Analyzing the impact of MOACO components: An algorithmic study on the multi-objective shortest path problem

Leonardo C.T. Bezerra a, Elizabeth F.G. Goldbarg a,⁎, Marco C. Goldbarg a, Luciana S. Buriol b

aUniversidade Federal do Rio Grande do Norte, Natal, RN, Brazil
bUniversidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brazil

1. Introduction

Multi-objective Ant Colony Optimization (MOACO) algorithms have been successfully applied to several multi-objective combinatorial optimization problems (MCOP) over the past decade. Recently, we proposed a MOACO algorithm named GRACE for the multi-objective shortest path (MSP) problem, confirming the efficiency of such metaheuristic for this MCOP. In this paper, we investigate several extensions of GRACE, proposing several single and multi-colony variants of the original algorithm. All variants are compared on the original set of instances used for proposing GRACE. The best-performing variants are also assessed using a new benchmark containing 300 larger instances with three different underlying graph structures. Experimental evaluation shows one of the variants to produce better results than the others, including the original GRACE, thus improving the state-of-the-art of MSP.

© 2012 Elsevier Ltd. All rights reserved.
2. The multi-objective shortest path problem

The multi-objective shortest path problem studied in this paper is a generalization of the classical point-to-point shortest path problem, and is presented by Raith and Ehrgott using a network flow formulation for two objectives (Raith & Ehrgott, 2009). The expanded version of this formulation to deal with any number of objectives is used in this work:

\[
\begin{align*}
\text{min } z(x) &= \{ \sum_{(j_i,j_j) \in A} c_{ij} x_{ij} \\
& \text{subject to: } \sum_{(j_j,i) \in A} x_{ij} - \sum_{(i,j_i) \in A} x_{ij} &= \begin{cases} 
1 & \text{if } i = s, \\
0 & \text{if } i \neq s, t, \\
-1 & \text{if } i = t,
\end{cases} \\
x_{ij} &\in \{0,1\}, \quad \forall (i,j) \in A,
\end{align*}
\]

where \(s\) and \(t\) are, respectively, source and terminal nodes, \(c\) is a \(k\)-dimensional cost matrix for each edge \((i,j)\), and \(z\) is the objective vector composed by \(k\) objective functions.

The MSP belongs to a class of problems that have polynomial algorithms for their original single objective versions. This allows optimizers to use a two-phase strategy that was shown to perform efficiently on the bi-objective context (Paquete & Stützle, 2003; Raith & Ehrgott, 2009). This approach takes advantage of the fact that some of the solutions contained in the Pareto set can be retrieved with the help of scalarizations, i.e., by attributing weights to each objective. This means that part of the Pareto set can be retrieved polynomially. Having these solutions (or part of them) \textit{a priori}, an optimizer can narrow the search space when looking for non-supported solutions. The literature on this problem includes exact and heuristic algorithms, which include evolutionary and ant colony optimizers. Raith and Ehrgott, 2009 recently compared different exact proposals (Martins, 1994; Mote, Murphy, & Olson, 1991; Skrifer & Andersen, 2000) for the Bi-objective Shortest Path (BSP) problem, and showed that the two-phase approach is able to solve instances up to 21000 nodes in less than 30 s, and up to 300000 nodes in less than 40 s. This algorithm, however, has not been expanded to handle problems with more than two objectives (this generalization is not trivial).

Given the excellent results of exact algorithms for the BSP, our review of the MSP literature does not include algorithms proposed for dealing with two objectives. Among the EMO algorithms proposed for the MSP (He, Qi, & Fan, 2007; Mooney & Winstansley, 2006; Pangilinan & Janseens, 2007), some commonalities can be identified. First, the elitist approach can be identified in all, with each algorithm keeping two populations during its entire execution. Second, selection by binary tournament is used by all algorithms. Finally, crossover operators differ on the method to exchange genes, but all use a one-point approach. Important different features are listed in Table 1. We refer to the traditional exchange of genes used in evolutionary algorithms crossover operators as \textit{direct exchange}. The paired exchange used by He et al. (2007) and Pangilinan and Janseens (2007) is a four-stage procedure: (i) a locus \(i\) is randomly chosen in one of the parents \(p_1\); (ii) the operator searches the other parent \(p_2\) to find out if the same gene is present; (iii) if the gene is found in locus \(j\) of \(p_2\), a direct exchange is performed considering positions \(i\) and \(j\) as starting points for \(p_1\) and \(p_2\) respectively, and finally; (iv) if a cycle is present in any of the resulting offsprings, a repair function is called to remove it by discarding all elements of the cycle. Concerning mutation operators, the algorithm proposed by Mooney and Winstansley (2006) randomly chooses one locus of the chromosome, and replaces its gene with a randomly chosen node. This operation is only considered to succeed if a feasible path is generated. By their turn, He et al. (2007) and Pangilinan and Janseens (2007) reconstruct part of the chromosome, starting at a randomly chosen locus. The methods applied for the reconstruction of the chromosomes can be seen in Table 1. Since no common experimental setup is used by the authors, the actual performance of the algorithms remains unclear.

To the best of our knowledge, only one MOACO algorithm (other than GRACE) has been proposed for the MSP (Häckel et al., 2008), and will be detailed in the next section.

3. Ant Colony Optimization

Ant Colony Optimization (ACO) is a bio-inspired metaheuristic that uses the concept of \textit{swarm intelligence}, i.e., the ability of groups to communicate even in the absence of a central coordination, through a stimulus that is at the same time physical and local. This phenomenon is called \textit{stigmergy} (Grassé, 1959), and in the case of ants, this stimulus is called \textit{pheromone} (Dorigo & Socha, 2006). When looking for food, ants are able to identify and perform pheromone deposits over the paths they tread in order to guide other ants to follow their trails. Since pheromone evaporates with time, shorter paths are more likely to be reinforced than longer ones, promoting convergence towards promising solutions.

In an attempt to mimic this natural behavior, the Ant System (AS) metaheuristic was proposed (Dorigo, 1992), later improved into the Ant Colony System (ACS) (Dorigo & Gambardella, 1997) and the MAX – MIN Ant System (MMAS) (Stützle & Hoos, 2000). In all algorithms, solutions are iteratively built by agents called ants. Each ant constructs a solution and is able to evaluate its current state, i.e., to calculate the objective value of its current path. While building its solution, an ant makes decisions at each state based on the information available for the possible choices. Such information comes from the experience of other ants, by means of the amount of pheromone deposited on a given path, and from its own experience, by means of a heuristic. The probability that an ant makes the transition from state \(i\) to state \(j\), \(p_i(e_j)\), is given in Eq. (4) where \(N_j\) denotes the set of states that are reachable from \(i\), \(\pi_j\) denotes the amount of pheromone between states \(i\) and \(j\), \(\eta_j\) the heuristic value associated to the transition to state \(j\), and \(\alpha\) and \(\beta\) are parameters that weight the importance of the information that comes from the pheromone and the heuristic, respectively.
As soon as an ant finishes constructing its solution, and; (ii) in
update: (i) in ACS, a local pheromone deposit may be performed,
ences between AS, ACS and MMAS refer to pheromone limits and
formed, which naturally
approach is to
(4)

\[ p(e_i) = \begin{cases} 
\frac{\tau_{ij}^a}{\sum_{k \in N_i} \tau_{ik}^a} & \text{if } j \in N_i, \\
0 & \text{otherwise.} 
\end{cases} \]

At the end of each iteration, pheromone deposits may be per-
formed, which naturally evaporates over time. The major differ-
ences between AS, ACS and MMAS refer to pheromone limits and
and; (ii) in MMAS, the pheromone information is subject to limits, \( \tau_{\text{max}} \) and
Moreover, both MMAS and ACS use a pseudo-random proportional rule, i.e., ants may choose the best transition available instead of using stochasticity. In this paper, all algorithms implemented are based on the ACS, but the original transition rule from AS is adopted.

3.1. Multi-objective ACOs

Several proposals can be found on the literature of ACO algo-
rithms for multi-objective problems. These MOACO algorithms show significant differences regarding important design questions. Since a large number of proposals have been made, we review them according to the following three topics:

3.1.1. Solution construction

Since the nature of the objectives for a given problem may be het-
erogeneous, some algorithms choose to use objective-specific heu-
ristics (Alaya, Solnon, & Ghédira, 2007; Barán & Schaerer, 2003; Doerner et al., 2001, 2006; Doerner, Gutjahr, Hartl, Strauss, & Stummer, 2004; Ghoseiri & Nadjari, 2009; Häckel et al., 2008; Mora et al., 2009) and/or pheromone structures (Alaya et al., 2007; Barán & Schaerer, 2003; Doerner et al., 2001; Ghoseiri & Nadjari, 2009; Häckel et al., 2008; Iredi et al., 2001; Ke et al., 2010; Mora et al., 2009). These different sources of information need to be aggregated, tradi-
tionally by means of scalarization vectors. The commonly employed aggregation methods over the literature include weighted sum (Doerner et al., 2001, 2004, 2006; Mora et al., 2009), weighted product (Alaya et al., 2007; Ghoseiri & Nadjari, 2009; Iredi et al., 2001) and non-weighted sum (Alaya et al., 2007). Another possible aggregation approach is to randomly choose which objective to optimize at each iteration or step of the constructive procedure (Alaya et al., 2007). Furthermore, algorithms differ as to use a single scalarization vector per iteration (López-Liébana & Stützle, 2010), or many scalarization vectors simultaneously (Chica et al., 2011; Doerner et al., 2004, 2006; Iredi et al., 2001).

3.1.2. Number of colonies

The most straightforward generalizations of ACOs to deal with
multi-objective problems maintain traditional single colony ap-
proaches (Alaya et al., 2007; Barán & Schaerer, 2003; Doerner et al., 2004, 2006; Ghoseiri & Nadjari, 2009; Ke et al., 2010; Mora et al., 2009). Others, more elaborated, choose to use more than one colony in an attempt to specialize in different regions of the Pareto front. Theoretically, such specialization requires: (i) having (at least) one pheromone structure per colony; (ii) using only scala-
rization vectors that represent the region for which the colony is responsible; and; (iii) reinforcing the pheromone structures only with solutions that belong to the region for which the colony is responsible. Over the literature, condition (i) is not always used, since some algorithms use different colonies with common phero-
alone structures (Alaya et al., 2007; Häckel et al., 2008; Ke et al., 2010). Concerning the scalarization vectors used by each colony, colonies may share scalarization vectors (Iredi et al., 2001) or use complete disjoint sets of vectors (Chica et al., 2011; Iredi et al., 2001). Finally, concerning the regionalized selection for pher-
omone update, this approach has only been proposed so far for bi-
criteria problems (Iredi et al., 2001). The other method used over
the literature is the traditional update by origin, that is, if solution i was found by colony j, it will be used to update the pheromone structure (s) of j.

3.1.3. Selection criteria for pheromone update

The two major selection criteria for pheromone update cur-
cently found on the literature allow either the best-per-objective or the nondominated solutions to deposit pheromone. In the first case, a number of best-per-objective ants (Alaya et al., 2007; Doerner et al., 2001, 2004, 2006) is allowed to update. In the second, only ants that found nondominated solutions (Alaya et al., 2007; Barán & Schaerer, 2003; Ghoseiri & Nadjari, 2009; Häckel et al., 2008; Iredi et al., 2001; Ke et al., 2010; Mora et al., 2009) can perform deposits. In both cases, the reference used for the selection of best-per-objective or nondominated solutions vary according to: (i) colonies, which means solutions can be compared with a global (Ghoseiri & Nadjari, 2009; Iredi et al., 2001; Ke et al., 2010; Mora et al., 2009) or a local (Alaya et al., 2007; Doerner et al., 2004, 2006; Häckel et al., 2008; Iredi et al., 2001) pool of solutions, and; (ii) history, that is, solutions can be compared only with the solutions found in a given iteration (Doerner et al., 2004, 2006; Ghoseiri & Nadjari, 2009; Iredi et al., 2001) or to a history of best-so-far solutions (Häckel et al., 2008; Iredi et al., 2001; Ke et al., 2010; Mora et al., 2009).

3.2. MOACOs for the MSP

To the best of our knowledge, only two MOACO algorithms have been proposed so far for the MSP (Ghoseiri & Nadjari, 2009; Häckel et al., 2008). However, since the MOACO proposed by Ghoseiri and Nadjari (2009) addresses the particular case of \( k = 2 \) (BSP), we will limit ourselves to the description of the algorithm proposed by Häckel et al. (2008). The MOACO algorithm proposed by Häckel et al. (2008) uses a multi-colony approach, with disjoint intervals of weights. A single global pheromone matrix is adopted. Selection by dominance is used, as well as the update by origin method. The heuristic information comes from a Dynamic Programming algo-
rithm called Look-Ahead Heuristic (LAH), proposed by the authors. The experiments conducted in their work show that using LAH improves overall results, and that their ACO obtains solutions well distributed along the tri-dimensional objective space in compar-
isation to a dynamic multi-objective algorithm not referenced by the authors. However, in a previous work we have compared our implementation of this algorithm with GRACE and the latter out-
performed the first under the experimental setup reproduced here.

4. Extending GRACE

Many are the possibilities when designing a MOACO algorithm. In this section, several extensions are proposed and tested for
GRACE (Bezerra et al., 2011). The original algorithm is described in Section 4.1. The proposed variants are presented in Section 4.2.

4.1. GRACE

Devised as a two-phase algorithm, GRACE initially uses a search procedure, named Logos, to retrieve supported efficient solutions. This search strategy can be used for bi and tri-criteria scenarios (2 and 3-Logos, respectively). 2-Logos (see Algorithm 2-Logos below) iteratively divides each search region into 2 sub-regions, resembling a logarithmic function. The general framework of 2-Logos is presented below. In the first execution of 2-Logos two solutions \( s_1 \) and \( s_2 \) are obtained by solving the single-objective minimum shortest path for each objective separately. Each solution corresponds to solving an scalarized version of the MSP problem with weights (1,0) and (0,1). More generally, solutions \( s_1 \) and \( s_2 \) correspond respectively to points \((x_1,y_1)\) and \((x_2,y_2)\) in the objective space. A scalarization vector corresponding to \((x_{midpoint},y_{midpoint})\) is generated, and the single-objective minimum shortest path using this ponderation is retrieved, returning a solution \( s_{mid} \). If this solution is non-dominated regarding the global archive, the procedure is recursively called for \((s_1,s_{mid})\) and \((s_{mid},s_2)\). This strategy is similar to dichotomic approaches (Raith & Ehrgott, 2009; Steiner & Radzik, 2003).

Analogously, 3-Logos (see Algorithm 3-Logos below) initially finds the extreme supported efficient solutions using Dijkstra’s algorithm (Dijkstra, 1959), that is, the solutions for the MSP considering each objective separately. These three solutions are considered vertices of a triangle where each edge is formed by the line that connects each pair of input weights. The edges are initially scanned by 2-Logos. Then, a scalarization vector \( \lambda_{ctr} \) corresponding to the centroid of the triangle comprised by the input weights is calculated, in a similar fashion to the procedure adopted for 2-Logos. At this stage, Dijkstra’s algorithm is run on the scalarized MSP considering \( \lambda_{ctr} \). In case the obtained solution is new in the set \( S \) of non-dominated solutions, 3-Logos is recursively called for three new sub-triangles, as shown in Fig. 1. The pseudocode of 3-Logos can be seen below.

**Algorithm**: 2-Logos (extreme solutions \( s_1 \) and \( s_2 \))

- generate new scalarization vector: \( \lambda_{mid} = \frac{\lambda_1 + \lambda_2}{2} \)
- find \( s_{mid} \) using Dijkstra and \( \lambda_{mid} \)
- if \( s_{mid} \neq s_1 \) and \( s_{mid} \neq s_2 \) and \( s_{mid} \) is non-dominated w.r.t global archive
  - add \( s_{mid} \) to the global archive
  - call 2-Logos for \((s_1,s_{mid})\)
  - call 2-Logos for \((s_{mid},s_2)\)

**Algorithm**: 3-Logos (extreme solutions \( s_1, s_2, s_3 \))

- call 2-Logos for \((s_1,s_2)\)
- call 2-Logos for \((s_1,s_3)\)
- call 2-Logos for \((s_2,s_3)\)
- generate new scalarization vector: \( \lambda_{ctr} = \frac{\lambda_1 + \lambda_2 + \lambda_3}{3} \)
- find \( s_{ctr} \) using Dijkstra and \( \lambda_{ctr} \)
- if \( s_{ctr} \neq s_1 \neq s_2 \neq s_3 \) and \( s_{ctr} \) is non-dominated w.r.t global archive
  - add \( s_{ctr} \) to the global archive
- call 3-Logos for \((s_1,s_2,s_{ctr})\)
- call 3-Logos for \((s_1,s_{ctr},s_3)\)
- call 3-Logos for \((s_2,s_{ctr},s_3)\)

The pseudocode of the second phase of GRACE can be seen below. A single colony approach is adopted. The solutions found in the first phase are used to warm up the single pheromone matrix: every time an edge is found in a supported efficient solution, a deposit is made. Dijkstra’s algorithm provides heuristic information to the ants: a scalarization vector is used to obtain a single-objective problem, and Dijkstra’s algorithm is applied on an inverted graph, which is obtained from the original graph with the inversion of the direction of all edges. In other words, the heuristic information available in each node is the distance from it to the terminal node, under a given scalarization. These scalarization vectors are randomly generated at the beginning of each iteration, and all ants use the same vector per iteration. New non-dominated solutions found by the ants are added to an external archive, which is unlimited. The ants are only allowed to perform pheromone updates if they find new non-dominated solutions. Each of the ants allowed to update the pheromone matrix deposits an equal amount of pheromone, regardless of the objective values of the solution found. The main loop comprises a group of \( R \) iterations each called cycle. If during a cycle a new non-dominated solution arises, then a new cycle of iterations is allowed and the algorithm receives more processing time.

**Algorithm**: GRACE (graph \( G \), nodes \( s \) and \( t \))

- find initial solutions using Logos and Dijkstra
- warmup pheromone structure using initial solutions
- repeat
  - generate a random scalarization vector
  - calculate heuristic information using Dijkstra
  - for each ant \( a \) in \( 1..\#Ants \)
    - construct a solution \( sol \)
    - if \( sol \) is new and non-dominated
      - add \( sol \) to the global archive
      - update pheromone
  - until a cycle cannot find a new non-dominated solution
4.2. Proposed extensions

Extensions of this algorithm are presented in the following subsections, with single and multiple-colony approaches. When the concept behind the version does not allow the single colony approach, it is explicitly stated. In the multi-colony approaches, different colonies do not interact directly: each of them has its own pheromone matrix and heuristic information. A single type of heuristic information and pheromone is used for all colonies. However, when selecting ants to perform pheromone update, all colonies test their solutions according to dominance in regard to the external global archive that stores the solutions found by all colonies during the whole execution of the algorithm. Regarding the update by region strategy, an important question arises: how to order solutions to determine regions when this update strategy is used on problems with three objectives. Whereas a simple lexicographic ordering divides the objective space “equally” among objectives in the bi-objective case, there is no trivial solution for the three objective context. The strategies used to overcome this problem are described later.

Five variants of the basic algorithm are proposed differing from each other basically on scalarization vector used to provide heuristic information to the ants. These versions are named: Global-Random, Pseudo-Global-Random, Fixed-Unique, Fixed-Multi, and Objective-Driven. Moreover, two hybrid versions named Mixed and Mixed Objective-Driven are also proposed.

4.2.1. Global-Random

The first variant, called GRan, uses randomly generated scalarization vectors, as the original GRACE algorithm. However, multiple colonies may be used, in which case the update by origin method is adopted. The pseudocode of GRan can be seen below:

**Algorithm:** Variant GRan

```plaintext
for each colony c in 1..#Colonies
    randomly generate a scalarization vector \( \lambda \)
    for each ant a in 1..#Ants
        construct a solution \( sol \) using \( \lambda \)
        if \( sol \) is a new non-dominated solution
            update pheromone structure of colony c
```

4.2.2. Pseudo-Global-Random

The second variant, called PGRan, generates random scalarization vector within triangular bounds. Initially, all possible scalarization vectors are generated, given an arbitrary number \( d \) of divisions. For example, for \( k = 3 \) and \( d = 2 \), six vectors are created: \([1,0,0], [0,5,0,5,0], [0,5,0,5,0], [0,5,0,5,0], [0,1,0], \) and \([0,0,1]\). These vectors comprise triangles on the objective space, as depicted in Fig. 2. In this variant, these triangular regions are considered colonies, and at each iteration a random scalarization vector bounded by the colonies’ triangles are generated. The pseudocode of PGRan can be seen below. When multiple colonies are used, the update by origin method is adopted.

**Algorithm:** Variant PGRan

```plaintext
generate set of vectors systematically
for each colony c in 1..#Colonies
    generate random vector \( \lambda \) bounded by colony limits
    for each ant a in 1..#Ants
        construct a solution \( sol \) using \( \lambda \)
        if \( sol \) is a new non-dominated solution
            update pheromone structure of colony c
```

4.2.3. Fixed-Unique

The third variant, called FUn, generates the scalarization vectors systematically given an arbitrary number \( d \) of divisions, as in the generation of bounds for PGRan. In this variant, each colony is assigned a single scalarization vector, and the update by origin method is used. This variant cannot be implemented with only one colony, because when \( d = 1 \) the number of colonies equals the number of objectives, which is always greater than one. The pseudocode of FUn can be seen below, where \( \lambda_k \) is the \( k \)th scalarization vector.

**Algorithm:** Variant FUn

```plaintext
for each colony c in 1..#Colonies
    for each ant a in 1..#Ants
        construct solution \( sol \) using \( \lambda_c \)
        if \( sol \) is a new non-dominated solution
            update pheromone structure of colony c
```

4.2.4. Fixed-Multiple

The fourth variant, called FMu, also generates the scalarization vectors systematically given an arbitrary number \( d \) of divisions, as in FUn. In this variant, to each colony multiple scalarization vectors are assigned. The distribution of vectors is performed in the order they were generated. Hence, for \( m = 3 \), \( k = 3 \), and \( d = 2 \), colony 1 is assigned vectors \([1,0,0]\) and \([0.5,0.5,0]\), colony 2 is assigned vectors \([0.5,0.5,0]\) and \([0.5,0.5,0]\), and colony 3 is assigned vectors \([0,1,0]\) and \([0,0,1]\). If the number of vectors is not divisible by the number of colonies, the last colony receives the extra vectors. This variant can be implemented using single or multiple colonies. When multiple colonies are used, the update by origin method is adopted. The pseudocode of FMu can be seen below, where \( \lambda_{iv} \) is the \( iv \)th vector assigned to colony c.

**Algorithm:** Variant FMu

```plaintext
for each colony c in 1..#Colonies
    sequentially assign equal amount of vectors to colonies
    for each vector v in 1..#Vectors_per_colony
        for each ant a in 1..#Ants
            construct solution \( sol \) using \( \lambda_{iv} \)
            if \( sol \) is a new non-dominated solution
                update pheromone structure of colony c
```

4.2.5. Mixed

The fifth variant, named Mix, is identical to FMu, except for the inclusion of a random scalarization vector in each colony. Single or multiple colonies may be used. When multiple colonies are
adopted, the update by origin is used. The pseudocode of Mix can be seen below, where $x^c_i$ is the $i$th vector assigned to colony $c$.

**Algorithm: Variant Mix**

generate set of vectors systematically sequentially assign equal amount of vectors to colonies

for each colony $c$ in 1..#Colonies
  for each vector $v$ in 1..#Vectors
    construct solution $sol$ using $x^c_v$
    if $sol$ is a new non-dominated solution
      update pheromone structure of colony $c$
    randomly generate a scalarization vector $\lambda_{random}$
    for each ant $a$ in 1..#Ants
      construct a solution $sol$ using $\lambda_{random}$
      if $sol$ is a new non-dominated solution
        update pheromone structure of colony $c$

4.2.6. Objective-Driven

The sixth variant, named Obj, has colonies specialized on each objective. The scalarization vectors are generated as in FUn. For an instance with $k$ objectives, a given colony $i$, $i = 1, \ldots, \#Colonies$, is assigned all vectors that priorize criterion $1 + (i - 1)k$ (one plus the remainder of the integer division of $i - 1$ by $k$). For example, a vector $[0.5, 0.5, 0]$ would be rejected by colony 1, for the weight associated with the first objective is not greater than 0.5. Since there must be at least one colony per objective, a single colony version cannot be created. However, there is no limitation in how many colonies there might be per objective. Throughout this paper, only one colony per objective is used. Each colony has its own pheromone matrix, that specializes for its primary objective.

For the pheromone update, three strategies were tested: *update by origin*, *update by region* and using both strategies at the same time. In the case of the update by region strategy it is necessary to determine the region to which a new generated solution belongs. Given a new solution $s_{new}$ and an external archive $A$, the policy adopted states that: (i) if a solution $s \in A$ is dominated by the new solution $s_{new}$, then the pheromone matrices of the colonies that focus on the objectives that have been optimized are updated, or; (ii) if $s_{new}$ does not dominate any solution of $A$, it is compared with the last solution found. For every objective $i \in \{1, \ldots, k\}$, if $f_i(s_{new}) < f_i(s_{last})$, a pheromone update is performed on the matrix of the colony that focuses that objective.

When update by origin and update by region are used simultaneously, the strategy applied to determine which regions a solution belongs to is used first. The region corresponding to the colony the ant that generated that solution belongs to is always updated. The pseudocode for Obj is shown below.

**Algorithm: Variant Obj**

generate set of vectors systematically sequentially assign equal amount of vectors to colonies

for each colony $c$ in 1..#Colonies
  for each vector $v$ in 1..#Vectors
    if component $(1 + (c - 1)k) \cdot \lambda_v > 0.5$
      for each ant $a$ in 1..#Ants
        construct solution $sol$ using $\lambda_v$
        if $sol$ is a new non-dominated solution
          determine regions of $sol$
        if update by origin then update colony $c$
        if update by region then update regions

4.3. Mixed Objective-Driven

The seventh variant, called OMix, is basically Obj with the inclusion of a random scalarization vector (similar to Mix and FMu). The pseudocode for OMix is shown below.

**Algorithm: Variant OMix**

generate set of vectors systematically sequentially assign equal amount of vectors to colonies

for each colony $c$ in 1..#Colonies
  for each vector $v$ in 1..#Vectors
    if component $(1 + (c - 1)k) \cdot \lambda_v > 0.5$
      for each ant $a$ in 1..#Ants
        construct solution $sol$ using $\lambda_v$
        if $sol$ is a new non-dominated solution
          determine regions of $sol$
        if update by origin then update colony $c$
        if update by region then update regions

5. Experiments with the original instance set

In this section, we perform a comparison among all the proposed variants using the original instance set used by Bezerra et al. (2011). First, Section 5.1 presents the experimental setup used, namely the set of instances, performance assessment methodology and platform description. The parameter calibration performed for each variant is described in Section 5.2. Finally, results are presented and discussed in Section 5.3.

5.1. Methodology, platform and instances

The assessment of the results produced in the computational experiments reported in this paper is done based on the methodology presented by Knowles et al. (2006). At first, the approximation sets delivered by the tested algorithms are compared by means of dominance rankings. Given a list of $q$ optimizers, $r_i$ ($i = 1, \ldots, q$) independent executions for each of them and a collection $C$ containing all approximation sets $\mathcal{C}_j$ ($j = 1, \ldots, q$), each optimizer is assigned a rank, $\text{rank}(\mathcal{C}_j)$, equal to one plus the number of sets that are better than it. To state the concept of better between two approximation sets, some definitions are necessary. Given two objective vectors $z_1$ and $z_2$, $z_1$ is said to weakly dominate $z_2$ if $z_1$ is not worse than $z_2$ in all objectives (Zitzler et al., 2003). This concept is extended to approximation sets by the set relation better. Let $A_1$ and $A_2$ be two approximation sets. $A_1$ is said to be better than $A_2$ when every objective vector $z_2 \in A_2$ is weakly dominated by at least one $z_1 \in A_1$ and the approximation sets differ from each other, at least, in one objective vector. To verify if an approximation set is better than another, we used the binary additive epsilon indicator (Zitzler et al., 2003). Once all comparisons are performed, ranks are calculated and every optimizer is characterized by a sample $(\text{rank}(\mathcal{C}_1), \ldots, \text{rank}(\mathcal{C}_q))$. These samples can be compared using statistical tests. Knowles et al. (2006) claim that, if an optimizer $Q^j$ has dominance rankings statistically lower than another...
optimizer $Q^2$, then it can be said that $Q^2$ generates better approximation sets than $Q^2$, and no further testing is required.

In case no statistical difference is observed with the data produced with the dominance ranking method, quality indicators are used. Quality indicators are functions that attribute qualitative values to approximation sets. A unary quality indicator $I$ is defined as a mapping from the set of all approximation sets to the set of real numbers. Some quality indicators use a reference set to compare the approximation sets generated by an optimizer. This reference set can be generated by merging the approximation sets of all optimizers being tested (Knowles et al., 2006). Each indicator measures a specific characteristic of the set and, therefore, combining multiple indicators is suitable. In this work, two of these indicators are used:

1. the hypervolume unary indicator ($I_h$) (Zitzler & Thiele, 1999), which calculates the hypervolume of the objective space that is weakly dominated by the approximation set being tested (limited by a reference point);
2. the additive binary-$c$ indicator ($I_{bc}$) (Zitzler et al., 2003), which calculates the minimum $c$ which must be added to each solution in a set $A_s$ for it to become weakly dominated by another set $A_t$.

To compare more than two optimizers, we initially use the Kruskal and Wallis (1952) test, and if a statistically significant difference is found at a significance level of 95%, new samples/values are generated for a pairwise comparison. At this stage, two tests are applied: the Wilcoxon (1945) one-tailed test at 97.5% of significance level. If only two optimizers are being compared at a time, there is no need to use Kruskal–Wallis test, and the Wilcoxon test is directly applied. The results returned by the Wilcoxon test are, finally, compared using Taillard, Waelti, and Zuber (2009) test for comparison of proportions with significance level of 95%. For brevity, the only results reported here are the p-values returned by Taillard et al. (2009) test, which are rounded to two decimal places.

All tests were executed on an Intel Xeon QuadCore W3520 2.8 GHz, with 8 GB of RAM running Scientific Linux 5.5 64 bits distribution. Experiments were executed with 18 instances generated by MSPP (2010), from two distinct classes: square, which represents a square grid, and complete, which contains complete graphs. Fixed processing times for each instance are adopted as stopping criteria. Experiments were executed with 18 instances generated for a pairwise comparison. At this stage, two tests were conducted using MSPP (2010), which are rounded to two decimal places.

5.2. Parameter calibration

For the assessment of all variants, the parameters of each were initially calibrated. Final parameter settings are listed on Table 3. For GRan, 1, 3 and 5 colonies were tested. The single colony version corresponds to the original GRACE. Ants only update the pheromone matrix of its own colony. Results showed the single colony version to outperform the others. A possible explanation for these results is that, since this multi-colony approach is equivalent to a multi-start strategy, using multiple colonies only increases the computational effort.

For GRan and FUn variants, results showed these variants were competitive or did not improve the current results obtained by GRACE. For GRan, tests were conducted with $d \in \{1,2,3,4,5\}$, which correspond to 1, 4, 9, 16 and 25 colonies, respectively. The results of the pairwise comparisons showed that the single colony version, which corresponds to the single colony GRan, performs better than all other versions. For the FUn version, values 1, 2 and 3 were tested for parameter $d$ which correspond to 3, 6 and 10 scalarization vectors and colonies. Again, only update by origin is used. No statistically significant differences were found on the complete instances, but for square instances results show that the smaller the value of $d$, the better the performance of this algorithmic version. It was also noticed that this variant was never able to outperform the original algorithm. For such reasons, both GRan and FUn were not considered for further experiments.

For the FMu version, tests were conducted using $m \in \{1,3,5\}$ and $d \in \{3,6,10\}$, which correspond to a total of 10, 28 and 66 scalarization vectors, respectively. At first, versions with $m = 1$ were tested for the different values of parameter $d$. Significant different results were not found either with dominance ranking or the unary quality indicators among those versions. The same occurred for $m = 3$ and 5. This means the $m$ parameter could much likely be tested with any value for $d$, and results are expected to be similar. Therefore, fixing $d = 3$, comparisons show the same approximation sets have been generated for complete instances, regardless of the configuration used. However, for any number of divisions, the smaller the amount of colonies used, the better the overall performance of the algorithm on square instances. This result is most likely explained by the way weights are assigned to colonies: an assignment according to the order weights are generated does not help colonies specialize in any given region. We also observed that $d = 1$ generates results comparable to $d = 3$. This result is

multi-start strategy, using multiple colonies only increases the computational effort.

For GRan and FUn variants, results showed these variants were competitive or did not improve the current results obtained by GRACE. For GRan, tests were conducted with $d \in \{1,2,3,4,5\}$, which correspond to 1, 4, 9, 16 and 25 colonies, respectively. The results of the pairwise comparisons showed that the single colony version, which corresponds to the single colony GRan, performs better than all other versions. For the FUn version, values 1, 2 and 3 were tested for parameter $d$ which correspond to 3, 6 and 10 scalarization vectors and colonies. Again, only update by origin is used. No statistically significant differences were found on the complete instances, but for square instances results show that the smaller the value of $d$, the better the performance of this algorithmic version. It was also noticed that this variant was never able to outperform the original algorithm. For such reasons, both GRan and FUn were not considered for further experiments.

For the FMu version, tests were conducted using $m \in \{1,3,5\}$ and $d \in \{3,6,10\}$, which correspond to a total of 10, 28 and 66 scalarization vectors, respectively. At first, versions with $m = 1$ were tested for the different values of parameter $d$. Significant different results were not found either with dominance ranking or the unary quality indicators among those versions. The same occurred for $m = 3$ and 5. This means the $m$ parameter could much likely be tested with any value for $d$, and results are expected to be similar. Therefore, fixing $d = 3$, comparisons show the same approximation sets have been generated for complete instances, regardless of the configuration used. However, for any number of divisions, the smaller the amount of colonies used, the better the overall performance of the algorithm on square instances. This result is most likely explained by the way weights are assigned to colonies: an assignment according to the order weights are generated does not help colonies specialize in any given region. We also observed that $d = 1$ generates results comparable to $d = 3$. This result is
interesting, because for \( d = 1 \) FMu becomes a configuration very similar to FUn, both using only the three canonical weights: \((0,0,1)\), \((0,1,0)\) and \((1,0,0)\). The difference between their performance lies in the fact that FUn uses one colony for each objective, while FMu with \( d = 1 \) uses a single colony, thus promoting cooperation in the algorithm.

For the Mix variant, results of the previous experiments are used to reduce the number of possible values to be tested for tuning parameters \( m \) and \( d \). The values tested for \( m \) are 1 and 3, and for \( d \) are 3, 6 and 10. The results of the statistical tests concerning pairwise comparisons confirm that the single-colony version prevails on both instance sets \((p\text{-value} < 0.01)\). The explanation for these results is most likely the same discussed for FMu.

For the tuning of Obj, experiments were conducted initially to set parameter \( d \) with testing values 3, 4 and 5. Considering any of the pheromone update methods used in this work (update by origin, update by region and using both strategies at the same time), no significant differences were observed. In the second part of the test, the three pheromone update strategies were directly compared to understand their individual contribution to the performance of the algorithm. Results show that the update by origin strategy is the one that performs worst among the tested square instances. No significant differences were found between the performances of the algorithm that updates by region and the one that uses both strategies simultaneously. This result indicates that the efficiency of the algorithm is directly related to the update by region method.

The same values for parameter \( d \) were tested for OMix versions. No significant difference was found for the square instances, regardless of the pheromone update strategy used. For complete instances, however, two different scenarios were found. For all methods, \( d = 3 \) produced approximation sets significantly better than the versions with \( d = 5 \). No significant differences were detected between versions with \( d = 3 \) and \( d = 4 \). The different update strategies were tested for \( d = 3 \). No significant difference was found for complete instances, but for squares the same behavior observed for Obj is repeated: the update by region strategy is critical to the efficiency of the algorithm concerning the quality of the approximation sets.

5.3. Results and discussion

The comparison between the variants using dominance rankings shows significant results for almost all pairs of optimizers. Table 4 displays the \( p\)-values for the pairwise proportions comparison using the test proposed by Taillard et al. (2009). To make the assessment more straightforward, all values inferior to 0.05 are highlighted in boldface. If the \( p\)-value displayed at position \((i,j)\) is lower than 0.05, then variant \( i \) is considered to produce better approximation sets than variant \( j \), either according to dominance rankings (column DR) or both dominance rankings and unary quality indicators (column DR + UQI). Hence, if a line \( i \) contains no \( p\)-values inferior to 0.05, variant \( i \) cannot be considered to perform better than any other variant. Also, if any \( p\)-value on column \( i \) is lower than 0.05, some of the variants were considered to be superior to variant \( i \). Analogously, a high-performing variant can be identified when a given line \( i \) contains several \( p\)-values lower than 0.05, and the same does not happen on column \( i \).

The observed results are very different for each instance set. Considering only complete instances, Table 4 shows that the GRan variant, which is the original GRACE algorithm, and the Mixed variants are the optimizers that produce the approximation sets with lowest dominance rankings among all optimizers. No difference is found between GRan and Mixed, nor between FMu and Obj. When the quality indicators are assessed, FMu is said to produce approximation sets with lower values in comparison to Obj. For the set of square instances, however, dominance rankings are unable to find significant differences between GRan and Obj. GRan and OMix, FMu and Mix, Mix and Obj, Mix and OMix, and Obj and OMix. Nevertheless, column DR + UQI shows the quality indicators point differences for all such ties. These results point FMu and Obj, which had performed worst on complete instances, to perform best, and both GRan and Mix, which had previously performed best, to perform worst for this set of instances.

Three main conclusions can be drawn from the results of this set of experiments. First, fully stochastic methods such as GRan appear to perform better than systematic approaches like FMu and Obj on unstructured instances, which is the case of the complete graphs. The opposite scenario is observed for highly structured instances such as squares, favoring FMu and Obj. Second, both Mix and OMix variants are unable to maintain dual characteristics, with Mix becoming really similar to GRan, and OMix becoming similar to Obj. Finally, it is interesting to notice that for the MSP using a single colony like in FMu generates better results than using region-specific colonies, like in Obj.

6. Experiments with the novel benchmark

The first set of experiments conducted in this work showed the instance structure to be determinant to the variants performance. In this section, we report an experimentation performed to compare GRan and FMu, the best performing variants on each instance class. A larger benchmark was devised for this purpose, comprising three different graph structures: complete, random and (non-square) grids. Section 6.1 presents the methodology used for the creation, as well as the final instance list used. Section 6.2 describes

<table>
<thead>
<tr>
<th>Table 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p)-values from pairwise comparisons concerning single-colonies GRan, FMu, and Mix, and 3-colonies Obj and OMix with update by origin and by region simultaneously.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DR &amp; DR + UQI</td>
</tr>
<tr>
<td>Complete</td>
</tr>
<tr>
<td>GRan &amp; FMu &amp; Mix &amp; Obj &amp; OMix</td>
</tr>
<tr>
<td>GRan &amp; – &amp; 0 &amp; 0.09 &amp; 0 &amp; 0 &amp; – &amp; 0 &amp; 0.09 &amp; 0 &amp; 0</td>
</tr>
<tr>
<td>FMu &amp; 1 &amp; – &amp; 1 &amp; 0.5 &amp; 1 &amp; 1 &amp; – &amp; 1 &amp; 0.5 &amp; 1</td>
</tr>
<tr>
<td>Mix &amp; 0.91 &amp; 0 &amp; – &amp; 0 &amp; 0 &amp; 0.91 &amp; 0 &amp; – &amp; 0 &amp; 0</td>
</tr>
<tr>
<td>Obj &amp; 1 &amp; 0.5 &amp; 1 &amp; – &amp; 1 &amp; 1 &amp; 0.5 &amp; 1 &amp; – &amp; 1</td>
</tr>
<tr>
<td>OMix &amp; 1 &amp; 0 &amp; 1 &amp; 0 &amp; – &amp; 1 &amp; 0 &amp; 1 &amp; 0 &amp; –</td>
</tr>
<tr>
<td>Square</td>
</tr>
<tr>
<td>GRan &amp; – &amp; 0.5 &amp; 0.5 &amp; 0.5 &amp; 0.5 &amp; – &amp; 1 &amp; 0.5 &amp; 1 &amp; 1</td>
</tr>
<tr>
<td>FMu &amp; 0.5 &amp; – &amp; 0.5 &amp; 0.5 &amp; 0.5 &amp; 0 &amp; – &amp; 0 &amp; 0.04 &amp; 0</td>
</tr>
<tr>
<td>Mix &amp; 0.5 &amp; 0.5 &amp; – &amp; 0.5 &amp; 0.5 &amp; 0.6 &amp; 1 &amp; – &amp; 1 &amp; 1</td>
</tr>
<tr>
<td>Obj &amp; 0.5 &amp; 0.5 &amp; 0.5 &amp; – &amp; 0.5 &amp; 0 &amp; 0.96 &amp; 0 &amp; – &amp; 0.09</td>
</tr>
<tr>
<td>OMix &amp; 0.5 &amp; 0.5 &amp; 0.5 &amp; 0.5 &amp; – &amp; 0 &amp; 1 &amp; 0 &amp; 0.91 &amp; –</td>
</tr>
</tbody>
</table>
For each configuration with size and computational time, the algorithm was able to solve most instances within five minutes, and all within one hour. For grid graphs, it was unable to solve the majority of the instances, having solved only 20\% of 16, 20\% of 64, and 60\% of 16 nodes instances.

The filtered benchmark contains 10 different configurations per class ranging from 1000 to 8000 nodes. For each configuration, 10 instances were generated with different seeds, totalizing 300 instances. Details can be seen on Table 6. The d column for random instances stands for the graph density. For the time limits in seconds, we used $min(t_{\text{exact}}, 300)$, where $t_{\text{exact}}$ is the time used by the exact algorithm for the instances it was able to solve.

### 6.2. Methodology and parameter calibration

For the calibration of the variants, a representative sample was chosen from the benchmark (one instance per configuration). Fifteen runs were performed by each optimizer for each parameter setting during the calibration stage. All parameters tested are listed on Table 7, and the final settings are highlighted in boldface. For the comparison of the variants, each algorithm performed three independent runs on the remaining 270 instances, totaling 810 runs per algorithm.

The assessment of the optimizers was conducted using the same methodology from the first set of experiments. This time, however, each algorithm was executed on a single core from an Intel Xeon QuadCore 2.66 GHz with 10 GB of RAM. Since only three executions were conducted per instance, we used the original Friedman test (Conover, 1999), considering the average between the three runs.

### 6.3. Results and discussion

The Friedman test results on the dominance rankings produced by both optimizers show FMu to generate approximation sets with statistically lower rankings than GRan when all instances are considered simultaneously. The same result holds when instance classes are considered separately, except for grid instances, where no significant difference among them is found. Further investigation was conducted on the quality indicators for this set, and again FMu produced sets with lower values for both indicators, at significance level 0.05. Tables 8 and 9 show the average values per configuration of each indicator.

For complete instances, FMu is able to generate better approximation sets than GRan for instances with up to 3000 nodes, and the opposite happens for larger instances. For grid instances, the $I^\rho_\gamma$ indicator does not let us draw conclusions on why GRan and FMu interchangeably produce good results, but the $I^\rho_m$ shows FMu to generate approximation sets with lower values than GRan for all instances but the largest two. For random instances, FMu generates better values than GRan for all instances but the smaller two according to both indicators. For those two instances the $I^\rho_m$.
indicators favor GRan, whether the $l_I$ indicator favors FMu, which means the approximation sets are most likely incomparable.

These results show that the different underlying graph structures become less important as instances grow larger. One possible explanation is that for a larger search space systematic approaches tend to degenerate less than stochastic methods, favoring FMu over GRan. Since this configuration of GRan is equal to the original GRACE algorithm, these results show the FMU variant proposed in this paper is able to generate approximation sets of better quality than the state-of-the-art of the problem just by changing the methods for the generation and assignment of scalarization vectors.

7. Conclusions and future work

This paper addressed the importance of studying the impact of different algorithmic components for MOACO algorithms when designing an optimizer for a particular MCOP. Particularly, a MOACO algorithm named GRACE from the state-of-the-art of the multi-objective shortest path was extended and improved. Moreover, different methods for the generation and assignment of scalarization vectors for MOACO algorithms were proposed and compared, which can possibly be extended to multi-objective metaheuristics in general. Finally, a new instance benchmark was proposed for the MSP, which may be used from now on to compare different optimizers proposed for this problem.

As future work possibilities, the methods for generating and assigning weights to colonies should be studied for different MCOPs, in order to test the generality of the results found in this work. Moreover, the same topic addressed in this paper for MOACO algorithms needs to be extended to other multi-objective metaheuristics, such as GRASP, Particle Swarm Optimization and EMO algorithms such as the MOEA/D (Zhang & Li, 2007).

References


