+10	1.29E+10	1.24E+10	1.27E+10	**1.14E+10**
	1,000	2.40E+10	1.78E+10	**1.69E+10**	1.75E+10	1.98E+10
	5,000	1.41E+11	1.27E+11	**1.18E+11**	1.22E+11	1.41E+11
	10,000	2.84E+11	2.69E+11	**2.53E+11**	2.59E+11	2.84E+11
rastrigin	10	39.797	40.406	36.541	42.444	**23.328**
	50	339.488	507.592	489.820	518.225	**277.785**
	100	797.043	1.25E+03	1.17E+03	1.23E+03	**728.932**
	200	2.13E+03	3.03E+03	2.82E+03	3.01E+03	**2.06E+03**
	400	5.86E+03	7.16E+03	6.68E+03	7.12E+03	**5.76E+03**
	600	1.22E+04	1.15E+04	**1.08E+04**	1.14E+04	1.09E+04
	800	1.83E+04	1.60E+04	**1.51E+04**	1.59E+04	1.79E+04
	1,000	2.37E+04	2.07E+04	**1.95E+04**	2.03E+04	2.34E+04
	5,000	1.21E+05	1.21E+05	**1.12E+05**	1.14E+05	1.21E+05
	10,000	2.44E+05	2.44E+05	**2.31E+05**	2.34E+05	2.44E+05
griewank	10	39.911	0.037	0.041	0.042	**0.014**
	50	226.744	10.746	14.827	17.064	**0.076**
	100	504.322	324.123	284.274	347.190	**42.950**
	200	1.63E+03	2.18E+03	2.23E+03	2.35E+03	**981.648**
	400	**5.15E+03**	8.36E+03	8.42E+03	8.69E+03	5.98E+03
	600	**1.22E+04**	1.60E+04	1.59E+04	1.65E+04	1.29E+04
	800	**2.00E+04**	2.42E+04	2.42E+04	2.46E+04	2.06E+04
	1,000	3.08E+04	3.26E+04	3.24E+04	3.31E+04	**2.90E+04**
	5,000	2.23E+05	2.22E+05	2.22E+05	2.22E+05	**2.20E+05**
	10,000	4.74E+05	4.68E+05	**4.67E+05**	4.67E+05	4.68E+05
ellipsoid	10	1.78E+07	1.09E-102	1.61E-102	5.28E-100	**2.89E-106**
	50	4.57E+08	3.78E+05	3.48E+05	5.78E+05	**1.53E+04**
	100	1.24E+09	3.48E+07	3.84E+07	4.02E+07	**9.18E+06**
	200	3.28E+09	6.58E+08	6.77E+08	7.07E+08	**2.30E+08**
	400	**2.30E+09**	5.96E+09	5.68E+09	6.07E+09	2.90E+09
	600	**7.84E+09**	1.64E+10	1.60E+10	1.58E+10	1.00E+10
	800	**1.74E+10**	3.14E+10	3.11E+10	2.97E+10	2.10E+10
	1,000	4.71E+10	5.10E+10	4.89E+10	4.94E+10	**3.68E+10**
	5,000	7.36E+11	6.80E+11	6.68E+11	**6.62E+11**	7.13E+11
	10,000	2.23E+12	1.64E+12	**1.62E+12**	1.62E+12	2.20E+12
ackley	10	1.980	0.177	0.267	0.190	**3.62E-15**
	50	4.111	14.226	13.936	14.078	**4.036**
	100	**10.505**	18.185	17.900	18.261	13.438
	200	19.058	19.397	19.293	19.560	**18.721**
	400	20.880	20.097	**20.021**	20.233	20.706
	600	20.969	20.490	**20.310**	20.484	20.944
	800	21.004	20.747	**20.453**	20.592	20.995
	1,000	21.037	20.835	**20.550**	20.657	21.015
	5,000	21.092	21.058	**20.895**	20.944	21.090
	10,000	21.105	21.073	**20.970**	21.001	21.104
weierstrass	10	0.467	0.056	**2.08E-12**	0.302	0.032
	50	2.55E+03	2.02E+03	**520.135**	779.743	2.59E+03
	100	6.45E+03	5.23E+03	**2.09E+03**	2.77E+03	6.50E+03
	200	1.46E+04	1.21E+04	**6.45E+03**	7.74E+03	1.46E+04
	400	3.09E+04	2.66E+04	**1.69E+04**	1.99E+04	3.11E+04
	600	4.78E+04	4.15E+04	**2.80E+04**	3.28E+04	4.81E+04
	800	6.48E+04	5.64E+04	**3.90E+04**	4.70E+04	6.39E+04
	1,000	7.90E+04	6.92E+04	**5.22E+04**	6.03E+04	8.08E+04
	5,000	4.07E+05	3.53E+05	**3.10E+05**	3.41E+05	3.99E+05
	10,000	7.89E+05	6.74E+05	**6.09E+05**	6.62E+05	7.58E+05
ex-schaffer	10	110.821	88.564	**69.655**	89.109	93.924
	50	1.11E+03	1.06E+03	**840.724**	968.740	1.05E+03
	100	2.34E+03	2.27E+03	**1.94E+03**	2.12E+03	2.24E+03
	200	4.79E+03	4.68E+03	**4.17E+03**	4.55E+03	4.60E+03
	400	9.74E+03	9.53E+03	**8.72E+03**	9.39E+03	9.34E+03
	600	1.47E+04	1.44E+04	**1.33E+04**	1.43E+04	1.41E+04
	800	1.96E+04	1.92E+04	**1.80E+04**	1.92E+04	1.88E+04
	1,000	2.46E+04	2.41E+04	**2.26E+04**	2.41E+04	2.36E+04
	5,000	1.24E+05	1.22E+05	**1.17E+05**	1.23E+05	1.19E+05
	10,000	2.49E+05	2.39E+05	**2.35E+05**	2.47E+05	2.39E+05
happycat	10	9.375	10.405	9.464	8.626	**8.315**
	50	32.070	245.398	145.200	168.138	**28.197**
	100	**154.685**	2.71E+03	2.25E+03	3.11E+03	208.430
	200	**4.09E+03**	1.19E+04	1.27E+04	1.40E+04	6.10E+03
	400	**2.12E+04**	2.77E+04	2.65E+04	2.89E+04	2.36E+04
	600	3.67E+04	3.60E+04	3.55E+04	3.79E+04	**3.55E+04**
	800	4.87E+04	4.18E+04	**4.10E+04**	4.34E+04	4.47E+04
	1,000	6.71E+04	4.58E+04	**4.40E+04**	4.75E+04	5.41E+04
	5,000	8.16E+04	**6.34E+04**	6.36E+04	6.67E+04	7.84E+04
	10,000	8.14E+04	**6.74E+04**	6.79E+04	7.20E+04	7.97E+04
#wins		14	2	39	1	34
rank		3.76	3.27	2.14	3.42	2.41

TABLE 6. Pairwise comparison of each of the four proposed algorithms against the control method (ACO$_\mathbb{R}$).

	NN			Benchmarks		
	win	loss	rank	win	loss	rank
ACO$_\mathbb{R}$-**P** versus ACO$_\mathbb{R}$	48	17	1.262	58	32	1.356
ACO$_\mathbb{R}$-**PR** versus ACO$_\mathbb{R}$	48	17	1.262	61	29	1.322
ACO$_\mathbb{R}$-**PR2** versus ACO$_\mathbb{R}$	47	17	1.269	60	30	1.333
ACO$_\mathbb{R}$-**D** versus ACO$_\mathbb{R}$	44	21	1.323	69	21	1.233

TABLE 7. Results of Wilcoxon signed-rank tests, with the Holm-Bonferroni correction, for the neural network training results, comparing each algorithm to the control method (ACO$_\mathbb{R}$).

Comparison	p	Holm	sig.?
ACO$_\mathbb{R}$-**PR2** versus ACO$_\mathbb{R}$	0.0029	0.05	yes
ACO$_\mathbb{R}$-**P** versus ACO$_\mathbb{R}$	0.0017	0.025	yes
ACO$_\mathbb{R}$-**PR** versus ACO$_\mathbb{R}$	0.0012	0.01666	yes
ACO$_\mathbb{R}$-**D** versus ACO$_\mathbb{R}$	8.2E-05	0.0125	yes

TABLE 8. Results of Wilcoxon signed-rank tests, with the Holm-Bonferroni correction, for the synthetic benchmark results, comparing each algorithm to the control method (ACO$_\mathbb{R}$).

Comparison	p	Holm	sig.?
ACO$_\mathbb{R}$-**P** versus ACO$_\mathbb{R}$	0.0220	0.05	yes
ACO$_\mathbb{R}$-**PR2** versus ACO$_\mathbb{R}$	0.0151	0.025	yes
ACO$_\mathbb{R}$-**PR** versus ACO$_\mathbb{R}$	0.0044	0.01666	yes
ACO$_\mathbb{R}$-**D** versus ACO$_\mathbb{R}$	1.4E-05	0.0125	yes

where ξ_0 and ξ' are two algorithm parameters, and I_{max} is the maximum number of iterations. In [40], ACO$_\mathbb{R}$-**D** was evaluated on the problem of neural network training using 36 datasets. In the present work, we extend the previously reported results for ACO$_\mathbb{R}$-**D** on neural network training by increasing the number of datasets to 65, and also evaluate ACO$_\mathbb{R}$-**D** on the benchmark continuous optimization problems.

Although the primary purpose of our experimental evaluation is to determine the extent to which our proposed algorithms improve on ACO$_\mathbb{R}$, we also carry out a follow-up experiment in which we compare the best performing of our proposed variations to two widely-used non-ACO algorithms. For neural network training, we compare performance on a subset of the datasets to the popular Back-Propagation (BP) learning algorithm. For the synthetic benchmark functions, we compare performance on a subset of the functions and a subset of the dimensionalities to cNrGA [66], a recent state-of-the-art genetic algorithm.

VI. EXPERIMENTAL RESULTS

Table 4 shows the average test set predictive accuracy for the algorithms under evaluation for each of the datasets, and Table 5 shows the average results for the algorithms for the benchmark functions for each value of the number of dimensions. In both tables, the best result in each row is shown in boldface. The penultimate row reports the number of wins for each algorithm, i.e. the number of cases for which each method had, or tied for, the best performance (where a case is a dataset when dealing with neural network training and is

TABLE 9. Neural Network Predictive Accuracy (%) Results for ACO$_\mathbb{R}$ and ACO$_\mathbb{R}$-**PR**, compared to BP.

dataset	ACO$_\mathbb{R}$	ACO$_\mathbb{R}$-**PR**	BP
annealing	71.73	**74.72**	67.41
balance	91.11	91.39	**96.16**
bcancer-wisc-diag	88.37	89.95	**93.86**
car	**92.99**	88.81	90.70
chess	94.05	**95.12**	92.61
credit-a	86.29	**86.55**	84.35
credit-g	77.54	**77.86**	72.20
dermatology	88.31	**95.41**	80.68
ecoli	**81.66**	78.46	79.53
nursery	**93.11**	91.94	91.15
parkinsons	**84.03**	83.79	79.94
pima	74.11	**74.14**	73.82
segmentation	83.10	80.38	**93.01**
soybean	42.62	**63.73**	57.58
transfusion	71.48	**72.41**	70.56
vehicle	62.56	59.14	**64.21**
vertebral-column-2c	74.35	77.00	**79.35**
vertebral-column-3c	57.92	59.78	**68.06**
voting	95.14	**95.58**	93.89
zoo	86.90	**92.05**	81.25
#wins	4	10	6
rank (avg)	2.05	1.70	2.25

TABLE 10. Results for the synthetic continuous optimization benchmark functions for ACO$_\mathbb{R}$ and ACO$_\mathbb{R}$-**PR**, compared to cNrGA.

function	*d*	ACO$_\mathbb{R}$	ACO$_\mathbb{R}$-**PR**	cNrGA
sphere	10	4.25E+03	**1.17E-104**	0.747
	50	2.51E+04	906.616	**104.190**
	100	5.57E+04	3.31E+04	**1.41E+03**
	200	1.81E+05	2.45E+05	**1.07E+04**
rosenbrock	10	1.19E+07	**58.939**	7.41E+05
	50	1.10E+08	**1.96E+06**	5.17E+06
	100	2.41E+08	**9.07E+07**	1.64E+08
	200	**6.07E+08**	8.80E+08	9.14E+08
griewank	10	39.911	**0.041**	0.766
	50	226.744	**14.827**	39.297
	100	504.322	**284.274**	511.061
	200	**1.63E+03**	2.23E+03	3.29E+03
ellipsoid	10	1.78E+07	**1.61E-102**	2.31E+03
	50	4.57E+08	**3.48E+05**	5.99E+06
	100	1.24E+09	**3.84E+07**	7.24E+07
	200	3.28E+09	6.77E+08	**5.34E+08**
weierstrass	10	0.467	**2.08E-12**	0.304
	50	2.55E+03	520.135	**7.285**
	100	6.45E+03	2.09E+03	**28.668**
	200	1.46E+04	6.45E+03	**95.403**
happycat	10	9.375	9.464	**0.461**
	50	32.070	145.200	**4.175**
	100	154.685	2.25E+03	**13.975**
	200	4.09E+03	1.27E+04	**51.550**
#wins		2	11	11
rank (avg)		2.58	1.75	1.67

a function-dimension pair when dealing with the benchmark functions). The final row reports the average rank for each algorithm. The average rank for a given algorithm *a* for a given set of cases is obtained by first computing the rank of *a* on each case individually, and then averaging the individual ranks across all cases. Note that the lower the value of the rank, the better the algorithm.

For both application domains, the best average rank is obtained by ACO$_\mathbb{R}$-**PR**; however, all four of our proposed algorithms have a better average rank than the control algorithm (standard ACO$_\mathbb{R}$).

ACO$_\mathbb{R}$ is an established algorithm, and we are treating it as the control method; thus, it is interesting to compare each algorithm to it, in isolation from the others. Table 6 shows pair-wise comparisons between each of the four proposed algorithms against the control method (ACO$_\mathbb{R}$). In each comparison, the table reports the number of wins (i.e. the number of cases in which the non-control algorithm performed better), the number of losses (i.e. the number of cases in which the control algorithm performed better), and the average rank for the non-control algorithm. Note that in the pair-wise comparisons, each of the proposed algorithms performs better than the control method in terms of both the number of wins and the average rank, for both application domains.

We analyze the statistical significance of the neural network results as follows. We apply a (two-tailed) Wilcoxon signed-rank test, at the 0.05 threshold, to compare ACO$_\mathbb{R}$, as the baseline method, to each of the four other methods, using the Holm-Bonferroni correction to control the family-wise error, and we report these results in Table 7. We also apply the same methodology to analyze the statistical significance of the results on the continuous optimization benchmark problems, and we report the results in Table 8. Tables 7-8 indicate that all four proposed algorithms perform better than the baseline ACO$_\mathbb{R}$ to a statistically significant extent, on both application domains.

As mentioned in Sect. V-C, we carry out a follow-up experiment in which we compare performance to two popular non-ACO algorithms. For neural network training, we compare the performance of ACO$_\mathbb{R}$ and ACO$_\mathbb{R}$-**PR** (the best performing of our proposed variations) to BP on a subset of the datasets (specifically the 20 datasets listed in Table 9). These results are shown in Table 9, and indicate that the best performance is obtained by ACO$_\mathbb{R}$-**PR** with 10 wins and an average rank of 1.7, versus 6 wins and an average rank of 2.25 for BP. For the synthetic benchmark functions, we compare performance to cNrGA on 24 of the function-dimensionality pairings (as listed in Table 10). Table 10 indicates that cNrGA and ACO$_\mathbb{R}$-**PR** have comparable performance. Both algorithms have 11 wins each (with 2 wins for ACO$_\mathbb{R}$), but cNrGA has a slightly better average rank (with an average rank of 1.67 versus 1.75 for ACO$_\mathbb{R}$-**PR**).

VII. CONCLUSION & FUTURE WORK DIRECTIONS

A. CONCLUSION

We have presented an approach for the dynamic adaptation of the ACO$_\mathbb{R}$ algorithm's control parameters, focusing in the present work on the search width parameter ξ, although our approach can be generalized to adapt other parameters as well, as discussed in Section VII-B.

Our proposed ACO$_\mathbb{R}$-**P** algorithm, detailed in Section III, is based on the idea of multiple competing personalities.

Specifically, a fixed set of personalities is pre-specified, where each personality is a full or partial specification of the algorithm's parameters. The probability that a given personality will be used to generate a solution depends on the success rate of other solutions previously generated by the same personality. In the present work, each personality specifies a value of ξ.

In the ACO$_{\mathbb{R}}$-**PR** proposal presented in Section IV, ACO$_{\mathbb{R}}$-**P** is generalized to allow each personality to either specify a value of ξ or to specify that recombination should be employed instead of ACO$_{\mathbb{R}}$'s usual solution construction mechanism. In the ACO$_{\mathbb{R}}$-**PR2** proposal, we allow two personalities to specify that recombination should be employed, with each personality specifying a different type of recombination operator.

We also include extended results for the ACO$_{\mathbb{R}}$-**D** proposal, which was presented in [40] and is a simpler alternative to ACO$_{\mathbb{R}}$-**P**; in ACO$_{\mathbb{R}}$-**D**, ξ starts off at a specified value and then gradually decays over time, similar to inertial decay in PSO. In that previous work, results were reported for ACO$_{\mathbb{R}}$-**D** for neural network training on 36 datasets. In the present work, extended results were reported for neural network training on 65 datasets, as well as for the synthetic benchmark functions.

All four approaches (ACO$_{\mathbb{R}}$-**P**, ACO$_{\mathbb{R}}$-**PR**, ACO$_{\mathbb{R}}$-**PR**, and ACO$_{\mathbb{R}}$-**PR2**) perform better than standard ACO$_{\mathbb{R}}$ to a statistically significant extent on both neural network training and the benchmark continuous optimization functions.

B. FUTURE WORK DIRECTIONS

Although we have focused on search width in the present work, the multiple personalities approach can be generalized to the other parameters as well. Let us consider how the multiple personalities approach might be generalized to adapt the q and L parameters, in addition to the ξ parameter. In this case, three sets of personalities would be defined, each specifying values for one of the three parameters. The utility of each personality (whether a ξ-personality, q-personality, or L-personality) would be judged separately by its past performance, and the roulette wheel equation (Eq. 8) would be applied three times, once for each of the three parameters. When adapting the archive size L, the physical size of the archive would be the largest value of L in any of the specified personalities, but the logical value of L would be limited to the size specified by the adopted personality. Thus, if we have a physical archive size of 1000, but the adopted personality specifies $L = 120$, then only the first 120 solutions in the sorted archive would be considered in solution generation.

For the sake of computational efficiency, the ω function (Eq. 2) should be pre-computed and pre-stored for each pairing of settings of q and L. For example, if we have 15 q personalities and 10 L personalities, then we would pre-compute 150 ω tables before the start of the computation.

Making ACO$_{\mathbb{R}}$'s parameters more self-adaptive in this way can make the algorithm more responsive to the different phases of the search, and, by enhancing search diversity, can potentially help avoid premature convergence and search stagnation.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their useful comments.

REFERENCES

[1] T. Liao, K. Socha, M. A. Montes de Oca, T. Stützle, and M. Dorigo, "Ant colony optimization for mixed-variable optimization problems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 503–518, Aug. 2014.

[2] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004.

[3] A. Eiben, Z. Michalewicz, M. Schoenauer, and J. Smith, "Parameter control in evolutionary algorithms," in *Parameter Setting in Evolutionary Algorithms*, F. J. Lobo, C. F. Lima, and Z. Michalewicz, Eds. Berlin, Germany: Springer, 2007, pp. 19–46.

[4] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *J. Artif. Intell. Res.*, vol. 36, pp. 267–306, Oct. 2009.

[5] P. Lin, J. Zhang, and M. A. Contreras, "Automatically configuring ACO using multilevel ParamILS to solve transportation planning problems with underlying weighted networks," *Swarm Evol. Comput.*, vol. 20, pp. 48–57, Feb. 2015.

[6] F. J. Lobo, C. F. Lima, and Z. Michalewicz, *Parameter Setting in Evolutionary Algorithms*. Berlin, Germany: Springer, 2007.

[7] P. Pellegrini, M. Birattari, and T. Stützle, "A critical analysis of parameter adaptation in ant colony optimization," *Swarm Intell.*, vol. 6, pp. 23–48, Mar. 2012.

[8] T. Stützle *et al.*, "Parameter adaptation in ant colony optimization," in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds. Berlin, Germany: Springer, 2012, pp. 191–215.

[9] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2002, pp. 11–18.

[10] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-race and iterated F-race: An overview," in *Experimental Methods for the Analysis of Optimization Algorithms*, T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, Eds. Berlin, Germany: Springer, 2010, pp. 311–316.

[11] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in *Principles and Practice of Constraint Programming* (Lecture Notes in Computer Science), vol. 5732. Berlin, Germany: Springer, 2009, pp. 142–157.

[12] A. S. Fukunaga, "Automated discovery of local search heuristics for satisfiability testing," *Evol. Comput.*, vol. 16, no. 1, p. 31–61, 2008.

[13] M. Oltean, "Evolving evolutionary algorithms using linear genetic programming," *Evol. Comput.*, vol. 13, no. 3, pp. 387–410, 2005.

[14] K. M. Salama, A. M. Abdelbar, A. M. Helal, and A. A. Freitas, "Instance-based classification with ant colony optimization," *Intell. Data Anal.*, vol. 21, no. 4, pp. 913–941, 2017.

[15] K. M. Salama, A. M. Abdelbar, and I. M. Anwar, "Data reduction for classification with ant colony algorithms," *Intell. Data Anal.*, vol. 20, no. 5, pp. 1021–1059, 2016.

[16] K. M. Salama and A. M. Abdelbar, "Using ant colony optimization to build cluster-based classification systems," in *Proc. Int. Conf. Swarm Intell. (ANTS)*, 2016, pp. 210–221.

[17] T. Liao, M. A. Montes de Oca, D. Aydin, T. Stützle, and M. Dorigo, "An incremental ant colony algorithm with local search for continuous optimization," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2011, pp. 125–132.

[18] D. Merkle and M. Middendorf, "Prospects for dynamic algorithm control: Lessons from the phase structure of ant scheduling algorithms," in *Proc. GECCO Workshop Next Ten Years Scheduling Res.*, 2001, pp. 121–126.

[19] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 333–346, Aug. 2002.

[20] B. Meyer, "Convergence control in ACO," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2004, pp. 1–12.

[21] Z. Cai, H. Huang, Y. Qin, and X. Ma, "Ant colony optimization based on adaptive volatility rate of pheromone trail," *Int. J. Commun., Netw. Syst. Sci.*, vol. 2, no. 8, pp. 792–796, 2009.

[22] S. Chusanapiputt, D. Nualhong, S. Jantarang, and S. Phoomvuthisarn, "Selective self-adaptive approach to ant system for solving unit commitment problem," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2006, pp. 1729–1736.

[23] Z. Hao, H. Huang, Y. Qin, and R. Cai, "An ACO algorithm with adaptive volatility rate of pheromone trail," in *Proc. Int. Conf. Comput. Sci.*, 2007, pp. 1167–1170.

[24] O. Kovářík and M. Skrbek, "Ant colony optimization with castes," in *Proc. Int. Conf. Artif. Neural Netw.*, 2008, pp. 435–442.

[25] Y. Li and W. Li, "Adaptive ant colony optimization algorithm based on information entropy: Foundation and application," *Fundamenta Informaticae*, vol. 77, no. 3, pp. 229–242, 2007.

[26] Z. Li, Y. Wang, J. Yu, Y. Zhang, and X. Li, "A novel cloud-based fuzzy self-adaptive ant colony system," in *Proc. Int. Conf. Natural Comput.*, vol. 7, Oct. 2008, pp. 460–465.

[27] M. Randall and J. Montgomery, "Candidate set strategies for ant colony optimisation," in *Proc. Int. Conf. Swarm Intell. (ANTS)*, 2002, pp. 374–381.

[28] M. Randall, "Near parameter free ant colony optimisation," in *Proc. Int. Conf. Swarm Intell. (ANTS)*, 2004, pp. 374–381.

[29] D. Martens, M. D. Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens, "Classification with ant colony optimization," *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 651–665, Oct. 2007.

[30] M. Förster, B. Bickel, B. Hardung, and G. Kókai, "Self-adaptive ant colony optimisation applied to function allocation in vehicle networks," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2007, pp. 1991–1998.

[31] M. Khichane, P. Albert, and C. Solnon, "An ACO-based reactive framework for ant colony optimization: First experiments on constraint satisfaction problems," in *Proc. Int. Conf. Learn. Intell. Optim.*, 2009, pp. 119–133.

[32] M. Pilat and T. White, "Using genetic algorithms to optimize ACO-TSP," in *Proc. Int. Conf. Swarm Intell. (ANTS)*, 2002, pp. 282–287.

[33] D. Gaertner and K. L. Clark, "On optimal parameters for ant colony optimization algorithms," in *Proc. Int. Conf. Artif. Intell.*, 2005, pp. 83–89.

[34] B. A. Garro, H. Sossa, and R. A. Vazquez, "Evolving ant colony system for optimizing path planning in mobile robots," in *Proc. Electron., Robot. Automot. Mech. Conf.*, Sep. 2007, pp. 444–449.

[35] Z.-F. Hao, R.-C. Cai, and H. Huang, "An adaptive parameter control strategy for ACO," in *Proc. Int. Conf. Mach. Learn.*, Aug. 2006, pp. 203–206.

[36] W. Ling and H. Luo, "An adaptive parameter control strategy for ant colony optimization," in *Proc. Int. Conf. Comput. Intell. Secur.*, Dec. 2007, pp. 142–146.

[37] D. Anghinolfi, A. Boccalatte, M. Paolucci, and C. Vecchiola, "Performance evaluation of an adaptive ant colony optimization applied to single machine scheduling," in *Proc. Int. Conf. Simulated Evol. Learn.*, 2008, pp. 411–420.

[38] L. Melo, F. Pereira, and E. Costa, "MC-Ant: A multi-colony ant algorithm," in *Proc. Int. Conf. Artif. Evol.*, 2009, pp. 25–36.

[39] A. M. Abdelbar and K. M. Salama, "Does the ACO$_\mathbb{R}$ algorithm benefit from the use of crossover?" in *Proc. Int. Conf. Swarm Intell. (ANTS)*, 2018, pp. 342–350.

[40] A. M. Abdelbar and K. M. Salama, "An extension of the ACO$_\mathbb{R}$ algorithm with time-decaying search width, with application to neural network training," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2016, pp. 2360–2366.

[41] I. D. I. D. Ariyasingha and T. G. I. Fernando, "Performance analysis of the multi-objective ant colony optimization algorithms for the traveling salesman problem," *Swarm Evol. Comput.*, vol. 23, pp. 11–26, Aug. 2015.

[42] C. Mao, L. Xiao, X. Yu, and J. Chen, "Adapting ant colony optimization to generate test data for software structural testing," *Swarm Evol. Comput.*, vol. 20, pp. 23–36, Feb. 2015.

[43] K. V. Narasimha, E. Kivelevitch, B. Sharma, and M. Kumar, "An ant colony optimization technique for solving min–max multi-depot vehicle routing problem," *Swarm Evol. Comput.*, vol. 13, pp. 63–73, Dec. 2013.

[44] A. Swarnkar, N. Gupta, and K. R. Niazi, "Adapted ant colony optimization for efficient reconfiguration of balanced and unbalanced distribution systems for loss minimization," *Swarm Evol. Comput.*, vol. 1, pp. 129–137, Sep. 2011.

[45] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 321–332, Aug. 2002.

[46] K. M. Salama, A. M. Abdelbar, and A. A. Freitas, "Multiple pheromone types and other extensions to the Ant-Miner classification rule discovery algorithm," *Swarm Intell.*, vol. 5, nos. 3–4, pp. 149–182, 2011.

[47] K. M. Salama, A. M. Abdelbar, F. E. B. Otero, and A. A. Freitas, "Utilizing multiple pheromones in an ant-based algorithm for continuous-attribute classification rule discovery," *Appl. Soft Comput.*, vol. 13, no. 1, pp. 667–675, 2013.

[48] F. E. B. Otero, A. A. Freitas, and C. G. Johnson, "A new sequential covering strategy for inducing classification rules with ant colony algorithms," *IEEE Trans. Evol. Comput.*, vol. 17, no. 1, pp. 64–76, Feb. 2013.

[49] F. E. B. Otero and A. A. Freitas, "Improving the interpretability of classification rules discovered by an ant colony algorithm: Extended results," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, 2013, pp. 73–80.

[50] N. K. Sreeja and A. Sankar, "A hierarchical heterogeneous ant colony optimization based approach for efficient action rule mining," *Swarm Evol. Comput.*, vol. 29, pp. 1–12, Aug. 2016.

[51] U. Boryczka and J. Kozak, "An adaptive discretization in the ACDT algorithm for continuous attributes," in *Proc. Int. Conf. Comput. Collective Intell. (ICCI)*, 2011, pp. 475–484.

[52] F. E. B. Otero, A. A. Freitas, and C. G. Johnson, "Inducing decision trees with an ant colony optimization algorithm," *Appl. Soft Comput.*, vol. 12, no. 11, pp. 3615–3626, 2012.

[53] K. M. Salama and A. A. Freitas, "Learning Bayesian network classifiers using ant colony optimization," *Swarm Intell.*, vol. 7, no. 2, pp. 229–254, 2013.

[54] K. M. Salama and A. A. Freitas, "Ant colony algorithms for constructing Bayesian multi-net classifiers," *Intell. Data Anal.*, vol. 19, no. 2, pp. 233–257, 2015.

[55] K. M. Salama and A. A. Freitas, "Clustering-based Bayesian multi-net classifier construction with ant colony optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2013, pp. 3079–3086.

[56] K. M. Salama and A. A. Freitas, "Extending the ABC-miner Bayesian classification algorithm," in *Proc. Int. Workshop Nature Inspired Cooperation Strategies Optim. (NICSO)*, 2013, pp. 1–12.

[57] K. M. Salama and A. A. Freitas, "Classification with cluster-based Bayesian multi-nets using ant colony optimisation," *Swarm Evol. Comput.*, vol. 18, pp. 54–70, Oct. 2014.

[58] M. Dorigo and T. Stützle, "Ant colony optimization: Overview and recent advances," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. New York, NY, USA: Springer, 2010, pp. 227–263.

[59] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *Eur. J. Oper. Res.*, vol. 185, no. 3, pp. 1155–1173, 2008.

[60] K. Socha and C. Blum, "An ant colony optimization algorithm for continuous optimization: Application to feed-forward neural network training," *Neural Comput. Appl.*, vol. 16, pp. 235–247, May 2007.

[61] S. Tsutsui, "Ant colony optimisation for continuous domains with aggregation pheromones metaphor," in *Proc. Int. Conf. Recent Adv. Soft Comput. (RASC)*, 2004, pp. 207–212.

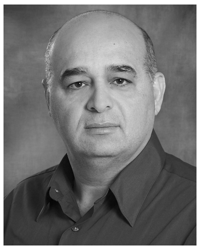
[62] K. M. Salama and A. M. Abdelbar, "Learning neural network structures with ant colony algorithms," *Swarm Intell.*, vol. 9, no. 4, pp. 229–265, 2015.

[63] A. M. Abdelbar and K. M. Salama, "A gradient-guided ACO algorithm for neural network learning," in *Proc. IEEE Swarm Intell. Symp. (SIS)*, Dec. 2015, pp. 1133–1140.

[64] A. M. Abdelbar, I. El-Nabarawy, D. C. Wunsch, II, and K. M. Salama, "Ant colony optimization applied to the training of a high order neural network with adaptable exponential weights," in *Applied Artificial Higher Order Neural Networks for Control and Recognition*, M. Zhang, Ed. Hershey, PA, USA: IGI Global, 2016, pp. 362–374.

[65] U. Kumar, S. Soman, and Jayadeva, "Benchmarking NLopt and state-of-the-art algorithms for continuous global optimization via IACO$_R$," *Swarm Evol. Comput.*, vol. 27, pp. 116–131, Apr. 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7942005

[66] Y. F. Lou and S. Y. Yuen, "Non-revisiting genetic algorithm with adaptive mutation using constant memory," *Memetic Comput.*, vol. 8, no. 3, pp. 189–210, 2016.



ASHRAF M. ABDELBAR received the B.Sc. degree in computer science from The American University in Cairo, Egypt, in 1991, and the M.Sc. and Ph.D. degrees in computer science from Clemson University, USA, in 1994 and 1996, respectively. From 1996 to 2013, he was a Faculty Member with The American University in Cairo. He is currently a Professor with the Department of Mathematics and Computer Science, Brandon University, and has served as Chair of the department. He has published more than 100 papers in various subareas of artificial intelligence, including Bayesian belief networks, neural networks, and swarm intelligence. He was a recipient of the INNS Young Investigator Award. He has served on the ACM Membership Activities Board and on the CIPS Computer Science Accreditation Council. He has served as a Program Evaluator for ABET. He has been coaching student teams for the ACM International Collegiate Programming Contest (ICPC), since 1998; his teams have advanced to the ICPC World Finals five times over the course of his career to date.

KHALID M. SALAMA received the B.Sc. degree from the Faculty of Computer Science and Information Systems, Ain Shams University, Egypt, in 2007, the M.Sc. degree from the Department of Computer Science and Engineering, The American University in Cairo, Egypt, in 2010, and the Ph.D. degree from the School of Computing, University of Kent, U.K., in 2014. Since 2015, he has been a Technical Consultant for building data, advanced analytics, and machine learning solutions. He is currently a Research Fellow with the Computational Intelligence Research Group, School of Computing, University of Kent. He has published more than 25 papers in various subareas of artificial intelligence, including Bayesian belief networks, neural networks, and swarm intelligence. His research interests include the applications of evolutionary and swarm algorithms in the field of data mining and machine learning.