Automatic Generation of Multi-objective ACO Algorithms for the Bi-objective Knapsack

Leonardo C. T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
leonardo@iridia.ulb.ac.be, {manuel.lopez-ibanez, stuetzle}@ulb.ac.be

Abstract. Multi-objective ant colony optimization (MOACO) algorithms have shown promising results for various multi-objective problems, but they also offer a large number of possible design choices. Often, exploring all possible configurations is practically infeasible. Recently, the automatic configuration of a MOACO framework was explored and was shown to result in new state-of-the-art MOACO algorithms for the bi-objective traveling salesman problem. In this paper, we apply this approach to the bi-objective bidimensional knapsack problem (bBKP) to prove its generality and power. As a first step, we tune and improve the performance of four MOACO algorithms that have been earlier proposed for the bBKP. In a second step, we configure the full MOACO framework and show that the automatically configured MOACO framework outperforms all previous MOACO algorithms for the bBKP as well as their improved variants.

1 Introduction

Multi-objective ant colony optimization (MOACO) algorithms have been applied to multi-objective combinatorial optimization problems (MCOPs) since more than 10 years [7, 3, 1, 10, 12, 9]. The interest in MOACO algorithms may be explained by the practical relevance of multi-objective problems and by the positive results that have been achieved with these algorithms. The available MOACO algorithms provide a large number of different design choices that allow the instantiation of a huge number of structurally different MOACO algorithms. Recently, López-Ibáñez and Stützle [13] proposed a MOACO framework that implements most of those design possibilities. The automatic configuration tool *Iterated F-race* (I/F-Race) [2, 11] was used to automatically generate MOACO algorithms for the bi-objective traveling salesman problem (bTSP). The authors showed that the automatic configuration of a generic MOACO framework produced better results than the MOACO algorithms from the literature used to build the framework. In this paper, we continue the investigation of the effectiveness of this approach by extending the MOACO framework to deal with the bi-objective bidimensional knapsack problem (bBKP).

The bBKP is a popular benchmark problem in multi-objective optimization [16, 14]. Moreover, four different MOACO algorithms have been proposed for the bBKP [1]. The bBKP has also some properties that make it interesting for further exploring the possibilities of the automatic design of MOACO algorithms from a flexible framework. In particular, the representation of solutions is different from the TSP, pheromone information is represented by a vector instead of a matrix, and the structure of the solution space is quite different from the TSP.

This paper shows that the proposed method for the automatic design of MOACO algorithms also works for the bBKP. The proposed method is able to generate, with little effort from the human designer, MOACO algorithms that are clearly better than those proposed earlier for the bBKP, even after tuning the ACO settings of the MOACO algorithms from the literature and improving significantly their performance.

2 The Bi-objective Bidimensional Knapsack Problem

In an MCOP, the quality of solutions is evaluated based on a D-dimensional objective vector. Given two different candidate solutions x_1 and x_2 of a maximization problem, the Pareto dominance relation states that x_1 *dominates* x_2 iff $\forall d = 1, \ldots, D \; f^d(x_1) \geq f^d(x_2)$, and $\exists j \in \{1, \ldots, D\}$ such that $f^d(x_1) > f^d(x_2)$. The goal in MCOPs that are tackled according to Pareto dominance is to identify the Pareto-optimal set, i.e., the solutions that are *nondominated* w.r.t. all feasible solutions. Since most of such MCOPs are NP-hard, this goal is typically relaxed towards finding an as good as possible approximation to the Pareto set.

In this paper, we tackle the bBKP, which is a widely used bi-objective benchmark problem [16, 14]. The bBKP is a special case of the general multi-objective multidimensional knapsack problem (moMKP), which is formalized as follows:

$$\max \; f^d(x) = \sum_{i=1}^{n} p_i^d x_i \quad d = 1, \ldots, D \qquad \text{s.t.} \quad \sum_{i=1}^{n} w_i^j x_i \leq W_j \quad j = 1, \ldots, m$$

where each item i has D profits and m costs, f^d is the d-th component of the D-dimensional objective vector f, n is the number of items, p_i^d is the d-th profit of item i, w_i^j is the j-th cost of item i, W_j is the j-th capacity of the knapsack, and x_i is a decision variable in $\{0, 1\}$ that controls whether item i is included in the knapsack ($x_i = 1$) or not ($x_i = 0$). The set of feasible solutions is $X \subseteq \{0, 1\}^n$. The bBKP is a special case of the moMKP where $D = m = 2$.

3 ACO Algorithms for the bBKP

When applying ACO to the single-objective multidimensional knapsack problem, the pheromone information is defined as a vector, where each component τ_i gives the desirability of adding item i to the knapsack. Each ant k constructs a solution by adding, at each step, item i to the knapsack with a probability p_i

$$p_i = \begin{cases} \frac{\tau_i^\alpha \cdot \eta_i^\beta}{\sum_{j \in N^k} \tau_j^\alpha \cdot \eta_j^\beta} & \forall i \in N^k, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where η_i is a heuristic estimation of the benefit of adding item i, and N^k is a set of candidate items. After each step, the item added to the current solution and those items that do not fit anymore in the remaining capacity of the knapsack are removed from the candidate set. The solution construction stops when the candidate set is empty. After the constructed solutions are evaluated, the pheromone information is updated in two steps. First, pheromone values are evaporated, that is, decreased by a factor ρ. Second, the pheromone values corresponding to items present in the best solutions are updated by depositing an amount of pheromone $\Delta\tau$, thus increasing the probability that newly constructed solutions contain those items. Alaya et al. [1] proposed four different algorithms that extend the ACO metaheuristic to the bBKP.

mACO$_1$ has one pheromone vector for each objective, that is, τ^1 and τ^2. Ants are divided in three groups $\lambda \in \{0, 0.5, 1\}$ according to the weight λ they use for aggregating the two pheromone vectors when constructing solutions. The solution construction uses *random aggregation*, that is, at each step the pheromone information to be used is chosen as τ^1 with a probability $(1 - \lambda)$, and as τ^2, otherwise. This means that ants using $\lambda = 0$ or $\lambda = 1$ use only τ^1 or τ^2, respectively. The heuristic information is aggregated by means of *weighted sum aggregation*, that is, $\eta = (1 - \lambda) \cdot \eta^1 + \lambda \cdot \eta^2$, where η^1 and η^2 are the heuristic information corresponding to each objective.

The pheromone update method used by mACO$_1$ is a particular case for $\lambda \in \{0, 0.5, 1\}$ of a method called *best-of-objective-per-weight* (BOW) [13]. In BOW, those solutions generated with the same weight λ are kept in the same list. For the lists of $\lambda \notin \{0, 1\}$, the best solution according to each objective updates the pheromone vector of the corresponding objective. For the list of $\lambda = 0$, only the best solution according to the first objective updates τ^1, whereas for the list of $\lambda = 1$, only the best solution according to the second objective updates τ^2.

Finally, mACO$_1$ uses a particular pheromone deposit. Given the best solution constructed in the current iteration and the best-so-far solution according to objective d (s^d_{ib} and s^d_{bf}, respectively), the amount of pheromone deposited is given by $\Delta\tau^d = \frac{1}{1+f^d(s^d_{\text{bf}})-f^d(s^d_{\text{ib}})}$. We refer to this method as *fobj-mACO*.

mACO$_2$ is identical to mACO$_1$ except for how the multiple pheromone vectors are aggregated. Instead of a random aggregation, mACO$_2$ uses a weighted sum aggregation, that is, $\tau = (1 - \lambda) \cdot \tau^1 + \lambda \cdot \tau^2$.

mACO$_3$ uses only a single pheromone vector. The heuristic information is also a single vector, which is statically computed at the start of the algorithm as $\eta_i = \eta^1_i + \eta^2_i$. Pheromone information is updated using all nondominated solutions found since the start of the algorithm, that is, the best-so-far archive. Every solution component is rewarded a constant $\Delta\tau = 1$ only once per iteration, regardless of how many times it is present on different solutions.

mACO$_4$ follows mACO$_1$: one pheromone vector per objective, which are aggregated by weighted random aggregation; BOW pheromone update, and pheromone deposit is *fobj-mACO*. However, there is only one weight $\lambda = 0.5$, and one heuristic vector defined as in mACO$_3$.

Algorithm 1 MOACO framework

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

The mACO algorithms can be instantiated as described above by our MOACO framework [13]. We have confirmed this approach is equivalent to the original [1].

4 A Flexible MOACO Framework for the bBKP

In this paper, we extend the flexible MOACO framework proposed for the bTSP by López-Ibáñez and Stützle [13] to also tackle the bBKP and we automatically instantiate MOACO algorithms. The MOACO framework is able to replicate most MOACO designs proposed in the literature and can generate new MOACO designs by combining components in novel ways. However, its application to the bBKP requires extending it concerning the solution representation and other problem-specific features. Here, we briefly summarize the high-level structure of the framework and its components (see [13] for further details).

The high-level algorithmic scheme of the MOACO framework is given in Algorithm 1. The MOACO framework is a multi-colony algorithm, where each colony c of ants has its own pheromone information and its own set of weights Λ_c for possibly aggregating information. The assignment of weights to colonies is defined by MOACO component MultiColonyWeights. Within each colony, each ant constructs a solution according to pheromone information τ and heuristic information η. Either τ or η may be the result of aggregation. That is, if the pheromone information consists of multiple pheromone vectors, one for each objective, these are aggregated into a single pheromone vector τ by means of MOACO component Aggregation (line 12), using a particular weight λ. If multiple heuristic vectors are used, they are aggregated in a similar way. Which weight is used by each ant may depend on the set of weights of each colony, the particular ant, and the particular iteration. The different possibilities are encapsulated by MOACO component NextWeight (line 11). Once all ants have constructed a solution, the resulting iteration-best archive of nondominated solutions (A^{iter}) is merged into the best-so-far archive (A^{bf}) (line 16). After this step, the pheromone information of each colony is updated in two steps. First, the set of solutions for update (either A^{iter} or A^{bf}), is partitioned among colonies according to component MultiColonyUpdate (line 19). Next, a number of solutions from each set is used to update the pheromone information of each colony in a way defined by component PheromoneUpdate (line 20). The algorithm stops when a termination criterion is met, typically a maximum number of iterations or a time limit, and returns the best-so-far archive.

The flexibility of the MOACO framework is given by the alternative definitions of the algorithmic components that specify the key steps in the algorithm. Defining these components in particular ways allows the framework to replicate most of the MOACO algorithms in the literature. A summary of the available alternatives is given in Table 1. The complete description of all components and their alternatives can be found in the original publication [13]. For brevity, we restrict ourselves here to the new extensions implemented for the bBKP.

Following [13], we use $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [15] as the underlying ACO algorithm that defines details such as the pheromone deposit $\Delta\tau$, and maximum and minimum pheromone levels ($\tau_{\max}$ and $\tau_{\min}$). Here, we have adapted $\mathcal{MM}$AS to the bBKP, but making more flexible the definition of $\Delta\tau$, $\tau_{\max}$ and $\tau_{\min}$ to be able to replicate faithfully the original mACO algorithms for the bBKP. The alternatives implemented for the definition of the pheromone deposit ($\Delta\tau$) are:

fobj, that is, $\Delta\tau^d = f^d(s)$, where τ^d is the pheromone information corresponding to objective d. If only one pheromone vector is used instead of multiple, then $\Delta\tau = f^1(s)+f^2(s)$. This method is the one used in the original $\mathcal{MM}$AS.

constant, that is, $\Delta\tau^d = 1 - \frac{r^d(s)-1}{N^{\mathrm{upd}}}$, where $r^d(s)$ is the rank of solution s ordered according to objective d and N^{upd} is the number of solutions used to update τ^d. This method is inspired by rank-based ant system [5].

fobj-mACO, this is the method used in mACO$_1$, mACO$_2$ and mACO$_4$.

MACS, that is, $\Delta\tau = f^1(s) \cdot f^2(s)$, which is adapted from MACS [3].

For the definition of the pheromone levels we consider two possibilities. The first is the **default** setting of $\mathcal{MM}$AS, which uses $\tau_{\max} = \frac{\max^{iter}(\Delta\tau)}{\rho}$, where $\max^{iter}(\Delta\tau)$ is the maximum amount of pheromone deposited at iteration $iter$ for a single pheromone component, and $\tau_{\min} = \frac{\tau_{\max}}{\nu \cdot n}$, where $\nu \in \mathbb{R}^+$ is a parameter ($\nu = 2$ in $\mathcal{MM}$AS). The second is the **value** setting, where $\tau_{\max}$ and $\tau_{\min}$

Table 1: Algorithmic components of the MOACO framework

Component	**Domain**	**Description**
$[\tau]$	{ single, multiple }	Num. pheromone vectors
$[\eta]$	{ single, multiple }	Num. heuristic vectors
N^{weights}	$\mathbb{N}^+$	Number of weights
Aggregation	{ weighted sum, weighted product, random	How weights are used to aggregate multiple $[\tau]$ or $[\eta]$
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 to update the pheromone information
N^{upd}	$\mathbb{N}^+$	Num. solutions that update each $[\tau]$
ChooseUpdateSet	{ best-so-far (BSF), iteration-best (IB), mixed	Whether the solutions used for update are taken from A^{bf}, A^{iter} or using both alternately
The following components have an effect only when using multiple colonies.		
N^{col}	$\mathbb{N}^+$	Number of colonies
MultiColonyWeights	{ same ($\cap_{100\%}$), overlapping ($\cap_{50\%}$), disjoint ($\cap_{0\%}$	Whether colonies share all, 50% or no weights.
MultiColonyUpdate	{ origin, region }	How solutions are assigned to colonies
New components added in this work for the bBKP.		
$\tau_{\max}$ method	{ default, value }	Method for calculating $\tau_{\max}$
$\tau_{\min}$ method	{ default, value }	Method for calculating $\tau_{\min}$
$\Delta\tau$	{ constant, fobj-mACO, fobj, MACS }	Method for calculating $\Delta\tau$
η_i	{ $\frac{\text{profit}}{\text{cost}}$, $\frac{\sum \text{profits}}{\text{cost}}$, $\frac{\text{profit}}{\sum \text{costs}}$ }	Heuristic information used

are set to two different constant values $\tau_{\max} > \tau_{\min}$. A **value** setting is used in all mACO algorithms.

In addition, we have implemented three alternatives for the heuristic information. For a given objective d and item i, the heuristic information can be either profit divided by cost ($\eta 1_i^d$), which is the one used in the mACO algorithms [1], sum profits divided by cost ($\eta 2_i^d$), or profit divided by sum costs ($\eta 3_i^d$) [14], that is,

$$\eta 1_i^d = \frac{p_i^d}{w_i^d} \qquad \eta 2_i^d = \frac{\sum_{k=1}^{D} p_i^k}{w_i^d} \qquad \eta 3_i^d = \frac{p_i^d}{\sum_{l=1}^{m} w_i^l} \tag{2}$$

Table 2: Termination criteria used in our experiments.

	TIME$_1$	**TIME**$_2$	**TIME**$_3$	**TIME**$_4$
Time (s)	$0.00001 \cdot n^2$	$0.00003 \cdot n^2$	$0.0001 \cdot n^2$	$0.001 \cdot n^2$
Equivalent to	9000 solutions of mACO$_2$	3000 solutions of mACO$_1$	30000 solutions of mACO$_3$	300000 solutions of mACO$_4$

5 Experimental Setup

Our experiments are divided in two stages. In a first stage, we automatically configure the ACO settings of the mACO algorithms and compare the resulting configurations with the original settings. This is done to avoid a bias by possibly poor ACO parameter settings of the mACO algorithms. In the second stage, we compare the best configurations with an algorithm automatically instantiated from the MOACO framework.

As the automatic algorithm configuration tool, we use I/F-Race [2, 11]. The input of I/F-Race is a definition of the parameter space, which may contain categorical and numerical parameters, and a set of training instances. I/F-Race was originally designed for single-objective algorithms, but it has been extended to handle the multi-objective case by using the hypervolume quality measure [13] (I_H). The hypervolume is a well-known quality measure in multi-objective optimization [17]. It computes for each approximation set, the volume in the objective space weakly dominated by the approximation set and bounded by a reference point; hence, the larger the hypervolume the better. We use the hypervolume (concretely, the implementation provided by Fonseca et al. [8]) not only in combination with I/F-Race, but also to compare the various MOACO algorithms.

For the application of I/F-Race, we create a training set of 100 randomly generated instances of the bBKP, following the method proposed by Zitzler and Thiele [16]. These instances have random sizes in the range $n \in \{100, \ldots, 750\}$. For comparing the algorithms, we generate a different test set of 50 bBKP instances for each size $n \in \{100, 250, 500, 750\}$. We include in our test set also the four instances by Zitzler and Thiele [16] of sizes $n \in \{100, 250, 500, 750\}$, called ZTZ instances. All algorithms are implemented in `C` and all experiments are run on a single core of Intel Xeon E5410 CPUs, running at 2.33GHz with 6MB of cache size under Cluster Rocks Linux version 4.2.1/CentOS 4.

The mACO algorithms were originally run with different termination criteria, that is, a different number of iterations, for each variant [1]. To replicate the original mACO experiments, we consider four different computation time limits in our experiments, which correspond to the mean time taken by each of the four mACO variants measured across 25 independent runs on the four ZTZ instances using the corresponding number of iterations (see Table 2). Then, we compute a formula that approximates the computation time obtained for each termination criterion. The four resulting termination criteria are given in Table 2, sorted from the shortest to the longest time.

Table 3: Parameter space for tuning the ACO settings of the mACO algorithms.

Parameter	α	β	ρ	q_0	a_f	τ_max method	τ_min method
Domain	$\{0,\dots,10\}$	$\{0,\dots,15\}$	$[0.01,1]$	$[0,0.99]$	$\{1,\dots,30\}$	$\{$*default*, *value*$\}$ *value* $\in [6,100]$	$\{$*default*, *value*$\}$ *value* $\in [0.01,6]$ $\nu \in [1.5,15]$

Comparisons are conducted using empirical attainment functions (EAFs), boxplots of the hypervolume (I^H) and the unary additive epsilon ($I^{\epsilon+}$) indicators [17], and the Friedman non-parametrical test. In the paper, only few representative results are given; for the complete set of results and the test and training instances we generated, we refer to the supplementary material [4].

6 Experimental Analysis

6.1 Improving the ACO settings of the mACO algorithms

In the first stage of our analysis, we automatically configure the ACO settings of the four mACO variants. The parameter space given to I/F-Race is shown in Table 3. Parameter a_f is a surrogate parameter of the total number of ants, which is given by $N^\mathrm{a} = a_\mathrm{f} \cdot (0.12 \cdot n + 36)$. N^a is rounded to the closest smaller number divisible by three, because mACO$_1$ and mACO$_2$ divide the ants into three groups. We apply I/F-Race with a budget of 5 000 independent runs in the tuning phase for each mACO algorithm and for each termination criterion TIME$_i$. Here, the mACO algorithms use their original heuristic information $\eta 1$ [1]. The resulting 16 configurations of mACO are provided as supplementary material [4]. Here, we focus on the configurations obtained when using TIME$_4$, which are shown in Table 4.

We compare all algorithms (original and tuned versions) in terms of the hypervolume. We run all algorithms for all four termination criteria 10 independent times on each of the 200 randomly generated bBKP instances (50 instances per instance size $n \in \{100, 250, 500, 750\}$). We normalize the objective values per instance to the interval $[1, 2]$, with 1 corresponding to the maximum value and 2 to the minimum, and compute the hypervolume using the reference point $(2.1, 2.1)$. To analyze the results, we apply the Friedman test, and its associated post-hoc test for multiple comparisons [6], using the median hypervolume obtained by each algorithm on each instance as values, the instances as the blocking factor and the different mACO algorithms as the treatment factor. In all cases, the Friedman test rejects the null hypothesis of equal performance at a significance level of 0.05. Those algorithms whose ranks differ by more than the critical difference are considered to be significantly different. Table 5 summarizes the results of applying this statistical analysis for each termination criterion. Ranks obtained by each algorithm are shown in parenthesis. The minimum significant rank difference is displayed between parenthesis on the header of each column.

Table 4: Settings chosen by *irace* for mACO$_i$-tuned under TIME$_4$.

Variant	α $\{0, ..., 10\}$	β $\{0, ..., 15\}$	ρ $[0.01, 1]$	q_0 $[0, 0.99]$	τ_{max} **method** $\{$*default*, *value*$\}$	τ_{min} **method** $\{$*default*, *value*$\}$	a_f $\{1, ..., 30\}$
mACO$_1$*-tuned*	8	1	0.03	0.03	*value* $= 65$	*value* $= 0.33$	27
mACO$_2$*-tuned*	3	1	0.07	0.10	*default*	*default*, $\nu = 6$	26
mACO$_3$*-tuned*	3	1	0.08	0.18	*value* $= 49$	*value* $= 0.34$	2
mACO$_4$*-tuned*	2	1	0.19	0.19	*default*	*default*, $\nu = 8$	5

Table 5: Friedman test results for I_H obtained by the mACO algorithms.

Rank	I_H **TIME**$_1$ (32.957)	I_H **TIME**$_2$ (31.793)	I_H **TIME**$_3$ (35.433)	I_H **TIME**$_4$ (40.745)
1	**mACO$_2$-tuned (293)**	**mACO$_2$-tuned (208)**	**mACO$_2$-tuned (220)**	**mACO$_2$-tuned (227)**
2	**mACO$_1$-tuned (319)**	mACO$_1$-tuned (402)	mACO$_1$-tuned (380)	mACO$_1$-tuned (373)
3	mACO$_2$ (591)	mACO$_2$ (610)	mACO$_2$ (644)	mACO$_3$-tuned (757)
4	mACO$_4$-tuned (958)	mACO$_3$-tuned (973)	mACO$_1$ (987)	mACO$_2$ (779)
5	mACO$_3$-tuned (1005)	mACO$_1$ (1036)	mACO$_3$-tuned (1040)	mACO$_4$-tuned (1076)
6	mACO$_3$ (1202)	mACO$_4$-tuned (1087)	mACO$_4$-tuned (1073)	mACO$_1$ (1238)
7	mACO$_1$ (1268)	mACO$_3$ (1301)	mACO$_3$ (1287)	mACO$_3$ (1282)
8	mACO$_4$ (1564)	mACO$_4$ (1583)	mACO$_4$ (1569)	mACO$_4$ (1468)

The best algorithm and those that are not significantly different from the best are marked in boldface.

From Table 5, we observe that mACO$_2$-tuned is the best performing algorithm for all different TIME$_i$, whereas mACO$_4$ performs the worst. This seems to contradict the results reported by Alaya et al. [1], which considered mACO$_4$ as the best performing variant. The different results are explained because, in their case, mACO$_4$ constructed 100 times more solutions than mACO$_2$, which roughly requires 100 times more computational time (Table 2). By contrast, we compare algorithms using the same computation time limit.

The main conclusion we take from these results is that each tuned mACO algorithm clearly outperforms its corresponding original version for each stopping criterion. Hence, we use these tuned variants for comparing against the automatically generated MOACO algorithm in the next section.

6.2 Automatically Generating MOACO Algorithms for the bBKP

In this second stage of our analysis, we automatically configure all parameters of the MOACO framework. In particular, for the parameters specific to the underlying ACO algorithms, we use the same parameter space as for the mACO algorithms (Table 3). For the multi-objective components, we consider all alternatives described in Table 1, plus the following ranges: $N^{\text{col}} \in \{1, 2, 5\}$ and $N^{\text{upd}} \in \{1, \dots, 10\}$. Since N^{a}, the number of ants, has to be divisible by N^{col}, and the result be divisible by N^{weights} (when *awpi* is used), N^{a} was always rounded to the largest smaller number divisible by 10. The weights are defined as, $N^{\text{weights}} \in \{0.2, 5, N^{\text{a}}\}$, when $N^{\text{col}} = 2$, and $N^{\text{weights}} \in \{0.5, 2, N^{\text{a}}\}$, when $N^{\text{col}} = 5$. For single colony versions, only two values were allowed: 0.2 and 0.5.

Table 6: Parameter settings chosen by I/F-Race for AutoMOACO: TIME$_4$.

Parameter	α	β	ρ	q_0	a_f	$\tau_{\max}$	$\tau_{\min}$	N^{col}	N^{weights}	MCWeights	NextWeight	MCUpdate
Value	1	12	0.12	0.57	8	83	2.49	5	N^{a}	$\cap_{50\%}$	*awpi*	*origin*

Parameter	N^{upd}	Selection	Ref.	$\Delta\tau$	$[\tau]$	$[\eta]$	$[\tau]$-Aggreg.	$[\eta]$-Aggreg.	Heuristic
Value	10	BO	BSF	*constant*	*multiple*	*multiple*	*product*	*sum*	$\eta 3$

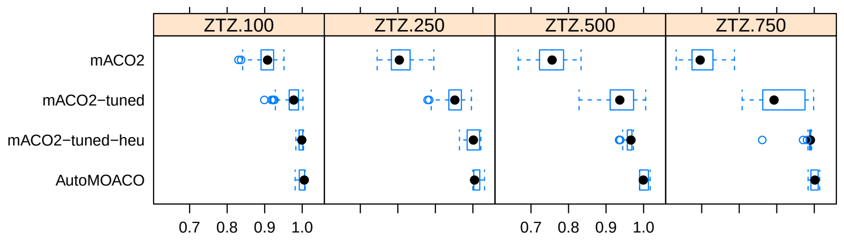

Fig. 1: Boxplots of the I_H indicator for several MOACO algorithms with TIME$_4$.

As in the previous section, we apply I/F-Race four times, once for each stopping criterion. The budget of each run of I/F-Race is 5 000 runs of the MOACO framework. The four resulting configurations are given as supplementary material [4]. Here, we focus on the configuration obtained for TIME$_4$ (Table 6).

The analysis of the AutoMOACO configurations shows several commonalities. First, heuristic $\eta 3$ is always chosen, which is different from the one used in the mACO algorithms. Second, the parameter β is always close to the maximum value allowed, thus giving very high importance to the heuristic information. Third, the parameter value of q_0 is also high. This together with the high value of the parameter β implies that most of the items are chosen greedily. Fourth, the number of ants is always very large. For example, 1000 ants are used for instance size 750. As a result, the number of iterations executed by the MOACO algorithm in the given time limits is rather small. It reaches from at most two iterations for the shortest time limits (TIME$_1$ and TIME$_2$) to about 60 to 85 iterations for the larger time limit (TIME$_4$). In the first case, if very few iterations are executed, the algorithm actually behaves as a greedy construction procedure that performs multiple scalarizations of the bi-objective problem. For the longer time limits, we confirmed that excluding the pheromone information (that is, setting $\alpha = 0$) makes the performance become significantly worse (see supplementary material [4]). This implies that for the larger computation time limits, despite the low number of iterations, the ACO component is effective.

Finally, we compare the performance obtained by the automatically configured MOACO algorithms and the mACO algorithms. Given the high impact of using heuristic information $\eta 3$, we repeated the tuning of each of the mACO variants as described above, but this time leaving open also the choice of the heuristic information. In the following comparison, we consider only the original and the two tuned variants of mACO$_2$, which are the best mACO variants for

Table 7: Friedman test results for I_H for various MOACO algorithms.

Rank	I_H **TIME**$_1$ (14.74)	I_H **TIME**$_2$ (11.416)	I_H **TIME**$_3$ (7.635)	I_H **TIME**$_4$ (5.987)
1	**AutoMOACO (236)**	**AutoMOACO (228)**	**AutoMOACO (212)**	**AutoMOACO (204)**
2	mACO$_2$-tuned-heu (365)	mACO$_2$-tuned-heu (373)	mACO$_2$-tuned-heu (388)	mACO$_2$-tuned-heu (399)
3	mACO$_2$-tuned (611)	mACO$_2$-tuned (599)	mACO$_2$-tuned (600)	mACO$_2$-tuned (597)
4	mACO$_2$ (688)	mACO$_2$ (800)	mACO$_2$ (800)	mACO$_2$ (800)

each of the time limits. In Fig. 1 we show boxplots of the hypervolume distribution for the algorithm automatically instantiated from the MOACO framework (AutoMOACO), the original mACO$_2$, mACO$_2$ tuned with $\eta 1$ and mACO$_2$ tuned leaving open the choice of the heuristic information (mACO$_2$-tuned-heu). The instances shown are the four ZTZ instances. Finally, Table 7 gives the results of the Friedman test, which is applied as described in Section 6.1. Clearly, the AutoMOACO algorithm is the top performer, outperforming significantly the other variants. For complete results, we again refer to the supplementary material [4].

7 Conclusions and Future Work

We have extended the MOACO framework [13] to the bBKP and automatically generated MOACO algorithms. The results reported here for the bBKP confirm the previous conclusions obtained in the bTSP, that is, the automatically configured MOACO algorithms outperform the MOACO algorithms from the literature, even after the ACO parameters of the latter have been tuned with the same effort. Interestingly, the MOACO algorithm tuned for very short time limit is rather a repeated stochastic greedy construction procedure than an ACO algorithm. Although this result may seem counter-intuitive at first, it is, however, a strength of automatic configuration procedures, because they are not biased towards our expectations. The fact that the resulting algorithm is better than the MOACO algorithms proposed in the literature, indicates that the automatic design works as desired, that is, it provides a high-performing algorithm for the given termination criterion. For higher computation time limits, the ACO component of the finally configured algorithm works and contributes to its high performance.

Future work should extend the MOACO framework, and apply the proposed automatic design method, to new problems in order to further confirm the above conclusions. The method is not restricted to MOACO algorithms, and, hence, extensions to other metaheuristics are possible.

Acknowledgments. The research leading to the results presented in this paper has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n^o 246939, and from the Meta-X project from the Scientific Research Directorate of the French Community of Belgium. Leonardo C. T. Bezerra, Manuel López-Ibáñez and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are a FRIA doctoral fellow, a postdoctoral researcher and a research associate, respectively.

References

1. Alaya, I., Solnon, C., Ghédira, K.: Ant colony optimization for multi-objective optimization problems. In: ICTAI 2007, vol. 1, pp. 450–457. IEEE Computer Society Press, Los Alamitos, CA (2007)
2. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein, T., et al. (eds.) Hybrid Metaheuristics, LNCS, vol. 4771, pp. 108–122. Springer (2007)
3. Barán, B., Schaerer, M.: A multiobjective ant colony system for vehicle routing problem with time windows. In: Proceedings of the Twenty-first IASTED Intern. Conf. on Appl. Informat.. pp. 97–102. Insbruck, Austria (2003)
4. Bezerra, L.C.T., López-Ibáñez, M., Stützle, T.: Automatic Generation of MOACO Algorithms for the Biobjective Bidimensional Knapsack Problem: Supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2012-008/ (2012)
5. Bullnheimer, B., Hartl, R., Strauss, C.: A new rank-based version of the Ant System: A computational study. Cen. Eur. J. for Oper. Res. and Econ. 7(1), 25–38 (1999)
6. Conover, W.J.: Practical Nonparametric Statistics. John Wiley & Sons, New York, NY, third edn. (1999)
7. Doerner, K.F., Hartl, R.F., Reimann, M.: Are COMPETants more competent for problem solving? The case of a multiple objective transportation problem. Cen. Eur. J. for Oper. Res. and Econ. 11(2), 115–141 (2003)
8. Fonseca, C.M., Paquete, L., López-Ibáñez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: CEC 2006, pp. 1157–1163. IEEE Press, Piscataway, NJ (Jul 2006)
9. García-Martínez, C., Cordón, O., Herrera, F.: A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. Eur. J. of Oper. Res. 180(1), 116–148 (2007)
10. Iredi, S., Merkle, D., Middendorf, M.: Bi-criterion optimization with multi colony ant algorithms. In: Zitzler, E., et al. (eds.) EMO 2001, LNCS, vol. 1993, pp. 359–372. Springer (2001)
11. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
12. López-Ibáñez, M., Stützle, T.: The impact of design choices of multi-objective ant colony optimization algorithms on performance: An experimental study on the biobjective TSP. In: Pelikan, M., Branke, J. (eds.) GECCO 2010, pp. 71–78. ACM press, New York, NY (2010)
13. López-Ibáñez, M., Stützle, T.: The automatic design of multi-objective ant colony optimization algorithms. IEEE Trans. on Evol. Comput. (2012), in press
14. Lust, T., Teghem, J.: The multiobjective multidimensional knapsack problem: a survey and a new approach. Arxiv preprint arXiv:1007.4063 (2010)
15. Stützle, T., Hoos, H.H.: $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. Future Generat. Comput. Systems 16(8), 889–914 (2000)
16. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto evolutionary algorithm. IEEE Trans. on Evol. Comput. 3(4), 257–271 (1999)
17. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Grunert da Fonseca, V.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. on Evol. Comput. 7(2), 117–132 (2003)