The Automatic Design of Multi-Objective Ant Colony Optimization Algorithms

Manuel López-Ibáñez and Thomas Stützle

***Abstract*—Multi-objective optimization problems are problems with several, typically conflicting criteria for evaluating solutions. Without any a priori preference information, the Pareto optimality principle establishes a partial order among solutions, and the output of the algorithm becomes a set of nondominated solutions rather than a single one. Various ant colony optimization (ACO) algorithms have been proposed in recent years for solving such problems. These multi-objective ACO (MOACO) algorithms exhibit different design choices for dealing with the particularities of the multi-objective context.**

This paper proposes a formulation of algorithmic components that suffices to describe most MOACO algorithms proposed so far. This formulation also shows that existing MOACO algorithms often share equivalent design choices but they are described in different terms. Moreover, this formulation is synthesized into a flexible algorithmic framework, from which not only existing MOACO algorithms may be instantiated, but also combinations of components that were never studied in the literature. In this sense, this paper goes beyond proposing a new MOACO algorithm, but it rather introduces a family of MOACO algorithms.

The flexibility of the proposed MOACO framework facilitates the application of automatic algorithm configuration techniques. The experimental results presented in this paper show that the automatically configured MOACO framework outperforms the MOACO algorithms that inspired the framework itself. This paper is also among the first to apply automatic algorithm configuration techniques to multi-objective algorithms.

***Index Terms*—Multiobjective Optimization, Ant Colony Optimization, Travelling Salesman Problem, Automatic Algorithm Configuration**

I. Introduction

ANT colony optimization (ACO) [1] is a metaheuristic inspired by the pheromone trail laying and following behavior of some real ant species. ACO was originally designed for solving single-objective combinatorial optimization problems. Due to notable results on these problems, ACO algorithms were soon extended to tackle problems with more complex features [2] and, in particular, multiple objective functions [3, 4]. The majority of these multi-objective ACO (MOACO) algorithms focus on problems in terms of Pareto optimality, that is, they do not make a priori assumptions about the decision maker's preferences. Moreover, most of these MOACO algorithms were proposed for bi-objective optimization problems, and, hence, these will be the focus of this paper.

M. López-Ibáñez and T. Stützle are with the IRIDIA laboratory, CoDE, Université libre de Bruxelles, 1050 Brussels, Belgium.
email: manuel.lopez-ibanez@ulb.ac.be, stuetzle@ulb.ac.be
This is the final draft version accepted by IEEE Transactions on Evolutionary Computation on December 13, 2011.
DOI: 10.1109/TEVC.2011.2182651

Compared to the substantial amount of research on evolutionary computation and local search algorithms for tackling multi-objective optimization problems, there are relatively few works on MOACO algorithms. Most articles propose only one specific MOACO algorithm [5, 6]; rare are studies that compare a few MOACO design alternatives [7–9]. A first review of existing MOACO algorithms included about ten MOACO algorithms [4]. It identified, among the ones proposed in the literature, the best MOACO algorithm adapted to a particular problem in a traditional "horse-race" experimental analysis. The algorithms tested in that review differ substantially with respect to the underlying ACO algorithm, e.g., some of them are based on the classical Ant System [10], whereas others build upon the typically better performing $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [11] and Ant Colony System (ACS) [12]. Therefore, it is difficult to conclude anything about the specific design of each MOACO algorithm. A more recent review [3] categorizes existing MOACO algorithms into several classes without any empirical comparison.

This paper integrates and extends our recent work on MOACO algorithms [13–15], which has advanced in several directions. As a first step, we present an algorithmic framework that synthesizes many of the design choices proposed in the literature and that offers new possibilities never considered before. This framework allows us to instantiate existing MOACO algorithms by properly configuring the MOACO framework. More importantly, it allows us to combine ideas from different MOACO algorithms to produce new ones. A first insight from the use of this framework is that existing MOACO algorithms share more algorithmic components than the respective publications may suggest. A second contribution is that we can devise more powerful combinations of algorithmic components and, at the same time, we can assess the influence of each design choice on performance. In addition, the flexibility of this framework facilitates the application of automatic techniques for configuring multi-objective algorithms. Most previous works that aim to produce high performing algorithms by combining a flexible software framework and an automatic configuration tool have dealt so far only with single-objective optimization problems [16]. We proposed integrating the hypervolume indicator into the iterated F-race tool (I/F-Race) [17, 18] in order to automatically configure our MOACO framework for tackling the bi-objective traveling salesman problem (bTSP) [15]. The integration of unary quality indicators such as hypervolume and epsilon measure into I/F-Race allows us to automatically configure multi-objective algorithms with many continuous, categorical and conditional parameters for an optimization problem, given a

set of representative instances of this problem. Independently, Wessing et al. [19] have presented results for configuring a multi-objective evolutionary algorithm using the hypervolume indicator. However, they considered only a single parameter, the variation operator, and configured the algorithm on a single instance; thus, their approach is prone to overtuning [20]. In this paper, we show that, by automatically configuring a flexible framework that synthesizes design concepts from different MOACO algorithms, we are able to find new designs of MOACO algorithms for the bTSP that outperform the MOACO algorithms found in the literature. Our results demonstrate the effectiveness of the usage of automatic algorithm configuration techniques for the design of stochastic local search (SLS) algorithms [21] for multi-objective optimization problems in terms of Pareto-optimality.

The paper is structured as follows. Section II introduces basic concepts that will be used throughout the paper. We present a general MOACO framework for bi-objective combinatorial optimization problems in Section III. The components of this framework correspond to different alternative choices in the design of a MOACO algorithm. Next, we describe in Section IV how the existing framework is able to replicate the design of most MOACO algorithms proposed in the literature. In Section V, we propose a method to automatically configure the MOACO framework for a particular bi-objective problem. The problem chosen is the bTSP, and we compare the results to the best MOACO configurations proposed in the literature. In particular, we perform a more comprehensive automatic configuration and comparison than the preliminary one described in our previous work [15]. Finally, in addition to the proposed configuration of the framework for the bTSP, we extract some general conclusions from this work and describe steps to exploit our proposals in Section VII.

II. Preliminaries

A. Bi-objective Combinatorial Optimization

We propose here a framework based on ACO for tackling bi-objective combinatorial optimization problems (bCOPs) in terms of Pareto optimality. In such bCOPs, candidate solutions are evaluated according to an *objective function vector* $\vec{f} = (f_1, f_2)$. Without a priori assumptions on the preferences of the decision maker, the goal is to determine a set of feasible solutions that "minimizes" the objective function vector $\vec{f}$. Let $\vec{u}$ and $\vec{v}$ be vectors in $\mathbb{R}^2$. We say that $\vec{u}$ *dominates* $\vec{v}$ ($\vec{u} \prec \vec{v}$) iff $\vec{u} \neq \vec{v}$ and $u_i \leq v_i,\ i = 1, 2$. Furthermore, $\vec{u}$ and $\vec{v}$ are *nondominated* iff $\vec{u} \not\prec \vec{v}$ and $\vec{v} \not\prec \vec{u}$. To simplify the notation, we also say that a feasible solution s dominates another solution s' iff $\vec{f}(s) \prec \vec{f}(s')$. A solution s is a *Pareto optimum* iff no other feasible solution s' exists such that $\vec{f}(s') \prec \vec{f}(s)$. The goal in bCOPs then typically is to determine the set of all Pareto-optimal solutions. However, this task is usually computationally intractable, and, hence, it is preferable to approximate the Pareto set as well as possible in a given amount of time. Such an approximation is always a set of solutions that are mutually nondominated.

B. The Bi-objective Traveling Salesman Problem (bTSP)

The single-objective TSP is one of the most important combinatorial optimization problems and it is used as a case study in the development of many ACO algorithms. In the TSP, we are given a complete graph $G = (V, E)$ with $n = |V|$ nodes $\{v_1, \ldots, v_n\}$, a set of edges E that fully connects the nodes, and a cost $c(v_i, v_j)$ associated with each edge. The goal is to find a Hamiltonian tour $p = (p_1, \ldots, p_n)$ that minimizes the total cost:

$$\text{minimize} \quad f(p) = c\left(v_{p_n}, v_{p_1}\right) + \sum_{i=1}^{n-1} c\left(v_{p_i}, v_{p_{i+1}}\right). \qquad (1)$$

The bi-objective TSP (bTSP) is the direct extension of the above formulation. For each edge, a vector of costs is given with two components $c_1(v_i, v_j)$ and $c_2(v_i, v_j)$, $i \neq j$, which correspond to the cost of the first and the second objective, respectively. We consider here the symmetric bTSP, where it additionally holds that $c_q(v_i, v_j) = c_q(v_j, v_i)$, $i \neq j$, $q = 1, 2$. We assume not to have a priori knowledge of the preferences of the decision maker. Therefore, the goal is to find the set of Hamiltonian tours that minimizes, in the sense of Pareto optimality, the vector of objective functions $\vec{f} = (f_1, f_2)$. The bTSP is often used as a benchmark problem for testing multi-objective combinatorial optimization algorithms [4, 13, 22–24].

C. Ant Colony Optimization (ACO)

The ACO metaheuristic [1] describes a number of algorithms inspired by the swarm behavior of real ants species. In ACO algorithms, a number of artificial ants construct candidate solutions to a problem. The construction decisions of the ants are biased by a common numerical information called (artificial) pheromone. This pheromone is updated according to the quality of the solutions constructed by the ants in order to bias the construction of new solutions in subsequent iterations. In particular, an ant constructs a candidate solution to a problem by iteratively adding solution components to its partial solution in a stochastic fashion. In the case of the TSP, the probability that an ant k chooses to visit node j after node i is given by:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta} \quad \text{if } j \in \mathcal{N}_i^k, \qquad (2)$$

where τ_{ij} is the pheromone associated with adding the edge (i, j) to the current partial tour, η_{ij} is a static greedy measure of the "goodness" of edge (i, j) called *heuristic information*, and $\mathcal{N}_i^k$ denotes the set of feasible choices available for ant k located in node i given its current partial solution.

After a number of ants have constructed a solution each, one or more of these solutions are used to update the pheromone information in such a way as to bias future choices towards high quality solutions. It is common in ACO algorithms to perform other actions, such as local search, to further improve solutions before updating the pheromone information. The pseudo-code of the ACO metaheuristic is shown in Algorithm 1.

Procedure ACO_Metaheuristic
  repeat
    ConstructSolutions()
    OptionalActions() // e.g. local search
    UpdatePheromones()
  until stopping criteria met
  Output: best solution found

Figure 1. The ACO metaheuristic for single-objective combinatorial problems.

III. A Configurable MOACO Framework

There are a number of design questions when extending ACO algorithms to bCOPs. First, in single-objective ACO, the pheromone information is related to the objective function, that is, the components of higher quality solutions receive more pheromone. In a multi-objective context, the objective function is multi-dimensional and not scalar, and there is only a partial order among solutions. Moreover, in some problems, it could make sense to have different solution components for each objective, and, hence, associate different pheromone matrices to them [7]. For example, in applications of ACO to multi-objective scheduling problems, one pheromone matrix may represent jobs×jobs relationships, and the other position×jobs relationships. In this case, both matrices could be updated with the same pheromone amount because their meaning is different. In other problems, such as the bTSP, both pheromone matrices represent the same solution components (an edge), and, hence, they would need to be updated either using different solutions or using different pheromone amounts [5, 9]. Alternatively, it could also make sense to use a single pheromone matrix [6]. If multiple pheromone or heuristic matrices are used, they are typically aggregated during the solution construction by means of weights [5, 7]. However, there are strong differences among MOACO algorithms in the way this aggregation is done and how many weights are used. A further design question is which ants are selected for depositing pheromone. Existing MOACO algorithms select some (or all) nondominated solutions [7], or they select the best solutions with respect to the objective associated with the pheromone matrix that is updated [5, 9]. Finally, several MOACO algorithms make use of multiple colonies of ants [7, 9].

We take a component-wise view of the design of MOACO algorithms, which allows us to abstract from the particular design choices taken in earlier works. We have identified several algorithmic components that correspond to the design alternatives described above. We divide these components in three main groups: (*i*) those related to the definition of pheromones and the construction of solutions; (*ii*) those related to the update of the pheromones; and (*iii*) those related to the use of multiple colonies. We examine these components in more detail in the following section. We will discuss the connection between these components and existing MOACO algorithms in Section IV.

Our proposed MOACO framework does not specify many details, such as how pheromone values are initialized, updated or evaporated. It neither specifies whether the pheromone information is updated using the best solutions found since the start of the algorithm (best-so-far) or found in the current iteration (iteration-best). These details are relegated to the underlying ACO algorithm, which we assume is an efficient implementation for the problem at hand of any modern ACO algorithm such as $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [11] or Ant Colony System (ACS) [12]. This approach allows us to focus, in our framework, on the multi-objective components of MOACO algorithms.

A. MOACO algorithm components for solution construction

One of the most crucial design decisions when applying an ACO algorithm to a problem is the definition of pheromone (and heuristic) information. In the multi-objective context, this problem is exacerbated by the fact that there are several ways to evaluate a single solution, and by the fact that multiple solutions may be (Pareto-)optimal. This design question is answered by the following algorithmic components.

Single/Multiple Pheromone/Heuristic Information. We may have one (*single*) or several (*multiple*) matrices for either the pheromone information τ or, if applicable, the heuristic information η. In the case of multiple heuristic matrices, each matrix is associated with a different objective, such that η^1 and η^2 correspond to the heuristic information of each objective, respectively. For some problems, it may not be possible to define the heuristic information for each objective independently, so we will be forced to use a single heuristic matrix. Similarly, in the case of multiple pheromone matrices, τ^1 and τ^2 would be associated with each objective; the procedure that selects the solutions used for updating each matrix (see next subsection) should enforce this distinction somehow.

During solution construction, whenever we have multiple matrices, we will need to aggregate them into a single matrix. The aggregation method is defined by the component Aggregation, which is described below. It is conceivable to use a different aggregation method for pheromone and heuristic information; however, we do not explore this possibility here.

Aggregation. The values from multiple pheromone (or heuristic) matrices need to be aggregated into a single pheromone (or heuristic) value. We have identified three alternatives in the literature:

- **Weighted sum**, e.g.,
$$\tau_{ij} = (1-\lambda)\tau^1_{ij} + \lambda\tau^2_{ij} \quad \text{and} \quad \eta_{ij} = (1-\lambda)\eta^1_{ij} + \lambda\eta^2_{ij}.$$
- **Weighted product**, e.g.,
$$\tau_{ij} = (\tau^1_{ij})^{(1-\lambda)} \cdot (\tau^2_{ij})^{\lambda} \quad \text{and} \quad \eta_{ij} = (\eta^1_{ij})^{(1-\lambda)} \cdot (\eta^2_{ij})^{\lambda}.$$
- **Random**. At each construction step, given a uniform random number $\mathcal{U}(0,1)$, an ant selects the first of the two matrices if $\mathcal{U}(0,1) < 1-\lambda$; otherwise it selects the other matrix.

In the three aggregation methods described above, there is a weight λ that biases the aggregation towards one objective or the other. The set of weights Λ is defined by the components N^{weights} and NextWeight.

N^{weights} and NextWeight. The set of weights is defined within the interval $[0, 1]$ as

$$\Lambda = \left\{\lambda_i = 1 - (i-1)/(N^{\text{weights}} - 1), i = 1, \ldots, N^{\text{weights}}\right\},$$

where $N^{\text{weights}} = |\Lambda|$ is a parameter of the framework. Component NextWeight determines which particular weight is used by an ant at a certain iteration. The options tested for NextWeight are either that all ants use the same weight at a certain iteration (*one-weight-per-iteration*), or that all weights are used at each iteration (*all-weights-per-iteration*). In the case of *one-weight-per-iteration*, the weight used in successive iterations follows an ordered sequence of the elements of Λ, and the order is reversed when the last weight in the sequence is reached. In the case of *all-weights-per-iteration*, when the number of ants N^{a} is larger than N^{weights}, several ants will use the same weight. An obvious speed-up is to compute the aggregation of the pheromone matrices only once per weight per iteration. This describes the behavior for a single colony; the multi-colony case will be discussed in Section III-C.

B. MOACO algorithm components for pheromone update

Given a set A^{upd} of candidate solutions for updating the pheromone information, the PheromoneUpdate component decides which solutions update the pheromone information and how. The N^{upd} parameter determines how many solutions are used to update each pheromone matrix. We consider the following alternatives for the PheromoneUpdate component:

- **Nondominated solutions.** The solutions used for updating the pheromone information are the nondominated solutions in A^{upd}. When there are more nondominated solutions than N^{upd}, we apply the truncation mechanism of SPEA2 [25] to select only N^{upd} solutions. In principle, it is possible to combine this *nondominated* pheromone update method and multiple pheromone matrices by using the same solutions to update both matrices [7] as long as the update is different in each pheromone matrix.
- **Best-of-objective.** This mechanism first selects from A^{upd} the N^{upd} best solutions with respect to each objective. In the case of multiple pheromone matrices, each pheromone matrix is updated using the N^{upd} solutions associated with the corresponding objective. Otherwise, the $2 \cdot N^{\text{upd}}$ solutions update the single pheromone matrix.
- **Best-of-objective-per-weight.** For each weight λ and each objective, there is a list of the N^{upd} best solutions for that objective generated using λ. In the particular case of $\lambda = 0$, we keep only the list for the first objective, and we do the same for $\lambda = 1$ and the second objective. When using multiple pheromone matrices, each matrix is updated using only solutions from lists associated with the same objective. In the single pheromone matrix case, both lists are used for the update. Therefore, in the particular case of two weights, this method is equivalent to *best-of-objective*. The *best-of-objective-per-weight* method is used by the existing mACO-1 and mACO-2 algorithms [9], and we include it for completeness. However, it is not clear how this approach should be extended to multiple colonies, since solutions may be exchanged among colonies with different weights.

The above methods do not describe whether A^{upd} is composed of the best-so-far or iteration-best solutions, since this is determined by the particular underlying ACO algorithm.

C. MOACO algorithm components for multiple colonies

Many MOACO algorithms propose the use of multiple ant colonies, using various definitions of what a colony is. One popular approach is to define two colonies, each of them having its own pheromone information. These colonies may exchange solutions that are used to update the pheromone information. Some ants, which sometimes are said to belong to "extra" colonies, aggregate the pheromone information of the two other colonies [9, 26]. This approach is actually equivalent to using one colony with multiple pheromone matrices, which are aggregated by means of different weights using the all-weights-per-iteration option, and it does not seem appropriate to say that ants belong to different colonies simply because they use different weights. Differently, the multi-colony architecture of Iredi et al. [7] defines a colony as a group of ants associated with a particular pheromone information, such that ants from each colony construct solutions only according to the pheromone information of their colony. In the case of multiple pheromone information, each colony has two pheromone matrices, one for each objective. When a set of weights is used, each colony has its own set of weights Λ_c. This multi-colony architecture allows us to generalize all previous components and, hence, we adopt it for our framework. Moreover, colonies cooperate by using a common archive of nondominated solutions for detecting dominated ones. Further cooperation is enforced by exchanging solutions for updating the pheromone information. This setting is controlled by the MultiColonyUpdate component described below. The number of colonies is given by component N^{col}, and the components described below only have an effect when N^{col} is larger than one.

MultiColonyWeights. In the case of multiple colonies, we create a set of weights Λ_c of size N^{weights} for each colony c. These sets are constructed by following the two alternatives proposed by Iredi et al. [7]: *disjoint* and *overlapping* intervals. In both cases, we first generate the necessary number of equally distributed weights in the interval $[0, 1]$. Then, for *disjoint* intervals, this set is partitioned into equal disjoint sub-intervals per colony, that is $\lambda_{c,i} = ((c-1) \cdot N^{\text{weights}} + (i-1))/(N^{\text{weights}} \cdot N^{\text{col}})$, $i = 1, \ldots, N^{\text{weights}}$, $c = 1, \ldots, N^{\text{col}}$. In the case of *overlapping* intervals, the sub-intervals overlap by 50% and hence, Λ_c and Λ_{c+1} share 50% of their weights.

MultiColonyUpdate. In the case of multiple colonies, the solutions generated by all colonies in the current iteration are stored in a common archive A^{iter}, so that all colonies contribute to detect and remove dominated solutions. The set A^{upd} is then built from the remaining nondominated solutions in A^{iter} or from the archive of all nondominated solutions ever found, depending on the underlying ACO algorithm. After this step, the solutions in A^{upd} are assigned back to each colony

for updating the pheromone information. The basic method, called *update by origin*, assigns each solution from A^{upd} to its original colony. To enforce more cooperation, colonies may exchange solutions. One method of exchange, called *update by region*, divides A^{upd} in equal parts among the colonies in such a way that each colony roughly corresponds to one region of the objective space. Both settings, *update by origin* and *update by region*, were originally proposed by Iredi et al. [7].

D. The MOACO framework

Table I summarizes the algorithmic components defined above and their domains. Some settings are only significant for certain values of other settings. For example, an aggregation method is only necessary if there are multiple pheromone or heuristic matrices. We propose Algorithm 2 as a way to integrate these components into a flexible MOACO framework for bi-objective optimization. It follows the basic outline of the ACO metaheuristic. First, the algorithm initializes the pheromone information (function InitializePheromoneInformation, line 2) and the set of weights of each colony (function MultiColonyWeights, line 3), whereas heuristic information is initialized only once, since it is shared by the different colonies. The archive of all nondominated solutions ever found (*best-so-far* archive, A^{bf}) and the iteration counter are initialized in lines 6 and 7, respectively. At each iteration, each colony constructs N^{a} solutions according to its own pheromone information, which may be possibly aggregated by a weight λ from its own set of weights (lines 13–15). Depending on the setting of NextWeight, λ may be the same weight or a different one for each ant in the colony. In single-objective ACO algorithms, solutions are often improved by means of local search. Therefore, we have included an optional WeightedLocalSearch procedure (line 16) that improves a solution by applying local search to a weighted sum aggregation of the objective functions, using the same weight as it was used for constructing the solution. The new solution is added to the archive of the current iteration shared by all colonies A^{iter} (line 17). Once all ants from all colonies have finished constructing solutions, the best-so-far archive A^{bf} is updated with the nondominated solutions found in the current iteration A^{iter} (in function RemoveDominated, line 20). The update of the pheromones consists of two phases. First, MultiColonyUpdate distributes the archive of solutions for update (A^{upd}) among the colonies. Second, procedure PheromoneUpdate (line 23) decides how the solutions assigned to each colony update the pheromone information of the colony. The MOACO algorithm continues until a certain number of iterations or a time limit is reached. Figure 3 graphically illustrates the relationships between various components of the MOACO framework.

1: **for** each colony $c \in \{1, \ldots, N^{\text{col}}\}$ **do**
2: InitializePheromoneInformation()
3: $\Lambda_c :=$ MultiColonyWeights()
4: **end for**
5: InitializeHeuristicInformation()
6: $A^{\text{bf}} := \emptyset$
7: $iter := 0$
8: **while** not stopping criteria met **do**
9: $A^{iter} := \emptyset$
10: **for** each colony $c \in \{1, \ldots, N^{\text{col}}\}$ **do**
11: **for** each ant $k \in \{1, \ldots, N^{\text{a}}\}$ **do**
12: $\lambda :=$ NextWeight(Λ_c, k, $iter$)
13: $\tau := \begin{cases} \text{Aggregation}(\lambda, \{\tau_c^1, \tau_c^2\}) & \text{if multiple } [\tau] \\ \tau_c & \text{if single } [\tau] \end{cases}$
14: $\eta := \begin{cases} \text{Aggregation}(\lambda, \{\eta^1, \eta^2\}) & \text{if multiple } [\eta] \\ \eta & \text{if single } [\eta] \end{cases}$
15: $s :=$ ConstructSolution(τ, η)
16: $s :=$ WeightedLocalSearch(s, λ) // Optional
17: $A^{iter} :=$ RemoveDominated($A^{iter} \cup \{s\}$)
18: **end for**
19: **end for**
20: $A^{\text{bf}} :=$ RemoveDominated($A^{\text{bf}} \cup A^{iter}$)
21: **for** each colony $c \in \{1, \ldots, N^{\text{col}}\}$ **do**
22: $A_c^{\text{upd}} :=$ MultiColonyUpdate(A^{upd})
23: PheromoneUpdate($A_c^{\text{upd}}, N^{\text{upd}}$)
24: **end for**
25: $iter := iter + 1$
26: **end while**
27: **Output:** A^{bf}

Figure 2. MOACO framework.

IV. The Design of MOACO Algorithms

Our previous work [14] has examined the design of several MOACO algorithms from the literature. This study has informed the design of the MOACO framework described in the previous section. In this section, we explain how certain configurations of the framework replicate the design of existing MOACO algorithms. The aim here is not to faithfully reproduce the original algorithms, but to identify which particular configuration of algorithmic components in the framework corresponds to the algorithmic design choices made in each case. In particular, we do not pay attention here to the underlying ACO algorithm, pheromone initialization or update amounts, heuristic information, or problem-dependent speed-ups. We are aware that the present review does not include all MOACO algorithms in the literature. We do not attempt to adapt algorithms not specifically designed for Pareto optimization, such as MACS-VRPTW [27], or algorithms that diverge from the basic structure of the ACO metaheuristic, such as Population-based ACO [28, 29].

Table II summarizes the configuration of the MOACO framework that instantiates the MOACO algorithms reviewed in this section.

A. MOAQ

Although Multiple Objective Ant-Q (MOAQ) [30] was originally designed for problems with a given preference order of the objectives (lexicographical optimization), García-Martínez et al. [4] generalized MOAQ to bi-objective problems in terms of Pareto optimality. Their proposal uses a single pheromone matrix and multiple heuristic matrices, one

Table I
ALGORITHMIC COMPONENTS OF THE PROPOSED MOACO FRAMEWORK.

Component	Domain	Description
$[\tau]$	{ single, multiple }	Definition of pheromone matrices
$[\eta]$	{ single, multiple }	Definition of heuristic matrices
Aggregation	{ weighted sum, weighted product, random	How weights are used to aggregate different matrices
N^{weights}	$\mathbb{N}^+$	Number of weights (per colony)
NextWeight	{ one weight per iteration (1wpi), all weights per iteration (awpi)	How weights are used at each iteration
PheromoneUpdate	{ nondominated solutions (ND), best-of-objective (BO), best-of-objective-per-weight (BOW)	Which solutions are selected for updating the pheromone matrices
N^{upd}	$\mathbb{N}^+$	Number of solutions that update each $[\tau]$ matrix
N^{col}	$\mathbb{N}^+$	Number of colonies
MultiColonyWeights	{ disjoint, overlapping }	How weights are partitioned among colonies
MultiColonyUpdate	{ origin, region }	How solutions are assigned to colonies for update

Table II
TAXONOMY OF MOACO ALGORITHMS AS INSTANTIATIONS OF THE PROPOSED FRAMEWORK (N^{A} IS THE NUMBER OF ANTS).

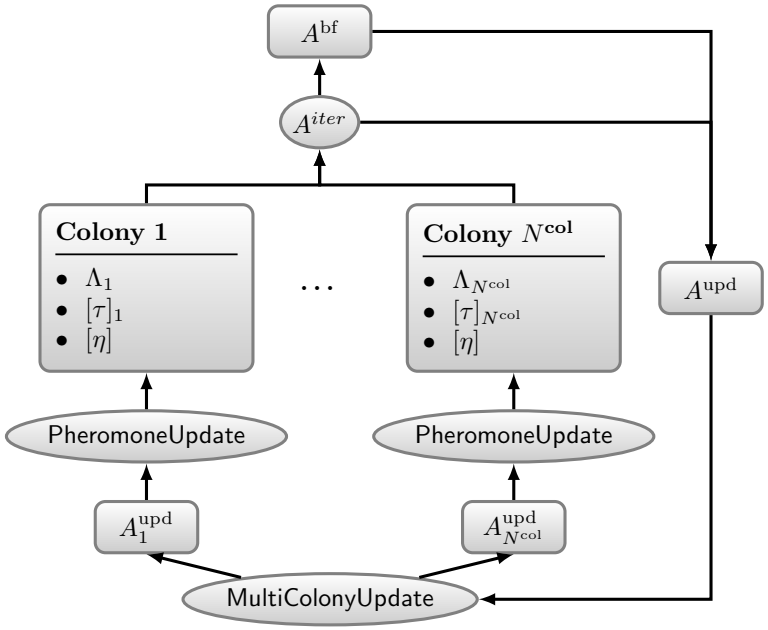
Algorithm	N^{col}	$[\tau]$	$[\eta]$	Aggregation	N^{weights}	PheromoneUpdate	N^{upd}
MOAQ [4, 30]	1	1	2	–	2 ($\Lambda = \{0, 1\}$)	nondominated solutions	∞
BicriterionAnt [7]	any	2	2	weighted product	N^{a}	nondominated solutions	∞
MACS [6]	1	1	2	weighted product	N^{a}	nondominated solutions	∞
COMPETants [26]	1	2	2	weighted sum	3 ($\Lambda = \{0, 0.5, 1\}$)	best-of-objective	any
P-ACO [5]	1	2	1, 2	weighted sum	N^{a}	best-of-objective	2
mACO-1 [9]	1	2	2	{ random (τ) weighted sum (η)	3 ($\Lambda = \{0, 0.5, 1\}$)	best-of-objective-per-weight	1
mACO-2 [9]	1	2	2	weighted sum	3 ($\Lambda = \{0, 0.5, 1\}$)	best-of-objective-per-weight	1
mACO-3 [9]	1	1	1	–	–	nondominated solutions	∞
mACO-4 [9]	1	2	1	random (τ)	1 ($\Lambda = \{0.5\}$)	best-of-objective	1

for each objective. For the pheromone update, they use all nondominated solutions. García-Martínez et al. [4] divide the ants into two groups, and each group only uses the heuristic information corresponding to one objective. The equivalent in our framework is to use the weights $\Lambda = \{0, 1\}$ for aggregating the two heuristic matrices, such that half of the ants use each weight. Such a set of weights effectively means that no actual aggregation takes place, but instead half of the ants use one heuristic matrix and the other half use the other one.

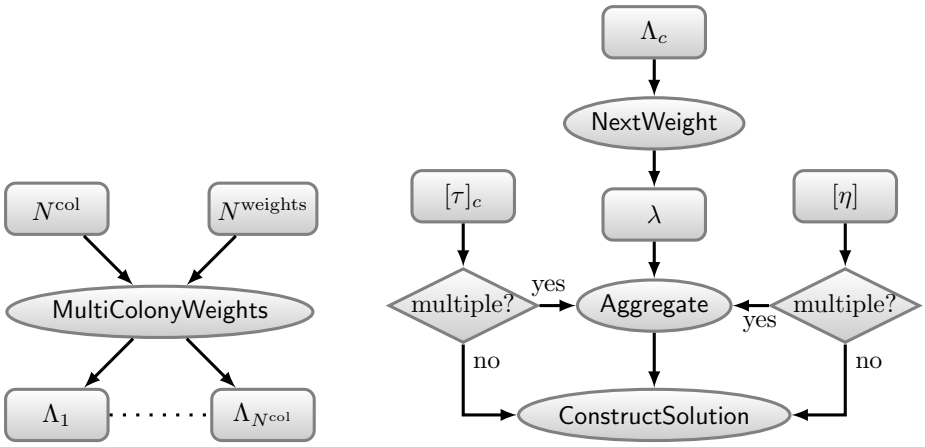
B. BicriterionAnt

Iredi et al. [7] proposed a MOACO algorithm, henceforth called BicriterionAnt, with multiple pheromone and heuristic matrices, aggregated by weighted product. Each ant k uses a different weight λ_k for aggregating the pheromone matrices; thus, there are as many weights as ants ($N^{\text{weights}} = N^{\text{a}}$). The authors of BicriterionAnt suggest to update the pheromone matrices using the nondominated solutions found in the current iteration. Moreover, they update both pheromone matrices with the same amount. Such an update does not result in identical pheromone matrices because, in the problem tackled by them, the pheromone matrices represent different solution components as explained on page 3 of this article. When applying BicriterionAnt to the bTSP, García-Martínez et al. [4] used the objective function value of each objective to update the corresponding matrix $\Delta\tau^k = 1/f^k(s_a)$. We do the same in our framework for the combination of multiple pheromone matrices and nondominated pheromone update.

In addition to the above algorithm, Iredi et al. [7] proposed the use of *multiple colonies*, and define a colony as a group of ants that construct solutions according to their own pheromone information. In this sense, a multi-colony BicriterionAnt algorithm is similar to a multi-start approach with some particularities. First, different colonies specialize in different regions of the Pareto frontier by using different sets of weights to aggregate pheromone information. Second, colonies cooperate to detect dominated solutions by keeping solutions in a common archive. Third, colonies may also cooperate by exchanging solutions. As explained in Section III-C, this matches the multi-colony approach adopted in our MOACO framework.



(a) Main algorithm



(b) Division of weights among colonies

(c) Solution construction within colony c

Figure 3. Diagram showing the relationships between various components of the MOACO framework. Rounded rectangles represent data objects, and ellipses represent procedures. Figure (a) gives a high-level overview of the main algorithm. Figure (b) shows the algorithm components influencing the distribution of the weights among colonies. Figure (c) gives a detailed view of the components involved in the solution construction within a single colony.

C. Multiple Ant Colony System

Multiple Ant Colony System (MACS) [6] uses one heuristic matrix for each objective and a single pheromone matrix. The heuristic matrices are aggregated by weighted product, and each ant uses a different weight. In addition, the pheromone information is updated with nondominated solutions. MACS can be seen as a variant of MOAQ, as defined by García-Martínez et al. [4], that uses more than two weights to aggregate the heuristic information. On the other hand, the only difference between MACS and single-colony BicriterionAnt is the number of pheromone matrices. As a result, it is straightforward to define a multi-colony MACS, as we do in our MOACO framework.

D. COMPETants

COMPETants [26] is presented as a multi-colony approach, with one colony for each objective. Each colony has one pheromone and heuristic matrix. Each colony constructs solutions independently, except for a number of ants (called "spies"), which aggregate the two pheromone matrices by weighted sum (with $\lambda = 0.5$) using either the first or the second heuristic matrix, thus creating two solutions. Finally, a number of ants from each colony are used to update the pheromone matrix of each colony.

We can formulate COMPETants in the MOACO framework by using two pheromone and heuristic matrices, which are aggregated by weighted sum and three weights $\Lambda =$

$\{0, 0.5, 1\}$. Thus, COMPETants is a single-colony approach in our framework, and it is straightforward to define variants of this formulation with an arbitrary number of colonies. For the sake of simplicity, in our MOACO framework, the total number of ants is equally divided by the number of weights, and the number of ants per weight does not change during the run, as in the original proposal; nevertheless, this feature could be added to the framework as an additional component. Moreover, our previous work [14] showed that, at least in the bi-objective TSP, never aggregating the heuristic information, as in the original COMPETants, leads to poor quality in the middle of the Pareto front. Hence, we do not handle specially the heuristic information for ants using $\lambda = 0.5$ but the heuristic matrices are aggregated in the same way as pheromone matrices. Finally, each pheromone matrix is updated with the N^{upd} best solutions for the corresponding objective, which is the *best-of-objective* setting for PheromoneUpdate in the MOACO framework.

E. Pareto Ant Colony Optimization

Pareto Ant Colony Optimization (P-ACO) [5] uses multiple pheromone matrices, one for each objective, aggregated by means of a weighted sum. Like in BicriterionAnt and MACS, a different weight is associated with each ant. Moreover, pheromone matrices are updated with the best and second-best solution for each objective, which is basically the same update method used by COMPETants, and in our framework it corresponds to the *best-of-objective* setting for PheromoneUpdate and $N^{\text{upd}} = 2$. In the problem being solved in the original paper [5], there was no clear definition of heuristic information for each objective, and, hence, the authors used a single heuristic matrix. However, in later publications [31], P-ACO uses multiple heuristic matrices, one for each objective, which are aggregated in the same way as the pheromone matrices. In our earlier work [14], we show that there are important differences between using one or two heuristic matrices in P-ACO for the bi-objective TSP. In particular, we observe that a single heuristic matrix leads to a very narrow Pareto front and, hence, our MOACO framework uses multiple heuristic matrices when instantiating P-ACO. Nonetheless, our MOACO framework can replicate both variants.

F. mACO Variant 1 (mACO-1)

Alaya et al. [9] proposed four alternatives for the design of a MOACO algorithm. The first variant, mACO-1, is described as using multiple colonies: one colony per objective, and an extra colony that builds solutions by aggregating the pheromone matrices of the other two colonies following a random aggregation. Each colony uses the heuristic information of its corresponding objective, whereas the extra colony aggregates the heuristic matrices. In our framework, this proposal is formulated as a single colony with multiple pheromone and heuristic matrices aggregated using three weights $\Lambda = \{0, 0.5, 1\}$. In the original mACO-1, the pheromone information of each colony is updated with the best solution generated by the colony for the corresponding objective of the same colony. Moreover, the algorithm keeps a best solution generated by the extra colony for each objective, and uses them to update the corresponding pheromone matrix of the other two colonies. Since each colony in the original corresponds to a different weight λ in our formulation, this update method corresponds in our framework to *best-of-objective-per-weight* with $N^{\text{upd}} = 1$.

G. mACO Variant 2 (mACO-2)

The second variant (mACO-2) by Alaya et al. [9] only differs from mACO-1 in the pheromone aggregation. In mACO-2, pheromone matrices are aggregated by summing the pheromone matrices of each objective. When the underlying ACO algorithm is scale invariant [32], like AS, MMAS and ACS, this is equivalent to a weight $\lambda = 0.5$. Our previous work on the bi-objective TSP suggests that there is not a significant difference in quality between mACO-1 and mACO-2 [14].

H. mACO Variant 3 (mACO-3)

The third variant proposed by Alaya et al. [9] uses a single pheromone matrix, which is updated by using all nondominated solutions (either from the iteration-best archive or the best-so-far archive). Alaya et al. [9] emphasize that every pheromone value associated with a solution component is updated at most once, despite how many solutions contain it. This is in contrast with other algorithms such as MOAQ, MACS, and BicriterionAnt, that use such a *nondominated* pheromone update. However, we did not find any advantage in this special requirement, therefore our framework does not include it for the sake of simplicity. The heuristic information is also a single matrix. In problems where there is heuristic information available for each objective, these are aggregated prior to the run into a single heuristic matrix.

I. mACO Variant 4 (mACO-4)

In the last variant proposed by Alaya et al. [9], there is one pheromone matrix per objective, and these are always aggregated in the same random manner as for mACO-1, which in practice corresponds to a random aggregation with a single weight $\lambda = 0.5$. However, there is a single heuristic matrix, as in mACO-3. Finally, each pheromone matrix is updated with the best solution for each objective, which is the definition of the *best-of-objective* pheromone update used in our framework.

J. New Design Alternatives

From the review in this section, we can say that, despite the number of different MOACO algorithms proposed in the literature, there is a much larger number of unexplored combinations of their algorithmic components. In particular, multi-colony variants in the sense of Iredi et al. [7] can be defined for all the algorithms reviewed in this section. The random aggregation introduced by Alaya et al. [9] has only been tested with $\lambda = 0.5$ so far. Moreover, all the algorithms reviewed above use all weights available in each iteration (*all-weights-per-iteration*). In our earlier work for the bi-objective QAP [8], we proposed that all ants use the same weight in one iteration, and the next weight in the sequence in the next

iteration (*one-weight-per-iteration*). Therefore, we can easily construct new variants of most algorithms in Table II. Such novel variants can be considered new MOACO algorithms. However, we expect that many of them would not lead to any significant breakthrough. On the other hand, the particular problem being solved may influence the choice of the best design. In any case, it would be a major effort to test them one by one in order to find the best design. Instead, we propose to automatically find the best design for a particular problem by automatic (offline) configuration of the MOACO framework.

There are several automatic methods for offline configuration of single-objective optimization algorithms. In an earlier work, we have extended Iterated F-Race (I/F-Race) [33] to the multi-objective case by using unary quality measures [15]. In the next section, we apply this approach to the framework described in this paper and carry out a detailed analysis of the results.

V. Automatic Configuration of the MOACO framework

Instead of the classical "horse-race" between fully instantiated MOACO algorithms that would entail a great deal of human effort, we exploit automated algorithm configuration tools to obtain very high-performing MOACO variants. The goal of automated (offline) configuration is to find the best parameter settings of an algorithm to solve unseen instances of a problem, given a set of training instances *representative* of the same problem. We are inspired by the work of KhudaBukhsh et al. [16], who combined ideas from many different algorithms for SAT into a highly configurable SAT solver, and then used automatic configuration tools to find the best configuration for specific classes of SAT instances. Instead of configuring a problem-specific solver for particular instance classes, we aim to configure a problem-independent metaheuristic for a specific problem. In other words, our goal is to find a good instantiation of a metaheuristic from a large space of potential designs. Moreover, we extend this idea for the first time to the multi-objective context. We show in this section that the automatic configuration of our flexible MOACO framework allows us to find better MOACO algorithms for the bTSP than those available in the literature. We also study different configuration strategies, and examine the results of several independent automatic configuration runs.

The automatic configuration method used here is I/F-Race [33], which is a state-of-the-art automatic configuration method able to deal with continuous, categorical and conditional parameters. We use the implementation of I/F-Race provided by the `irace` package [18]. I/F-Race alternates between generating new candidate configurations and performing *races* to discard the worst-performing ones. Within a race, candidate configurations are run on one instance at a time. I/F-Race uses the Friedman test followed by a post-test analysis to discard configurations whenever there is sufficient statistical evidence that they perform worse than the rest. When only a small number of configurations remain in the race, the race stops. A new race starts with the best configurations previously found and with new candidate configurations generated from the best configurations using a simple probabilistic model. The automatic configuration process stops after reaching a given maximum budget (number of runs or time limit).

The current version of I/F-Race was designed for single-objective optimization problems, and, hence, it requires an evaluation criterion that assigns a single value to each run of a configuration. We apply I/F-Race to the multi-objective context by means of unary quality measures, which assign a single quality value to a nondominated set. In particular, we test two unary measures as the evaluation criterion of I/F-Race, namely, the hypervolume[1] and the (additive) epsilon measure [35]. The hypervolume is the volume of the objective space weakly dominated by a nondominated set and bounded by a reference point that is strictly dominated by all Pareto-optimal objective vectors. The larger the hypervolume, the better is the corresponding nondominated set. The additive epsilon measure provides the minimum value that must be subtracted from all objectives of a nondominated set so that it weakly dominates a reference set. This reference set is usually the nondominated set of all known solutions. A smaller epsilon measure value is preferable.

As training instances, we generated 36 bTSP random uniform Euclidean instances for each of $n = \{100, 200, 300\}$ nodes (108 instances in total). We use a different set of 12 (test) instances when comparing algorithms. These test instances are taken from Luis Paquete's webpage at http://eden.dei.uc.pt/~paquete/tsp.[2] Each experiment, that is, each run of the MOACO framework on each instance, is stopped after $300 \cdot (n/100)^2$ CPU-seconds.

As the underlying ACO algorithm, we use $\mathcal{MM}$AS as defined for the TSP [11]. In particular, we use the default settings described in Table III, we set $\Delta\tau = 1$ for the amount of pheromone deposited by an ant, and we do not use candidate lists for the solution construction. Following previous work [36], we also incorporate the *pseudo-random action choice rule* of ACS [12], which allows for a greedier solution construction. Parameter q_0 controls the greediness of the pseudo-random action choice rule. A value of $q_0 = 0$ disables it and reverts back to the original $\mathcal{MM}$AS. The MOACO framework is implemented in `C`, and the underlying ACO algorithm is derived from ACOTSP [37]. The code is compiled with `gcc`, version 4.4. All experiments reported in the following are carried out on a single core of Intel Xeon E5410 CPUs, running at 2.33 Ghz with 6MB of cache size under Cluster Rocks Linux version 4.2.1/CentOS 4. The implementation is sequential and experiments run on a single core.

We divide the configuration effort in three stages:

- First, we study whether it is possible to automatically find a novel MOACO design. Hence, we configure only the multi-objective components of the MOACO framework,

[1]The hypervolume is calculated using the algorithm proposed by Fonseca et al. [34].

[2]We use instances `euclidAB100`, `euclidAB300`, `euclidAB500`, `euclidCD100`, `euclidCD300`, `euclidCD500`, `euclidEF100`, `euclidEF300`, `kroAB100`, `kroAB200`, `kroAD100`, and `kroBC100`

Table III
DEFAULT PARAMETER SETTINGS OF THE UNDERLYING ACO ALGORITHM ($\mathcal{MM}$AS).

Parameter	Value
N^{a}	$24 \cdot \lfloor n/100 \rfloor$
ρ	0.02 ($n < 300$), 0.05 ($n \geq 300$)
q_0	0
α	1
β	2

and we compare the results with the algorithms described in the literature.

- Second, we assess how much improvement may be achieved by configuring the parameter settings of the underlying ACO algorithm. We do so by automatically configuring the ACO algorithm settings of one of the configurations found in the previous stage.
- Finally, we ask whether this two-stage configuration approach is better than configuring all components and parameters at once using a configuration effort equivalent to the two previous stages.

A. Configuration of multi-objective components

In the first experiment, our goal is to find a new, hopefully better design of a MOACO algorithm for the bTSP. We search for this design in an automatic fashion by configuring the multi-objective components of the MOACO framework, while keeping fixed the underlying ACO algorithm parameter settings. The fixed ACO algorithm settings are given in Table III and the configuration domain of the multi-objective components is described in Table IV. The configuration budget is set to 1000 runs of the MOACO framework. We perform five independent repetitions of the configuration process using the hypervolume as the evaluation criterion, and another five repetitions using the unary epsilon measure.

The particular configurations found are given in Tables V and VI for the hypervolume and the epsilon measure, respectively. All ten configurations found use multiple heuristic matrices, a large number of colonies, update by region and aggregation by weighted product. Most configurations use multiple pheromone matrices updated by the best-of-objective (BO), and one-weight-per-iteration (1wpi).

We apply these 10 configurations to the test instances, and perform 15 independent runs of each configuration with different random seeds. For comparison, we also evaluate the MOACO algorithms from the literature described in Section IV. We evaluate the quality of the nondominated sets found in these test runs by means of both the hypervolume and the epsilon measure. The full analysis is available as supplementary material [38]. The results lead to the same conclusions independently of the quality measure used in the analysis. Moreover, there were no significant differences, according to the sign test, between configurations found when using the hypervolume as the evaluation criterion in I/F-Race and those found when using the epsilon measure. Hence, we discuss here only the results with respect to the hypervolume measure.

Table IV
DOMAIN OF ALGORITHMIC COMPONENTS OF THE PROPOSED MOACO FRAMEWORK USED FOR THE AUTOMATIC CONFIGURATION.

Component	Domain	Constraint
$[\tau]$	{ single, multiple }	
$[\eta]$	{ single, multiple }	
Aggregation	weighted sum, weighted product, random	only if multiple τ or η
N^{weights}	$\{2, 3, N^{\text{a}}/3, N^{\text{a}}/2, N^{\text{a}}\}$	(per colony)
NextWeight	one weight per iteration, all weights per iteration	
PheromoneUpdate	ND, BO, BOW*	* only with $N^{\text{col}} = 1$
N^{upd}	$\{1, 2, 5, 10\}$	
N^{col}	$\{1, 2, 3, 5, 10\}$	
MultiColonyWeights	{ disjoint, overlapping }	only if $N^{\text{col}} > 1$
MultiColonyUpdate	{ origin, region }	only if $N^{\text{col}} > 1$

Figure 4 displays, for each instance, one boxplot per configuration (y-axis) summarizing the hypervolume value (x-axis) obtained by the 15 independent runs. The boxplots are often reduced to a single point because the variance is extremely small across runs. The results obtained by the MOACO algorithms from the literature are consistent with previous works [4, 14], with the multi-colony variant of BicriterionAnt being the best. The improvement of the configurations obtained automatically, MOACO (1,2,3,4,5), over the MOACO algorithms from the literature is significantly large and consistent. In other words, our method is able to find a new design that surpasses the current state-of-the-art in the MOACO literature. Not surprisingly, this new design is similar to the multi-colony variant of BicriterionAnt. Yet, it includes significant differences, such as the use of best-of-objective update, a small value of N^{upd}, more than one ant per weight, and one-weight-per-iteration (1wpi).

B. Configuration of the underlying ACO algorithm

In the next experiment, we arbitrarily choose one of the configurations found when configuring the multi-objective components of the MOACO framework,[3] and we carry out an automatic configuration of the settings of the underlying ACO algorithm. The domains of the ACO algorithm settings are described in Table VII. Instead of configuring directly the number of ants per colony N^{a}, we configure a surrogate parameter a_f that defines the number of ants in dependence of the instance size n, that is, $N^{\text{a}} = 6 \cdot a_f \cdot \lfloor n/100 \rfloor$. We multiply this factor by six in order to be able to define N^{weights} as either $N^{\text{a}}/3$ or $N^{\text{a}}/2$ in later experiments.

We perform five independent repetitions of the configuration process using the hypervolume as the evaluation criterion, and

[3] Specifically, run 5 of Table V.

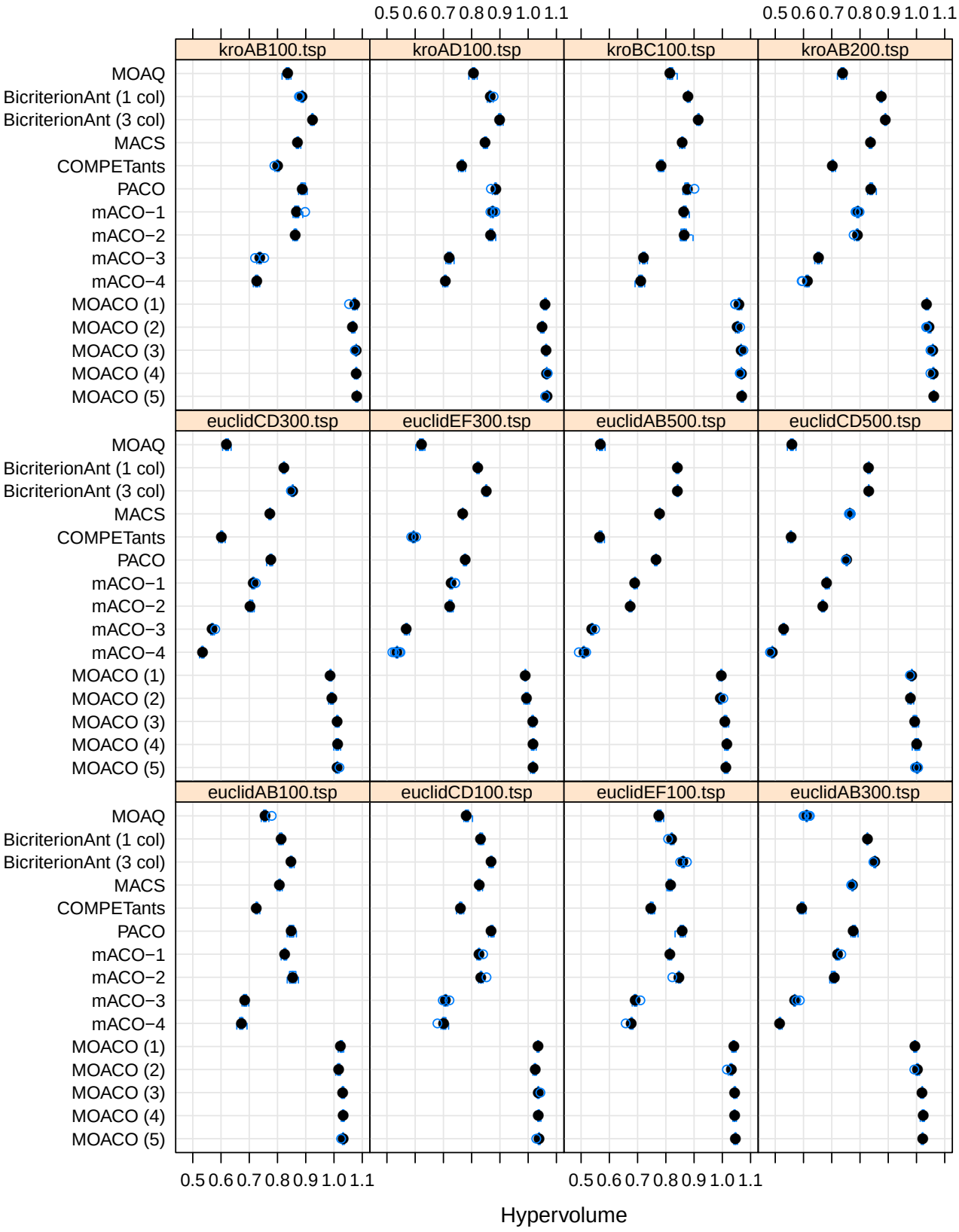


Figure 4. Boxplot of the hypervolume (larger is better) of 15 repetitions of each configuration of the MOACO framework on the test instances. The configurations from the literature are MOAQ, BicriterionAnt with 1 and 3 colonies, COMPETants, PACO, MACS, and the four variants of mACO (Section IV). MOACO (1,2,3,4,5) denote configurations found by five independent automatic configuration runs.

Table V
CONFIGURATIONS OBTAINED BY FIVE INDEPENDENT TUNER RUNS, WHEN CONFIGURING THE MULTI-OBJECTIVE COMPONENTS OF THE PROPOSED MOACO FRAMEWORK ACCORDING TO THE HYPERVOLUME.

Component	Run 1	Run 2	Run 3	Run 4	Run 5
$[\tau]$	single	multiple	multiple	multiple	multiple
$[\eta]$	multiple	multiple	multiple	multiple	multiple
Aggregation	w. product	w. product	w. product	w. product	w. product
N^{weights}	$N^{\text{a}}/3$	3	$N^{\text{a}}/2$	$N^{\text{a}}/3$	2
NextWeight	1wpi	1wpi	1wpi	1wpi	1wpi
PheromoneUpdate	ND	BO	BO	BO	BO
N^{upd}	1	2	2	2	1
N^{col}	10	5	10	10	10
MultiColonyWeights	disjoint	overlap	disjoint	overlap	overlap
MultiColonyUpdate	region	region	region	region	region

Table VI
CONFIGURATIONS OBTAINED BY FIVE INDEPENDENT TUNER RUNS, WHEN CONFIGURING THE MULTI-OBJECTIVE COMPONENTS OF THE PROPOSED MOACO FRAMEWORK ACCORDING TO THE EPSILON MEASURE.

Component	Run 1	Run 2	Run 3	Run 4	Run 5
$[\tau]$	multiple	single	multiple	multiple	multiple
$[\eta]$	multiple	multiple	multiple	multiple	multiple
Aggregation	w. product	w. product	w. product	w. product	w. product
N^{weights}	2	$N^{\text{a}}/3$	$N^{\text{a}}/2$	N^{a}	N^{a}
NextWeight	awpi	1wpi	1wpi	1wpi	1wpi
PheromoneUpdate	BO	ND	BO	BO	BO
N^{upd}	1	1	2	2	2
N^{col}	10	10	5	10	10
MultiColonyWeights	overlap	overlap	overlap	overlap	overlap
MultiColonyUpdate	region	region	region	region	region

Table VII
DOMAIN OF THE PARAMETER SETTINGS OF THE UNDERLYING ACO ALGORITHM ($\mathcal{MM}$AS) USED FOR AUTOMATIC CONFIGURATION.

Component	Domain
a_f	$\{1, 2, \ldots, 25\}$ where $N^{\text{a}} = 6 \cdot a_f \cdot \lfloor n/100 \rfloor$
ρ	$[0.01, 0.99]$
q_0	$[0.25, 0.99]$ or $q_0 = 0$
α	$[0, 5]$
β	$[0, 5]$

each repetition is stopped after 1000 runs of the MOACO framework. For comparison, we also configure in this way the ACO algorithm settings of the multi-colony version of BicriterionAnt, which was found to be the best MOACO algorithm from the literature in the previous section. The resulting parameter settings are again given as supplementary material [38].

One may ask whether it would not be better to configure both the multi-objective components and the ACO parameter settings of the MOACO framework at once, instead of consecutively as done above. To answer this question, we carry out the automatic configuration of all parameters of the MOACO framework at once, with the domains described in Tables IV and VII, and with a budget of 2000 experiments, that is, equivalent to the combined effort spent in the previous two-stage automatic configuration process. The resulting configurations are given as supplementary material [38]. The results presented below show that there is a significant difference in favor of automatically configure all parameters at once.

For the comparison of the configurations found in the various automatic configuration procedures, we perform 15 independent runs of each configuration on the test instances. We evaluate the results using the hypervolume measure, and the corresponding boxplots are shown in Fig. 5.[4] The plots show that automatically configuring the ACO algorithm settings of BicriterionAnt greatly improves its performance, and for instances with 500 cities, it surpasses the quality of the MOACO configuration with default ACO algorithm settings. Nonetheless, by automatically configuring the settings of the ACO algorithm underlying the MOACO framework, the improvement in hypervolume is even more remarkable, clearly outperforming the best configurations of BicriterionAnt. The plot does not show a clear difference between automatically configuring the MOACO framework using the two-stage approach or by configuring all parameters at once. However, configurations obtained by a two-stage approach obtain a lower hypervolume, i.e., they are worse, than those fully configured in 80% of the runs. The sign-test confirms that this difference is statistically significant.

VI. MOACO FRAMEWORK + LOCAL SEARCH (MOACO+LS)

The results presented above conclusively show that automatic configuration leads to better designs of MOACO algorithms than those presented in the literature. Although a large amount of work on MOACO algorithms do not consider

[4]Similar figures that include results with respect to the epsilon measure are available as supplementary material [38].

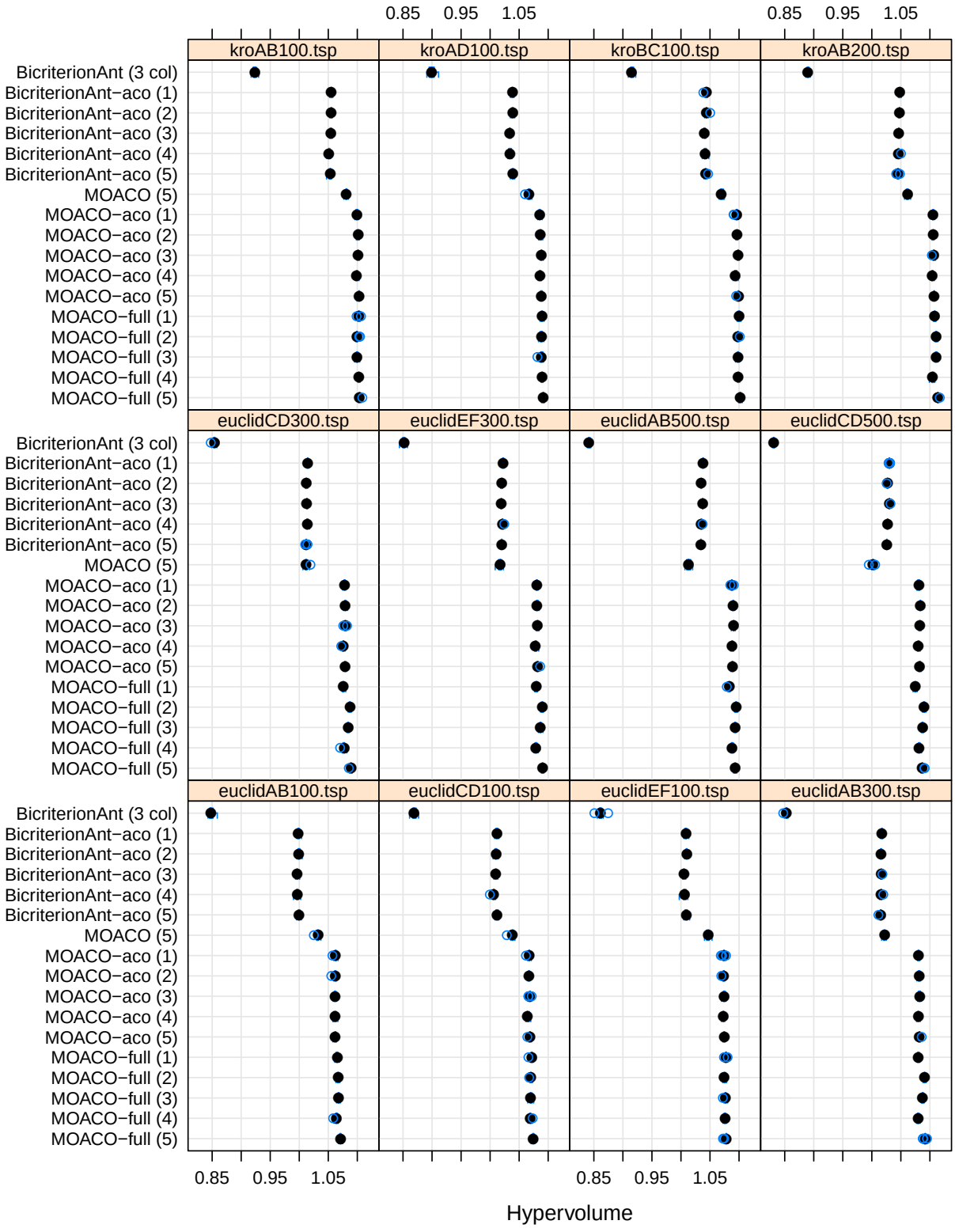


Figure 5. Boxplot of the hypervolume (larger is better) of 15 repetitions of each configuration of the MOACO framework on the test instances. BicriterionAnt (3 col) denotes BicriterionAnt with 3 colonies and default ACO algorithm settings; BicriterionAnt-aco (1,2,3,4,5) denote the same configuration but with automatically configured ACO settings (five independent configuration runs); MOACO (5) denotes the configuration obtained when configuring the multi-objective components with default ACO settings; MOACO-aco (1,2,3,4,5) denote the same configuration but with automatically configured ACO settings (five independent configuration runs); and MOACO-full (1,2,3,4,5) denote five configurations obtained by automatically configuring all parameter settings.

the usage of local search [4, 9], it is known that, on the bTSP, MOACO algorithms with local search by far outperform those without local search [13]. Therefore, we repeat the automatic configuration hybridizing the MOACO framework and local search.

A. Experimental Setup

The local search used here is an iterative improvement algorithm based on the 2-exchange neighborhood (2-opt). Local search is applied to each solution constructed by an ant. The bTSP is converted to a single-objective TSP by means of a weighted sum aggregation of the two distance matrices. If an ant uses a weight vector to construct a solution, the subsequent local search will use the same weight vector. Otherwise, if ants do not use weights for solution construction, a weight is still assigned to each ant following the settings described in Section III but they are only used by the local search. The local search also exploits standard speed-up techniques for the TSP, e.g. an ordered list of candidate edges of size 20 is computed for each weight vector. The ants use a different candidate list of size 20 for construction, which is obtained by sorting edges according to dominance ranking [39].

The remaining parameters of the MOACO framework are equal to the previous settings without local search (Tables III, IV, and VII) except that: *(i)* the default value of ρ is 0.2; *(ii)* the number of ants does not depend on the instance size and it is now calculated as $N^{\mathrm{a}} = 6 \cdot a_f$; and *(iii)* the time limit of each run is now $4 \cdot (n/100)^2$.

As training instances, we generated 10 bTSP random uniform Euclidean instances for each of $n = \{500, 600, 700, 800, 1000\}$ nodes (60 instances in total). The comparison and evaluation of the configurations obtained use a different set of 15 test instances, 3 instances of each size n.

B. Automatic configuration of MOACO+ls

We carry out four separate automatic configuration setups of the MOACO+ls framework for the bTSP.

1) We configure the multi-objective components of the MOACO framework and use default settings for the underlying ACO algorithm.
2) We configure the settings of the underlying ACO algorithm and use one of the MOACO designs found in the previous step.
3) We configure all settings of the MOACO framework at once using two times the budget of experiments as for the previous two steps. The aim of this setup is to investigate whether a two-stage configuration approach is more or less effective than configuring all parameters at once.
4) We configure the ACO algorithm settings of BicriterionAnt+ls for comparison with the other results.

We perform five repetitions of each automatic configuration setup, and the budget of each repetition is set to 1000 experiments (2000 for configuring all settings at once). The configuration tool uses the hypervolume measure as the evaluation criterion. For the sake of conciseness, the particular configurations found are reported in the supplementary pages [38]. It is enough to say here that the configurations show a larger variability of settings, explained by the fact that the local search reduces the effect of other parameters.

The configurations found automatically are evaluated on the test instances by performing 15 independent runs of each configuration, and computing the hypervolume measure of the resulting nondominated sets. Figure 6 summarizes with boxplots the hypervolume values of each configuration on each instance. The results follow the conclusions outlined in the previous section. First, the performance of BicriterionAnt is greatly improved by configuring its ACO algorithm settings. Second, despite this improvement, the automatically configured BicriterionAnt does not outperform an automatically-found design of the MOACO framework with default ACO algorithm settings. Moreover, automatically configuring the ACO algorithm settings of such a good design leads to further improved quality, clearly outperforming the best results of BicriterionAnt. Finally, although the boxplots do not visually show clear differences between the two-stage configuration process versus fully configuring the MOACO+ls framework at once, a sign-test indicates that there is a significantly larger probability, about 0.84, of obtaining a larger hypervolume with a configuration obtained by the latter method. This is consistent with the conclusions reached when automatically configuring the MOACO framework without local search.

These results clearly indicate that we have found new designs of MOACO algorithms that outperform the previous state of the art in the MOACO literature for the bTSP. More importantly, we have found these designs in a semi-automatic fashion, where most of the effort has been spent on the definition of the alternative design choices, leaving the task of finding the correct design to automatic and, hence, unbiased tools. The fact that some design choices never appear in some configurations gives a clear indication that they are not contributing to the performance of the algorithm. On the other hand, the variability of designs indicates that there are several alternative designs that produce similar quality of results.

VII. Conclusion

Instead of proposing a new MOACO algorithm that is slightly different from those found in the literature, we have proposed a novel approach to automatically design MOACO algorithms for multi-objective optimization problems. We have synthesized the existing knowledge from the MOACO algorithms found in the literature into a flexible, configurable framework, and we have explored the vast space of potential algorithmic designs in an automatic fashion by using offline configuration tools.

In this paper, we have shown that such an approach has both scientific and practical advantages. From a scientific standpoint, the process of synthesis involved in constructing the framework has led to the identification of many more commonalities in existing MOACO algorithms than previously thought. In some cases, some algorithms were found to be specializations of other algorithms, despite the fact that both

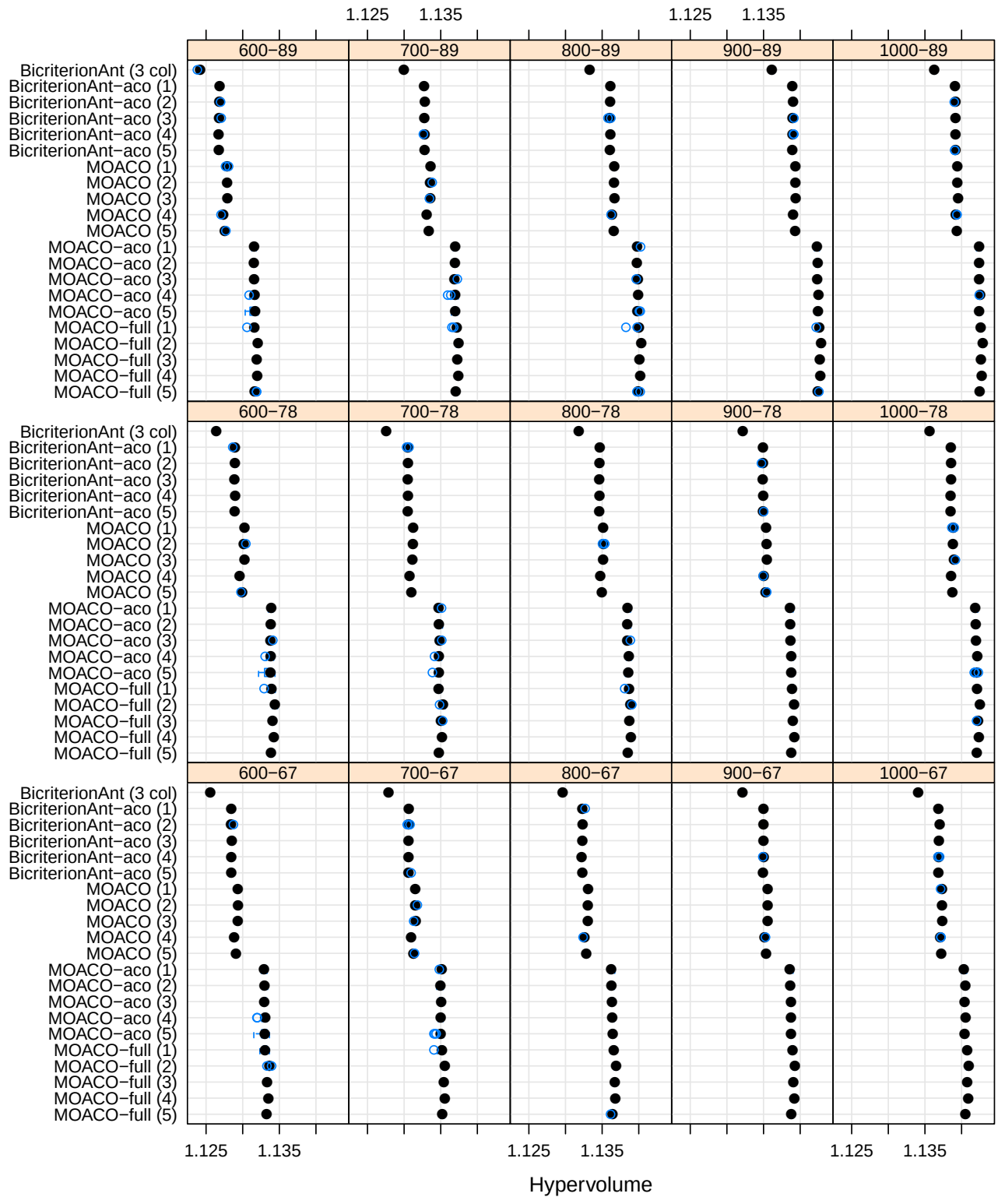


Figure 6. Boxplot of the hypervolume (larger is better) of 15 repetitions of each configuration of the MOACO+ls framework on the test instances. BicriterionAnt (3 col) denotes BicriterionAnt with 3 colonies and default ACO algorithm settings; BicriterionAnt-aco (1,2,3,4,5) denote the same configuration but with automatically configured ACO settings (five independent configuration runs); MOACO (1,2,3,4,5) denote five configurations obtained by automatically configuring the multi-objective components with default ACO settings; MOACO-aco (1,2,3,4,5) denote the configuration MOACO (5) but with automatically configured ACO settings (five independent configuration runs); and MOACO-full (1,2,3,4,5) denote five configurations obtained by automatically configuring all parameter settings.

were independently proposed in the literature. In addition, the integration of design choices from diverse algorithms within the same framework leads to a large number of potential new designs, and each of them can be considered a new MOACO algorithm by itself.

From a practical point of view, testing all possible designs on different problems would be a major effort. The use of automatic configuration tools greatly simplifies this task by identifying which design choices produce the best results for a particular problem. Our experimental results have shown that such an automatically configured design outperforms the MOACO algorithms found in the literature, even after the settings of the latter have been properly configured (which is almost never the case in the MOACO literature). In this sense, we conclude that we have designed a new state-of-the-art MOACO algorithm for the bTSP.

There are several research questions that we have left open in this paper. First, further analysis of the best configurations obtained should shed some light on the causes behind the good performance of some design choices. Second, there are open questions about the best way to automatically configure multi-objective optimization algorithms. Our experiments have not found any significant difference between using the hypervolume or the epsilon measure as the measure of the quality of configurations; further research is needed to confirm whether this generalizes to other problems or algorithms. Third, we have found that there is a small advantage in configuring all settings at once rather than configuring first the "design" of a MOACO algorithm and then its ACO settings. This may be due to interactions between the different components. More research is needed to confirm whether this result generalizes to other configuration scenarios.

In the future, we will apply the same technique to other bi-objective problems already tackled by MOACO algorithms. We make available our implementation of the MOACO framework for the bTSP,[5] and we welcome extensions of the proposed framework, either by applying it to new problems or by extending its components to include recent advances in MOACO algorithms. We expect that, properly configured, the MOACO framework will outperform any other MOACO algorithm, or at worst, match their results. The current MOACO framework, as most MOACO algorithms in the literature, is limited to bi-objective problems. More work is needed to apply ACO algorithms to problems with three and more objectives. However, the main ideas behind automatic design of multi-objective algorithms presented here can be extended to other metaheuristics, such as multi-objective evolutionary algorithms, where various design choices exist without a clear configuration of choices regarded as the best.

ACKNOWLEDGMENT

This work was supported by the European Research Council through the ERC Advanced Grant E-SWARM (contract 246939), and by the Meta-X project, funded by the Scientific Research Directorate of the French Community of Belgium. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Associate. The authors also acknowledge support from the FRFC project "*Méthodes de recherche hybrides pour la résolution de problèmes complexes*".

[5]Source code is available at http://iridia.ulb.ac.be/~manuel/moaco

REFERENCES

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

[2] T. Stützle, M. López-Ibáñez, and M. Dorigo, "A concise overview of applications of ant colony optimization," in *Wiley Encyclopedia of Operations Research and Management Science*, J. J. Cochran, Ed. John Wiley & Sons, 2011, vol. 2, pp. 896–911.

[3] D. Angus and C. Woodward, "Multiple objective ant colony optimization," *Swarm Intelligence*, vol. 3, no. 1, pp. 69–85, 2009.

[4] C. García-Martínez, O. Cordón, and F. Herrera, "A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP," *European Journal of Operational Research*, vol. 180, no. 1, pp. 116–148, 2007.

[5] K. F. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer, "Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection," *Annals of Operations Research*, vol. 131, pp. 79–99, 2004.

[6] B. Barán and M. Schaerer, "A multiobjective ant colony system for vehicle routing problem with time windows," in *Proceedings of the Twentyfirst IASTED International Conference on Applied Informatics*, Insbruck, Austria, 2003, pp. 97–102.

[7] S. Iredi, D. Merkle, and M. Middendorf, "Bi-criterion optimization with multi colony ant algorithms," in *Evolutionary Multi-criterion Optimization (EMO 2001)*, ser. Lecture Notes in Computer Science, E. Zitzler, K. Deb, L. Thiele, C. A. Coello, and D. Corne, Eds. Springer, Heidelberg, Germany, 2001, vol. 1993, pp. 359–372.

[8] M. López-Ibáñez, L. Paquete, and T. Stützle, "On the design of ACO for the biobjective quadratic assignment problem," in *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004*, ser. Lecture Notes in Computer Science, M. Dorigo *et al.*, Eds. Springer, Heidelberg, Germany, 2004, vol. 3172, pp. 214–225.

[9] I. Alaya, C. Solnon, and K. Ghédira, "Ant colony optimization for multi-objective optimization problems," in *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*. Los Alamitos, CA: IEEE Computer Society Press, 2007, vol. 1, pp. 450–457.

[10] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, vol. 26, no. 1, pp. 29–41, 1996.

[11] T. Stützle and H. H. Hoos, "$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 889–914, 2000.

[12] M. Dorigo and L. M. Gambardella, "Ant Colony System: A cooperative learning approach to the traveling

salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[13] M. López-Ibáñez and T. Stützle, "An analysis of algorithmic components for multiobjective ant colony optimization: A case study on the biobjective TSP," in *Artificial Evolution: 9th International Conference, Evolution Artificielle, EA, 2009*, ser. Lecture Notes in Computer Science, P. Collet, N. Monmarché, P. Legrand, M. Schoenauer, and E. Lutton, Eds. Springer, Heidelberg, Germany, 2010, vol. 5975, pp. 134–145.

[14] ——, "The impact of design choices of multi-objective ant colony optimization algorithms on performance: An experimental study on the biobjective TSP," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2010*, M. Pelikan and J. Branke, Eds. New York, NY: ACM press, 2010, pp. 71–78.

[15] ——, "Automatic configuration of multi-objective ACO algorithms," in *Swarm Intelligence, 7th International Conference, ANTS 2010*, ser. Lecture Notes in Computer Science, M. Dorigo *et al.*, Eds. Springer, Heidelberg, Germany, 2010, vol. 6234, pp. 95–106.

[16] A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown, "SATenstein: Automatically building local search SAT solvers from components," in *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009, pp. 517–524.

[17] P. Balaprakash, M. Birattari, and T. Stützle, "Improvement strategies for the F-race algorithm: Sampling design and iterative refinement," in *Hybrid Metaheuristics*, ser. Lecture Notes in Computer Science, T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, Eds. Springer, Heidelberg, Germany, 2007, vol. 4771, pp. 108–122.

[18] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari, "The irace package, iterated race for automatic algorithm configuration," IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2011-004, 2011. [Online]. Available: http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf

[19] S. Wessing, N. Beume, G. Rudolph, and B. Naujoks, "Parameter tuning boosts performance of variation operators in multiobjective optimization," in *Parallel Problem Solving from Nature, PPSN XI*, ser. Lecture Notes in Computer Science, R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, Eds. Springer, Heidelberg, Germany, 2010, vol. 6238, pp. 728–737.

[20] M. Birattari, *Tuning Metaheuristics: A Machine Learning Perspective*, ser. Studies in Computational Intelligence. Berlin / Heidelberg: Springer, 2009, vol. 197.

[21] H. H. Hoos and T. Stützle, *Stochastic Local Search—Foundations and Applications*. San Francisco, CA: Morgan Kaufmann Publishers, 2005.

[22] M. Ehrgott and X. Gandibleux, "Approximative solution methods for combinatorial multicriteria optimization," *TOP*, vol. 12, no. 1, pp. 1–88, 2004.

[23] L. Paquete and T. Stützle, "Design and analysis of stochastic local search for the multiobjective traveling salesman problem," *Computers & Operations Research*, vol. 36, no. 9, pp. 2619–2631, 2009.

[24] T. Lust and J. Teghem, "Two-phase Pareto local search for the biobjective traveling salesman problem," *Journal of Heuristics*, vol. 16, no. 3, pp. 475–510, 2010.

[25] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Evolutionary Methods for Design, Optimisation and Control*, K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papaliliou, and T. Fogarty, Eds. CIMNE, Barcelona, Spain, 2002, pp. 95–100.

[26] K. F. Doerner, R. F. Hartl, and M. Reimann, "Are CompetAnts more competent for problem solving? The case of a multiple objective transportation problem," *Central European Journal for Operations Research and Economics*, vol. 11, no. 2, pp. 115–141, 2003.

[27] L. M. Gambardella, É. D. Taillard, and G. Agazzi, "MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw Hill, London, UK, 1999, pp. 63–76.

[28] M. Guntsch and M. Middendorf, "Solving multi-objective permutation problems with population based ACO," in *Evolutionary Multi-criterion Optimization (EMO 2003)*, ser. Lecture Notes in Computer Science, C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, Eds., vol. 2632. Springer, Heidelberg, Germany, 2003, pp. 464–478.

[29] D. Angus, "Population-based ant colony optimisation for multi-objective function optimisation," in *Progress in Artificial Life*, ser. Lecture Notes in Computer Science, vol. 4828. Springer, Heidelberg, Germany, 2007, pp. 232–244.

[30] C. E. Mariano and E. Morales, "MOAQ: An Ant-Q algorithm for multiple objective optimization problems," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999*, W. Banzhaf, J. M. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. J. Jakiela, and R. E. Smith, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 1999, pp. 894–901.

[31] M. Schilde, K. F. Doerner, R. F. Hartl, and G. Kiechle, "Metaheuristics for the bi-objective orienteering problem," *Swarm Intelligence*, vol. 3, no. 3, pp. 179–201, 2009.

[32] M. Birattari, P. Pellegrini, and M. Dorigo, "On the invariance of ant colony optimization," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 732–742, 2007.

[33] 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. 311–336.

[34] C. M. Fonseca, L. Paquete, and M. López-Ibáñez, "An improved dimension-sweep algorithm for the hypervolume indicator," in *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*. Piscataway, NJ: IEEE Press, Jul. 2006, pp. 1157–1163.

[35] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[36] T. Stützle and H. H. Hoos, "$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System and local search for combinatorial optimization problems," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. Osman, and C. Roucairol, Eds. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999, pp. 137–154.

[37] T. Stützle, "ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem," 2002. [Online]. Available: http://www.aco-metaheuristic.org/aco-code/

[38] M. López-Ibáñez and T. Stützle, "The automatic design of multi-objective ant colony optimization algorithms: Supplementary material," 2011. [Online]. Available: http://iridia.ulb.ac.be/supp/IridiaSupp2011-007/Iridia-2011-007.pdf

[39] T. Lust and A. Jaszkiewicz, "Speed-up techniques for solving large-scale biobjective TSP," *Computers & Operations Research*, vol. 37, no. 3, pp. 521–533, 2010.

Manuel López-Ibáñez received the M.S. degree in computer science from the University of Granada, Granada, Spain, in 2004, and the Ph.D. degree from Edinburgh Napier University, Edinburgh, U.K., in 2009.

He is currently a Postdoctoral Researcher (Chargé de recherche) of the Belgian F.R.S.-FNRS at the Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Université libre de Bruxelles, Brussels, Belgium. His current research interests are the engineering, experimental analysis and automatic configuration of stochastic optimization algorithms for single and multi-objective optimization problems.

Thomas Stützle received the Diplom, M.S. degree, in business engineering from the Universität Karlsruhe (TH), Karlsruhe, Germany in 1994, and the Ph.D. degree and the "Habilitation" in computer science both from the Computer Science Department of Technische Universität Darmstadt, Darmstadt, Germany in 1998 and 2004, respectively.

He is currently a Research Associate of the Belgian F.R.S.-FNRS working in the Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Université libre de Bruxelles, Brussels, Belgium. He is author of the two books: *Stochastic Local Search: Foundations and Applications* (Morgan Kaufmann) and *Ant Colony Optimization* (MIT Press). He has published extensively in the wider area of metaheuristics (more than 150 peer-reviewed articles in journals, conference proceedings, or edited books). His research interests range from stochastic local search (SLS) algorithms, large scale experimental studies, automated design of algorithms, to SLS algorithms engineering.