Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Parallelization of a non-linear multi-objective optimization algorithm: Application to a location problem ☆

Aránzazu Gila Arrondo [a,*], Juana L. Redondo [b], José Fernández [a], Pilar M. Ortigosa [c]

[a] Dpt. of Statistics and Operations Research, University of Murcia, Murcia, Spain
[b] Dpt. of Computer Architecture and Technology, Agrifood Campus of International Excellence, ceiA3, University of Granada, Granada, Spain
[c] Dpt. of Informatics, Agrifood Campus of International Excellence, ceiA3, University of Almería, Almería, Spain

ARTICLE INFO

Keywords:
Nonlinear multi-objective optimization
Multi-objective evolutionary algorithm
Parallelism
Efficiency
Effectiveness
Tuning and Analysis Utilities Performance System
Facility location problem

ABSTRACT

Real-life problems usually include conflicting objectives. Solving multi-objective problems (i.e., obtaining the complete efficient set and the corresponding Pareto-front) via exact methods is in many cases nearly intractable. In order to cope with those problems, several (meta) heuristic procedures have been developed during the last decade whose aim is to obtain a good discrete approximation of the Pareto-front. In this vein, a new multi-objective evolutionary algorithm, called FEMOEA, which can be applied to many nonlinear multi-objective optimization problems, has recently been proposed. Through a comparison with an exact interval branch-and-bound algorithm, it has been shown that FEMOEA provides very good approximations of the Pareto-front. Furthermore, it has been compared to the reference algorithms NSGA-II, SPEA2 and MOEA/D. Comprehensive computational studies have shown that, among the studied algorithms, FEMOEA was the one providing, on average, the best results for all the quality indicators analyzed. However, when the set approximating the Pareto-front must have many points (because a high precision is required), the computational time needed by FEMOEA may not be negligible at all. Furthermore, the memory requirements needed by the algorithm when solving those instances may be so high that the available memory may not be enough. In those cases, parallelizing the algorithm and running it in a parallel architecture may be the best way forward. In this work, a parallelization of FEMOEA, called FEMOEA-Paral, is presented. To show its applicability, a bi-objective competitive facility location and design problem is solved. The results show that FEMOEA-Paral is able to maintain the effectiveness of the sequential version and this by reducing the computational costs. Furthermore, the parallel version shows good scalability. The efficiency results have been analyzed by means of a profiling and tracing toolkit for performance analysis.

☆ This work has been funded by Grants from the Spanish Ministry of Science and Innovation (TIN2008-01117, ECO20121-2497 and TIN2012-37483-C03-03), Junta de Andalucía (P10-TIC-6002, P11-TIC7176 and P12-TIC-301), Program CEI from MICINN (PYR-2012-15 CEI BioTIC GENIL, CEB09-0010) and Fundación Séneca (The Agency of Science and Technology of the Region of Murcia, 00003/CS/10 and 15254/PI/10), in part financed by the European Regional Development Fund (ERDF). Juana López Redondo is a fellow of the Spanish 'Juan de la Cierva' contract program.

* Corresponding author.
E-mail addresses: agarrondo@um.es (A.G. Arrondo), jlredondo@ugr.es (J.L. Redondo), josefdez@um.es (J. Fernández), ortigosa@ual.es (P.M. Ortigosa).
URL: http://www.um.es/geloca/gio/josemain.html (J. Fernández).

http://dx.doi.org/10.1016/j.amc.2014.08.036
0096-3003/© 2014 Elsevier Inc. All rights reserved.

1. Introduction

Multi-objective optimization problems are ubiquitous. Many real-life problems require taking several conflicting points of view into account [2,3,11,12,20,29]. Thus, general multi-objective optimization algorithms able to cope with those hard-to-solve optimization problems are required.

In this paper, we deal with the general nonlinear multi-objective optimization problem (MOP), which can be formulated as follows:

$$\begin{array}{ll} \min & \{f_1(y),\ldots,f_m(y)\}, \\ \text{s.t.} & y \in S \subseteq \mathbb{R}^n, \end{array} \tag{1}$$

where $f_1,\ldots,f_m : \mathbb{R}^n \longrightarrow \mathbb{R}$ are m real-valued functions. Let us denote by $f(y) = (f_1(y),\ldots,f_m(y))$ the vector of objective functions and by $Z = f(S)$ the image of the feasible region.

When dealing with multi-objective problems we need to clarify what 'solving' a problem means. In the following, some widely known definitions are provided to explain the concept of solution of (1).

Definition 1. A feasible vector $y^* \in S$ is said to be *efficient* iff there does not exist another feasible vector $y \in S$ such that $f_l(y) \leqslant f_l(y^*)$ for all $l = 1,\ldots,m$, and $f_j(y) < f_j(y^*)$ for at least one index j ($j \in \{1,\ldots,m\}$). The set S_E of all the efficient points is called the *efficient set* or *Pareto-set*. If y_1 and y_2 are two feasible points and $f_l(y_1) \leqslant f_l(y_2)$ for all $l = 1,\ldots,m$, with at least one of the inequalities being strict, then we say that y_1 *dominates* y_2.

Efficiency is defined in the decision space. The corresponding definition in the criterion space is as follows:

Definition 2. An objective vector $z^* = f(y^*) \in Z$ is said to be *non-dominated* iff y^* is efficient. The set Z_N of all non-dominated vectors is called the *non-dominated set* or *Pareto-front*. If y_1 and y_2 are two feasible points and y_1 dominates y_2, then we say that $f(y_1)$ dominates $f(y_2)$.

Ideally, solving (1) means obtaining the whole efficient set, that is, all the points which are efficient, and its corresponding Pareto-front. However, for a majority of MOPs, it is not easy to obtain an exact description of the efficient set or Pareto-front, since those sets typically include an infinite number of points (usually a continuum set). To the extent of our knowledge, only three exact general methods, namely, three branch-and-bound methods (see [9,10,24]) have been proposed in literature which obtain an enclosure of those sets up to a pre-specified precision. Specifically, they offer a list of boxes (multi-dimensional intervals) whose union contains the complete efficient set (and their images the corresponding Pareto-front) as a solution. However, they are time consuming. Furthermore, they have large memory requirements, so that only small instances can be solved with them. Other interesting approaches can be found in [7,28].

Contrarily, the use of (meta) heuristics may allow us to obtain 'good approximations' of the Pareto-front, even for problems with more variables and objectives. By a good approximation we mean a discrete set of points spread over the complete Pareto-front and evenly distributed over it. There is a plethora of methods with that purpose in literature although most of them are designed to deal with combinatorial MOPs (some exceptions are [5,14,17,21,25]). Nonetheless, the most common approach utilized in literature to cope with (1) is the use of multi-objective evolutionary algorithms (MOEAs). This is due to their ability to find multiple efficient solutions in one single simulation run (see [4] for an excellent introduction to the topic).

Recently, a new Fast and Efficient Multi-Objective Evolutionary Algorithm (FEMOEA), aimed at quickly obtaining a good fixed size approximation of the Pareto-front has been presented [22,23]. It combines ideas from other typical MOEAs with some concepts from other evolutionary algorithms (EAs) devised to cope with single-objective optimization problems. Furthermore, in order to accelerate its convergence towards the optimal Pareto-front, FEMOEA includes two new devices: a new improving method and a new stopping rule. These two contributions can be included in any MOEA. In [22], FEMOEA was compared to iB&B, an interval branch-and-bound algorithm able to obtain an enclosure of the true Pareto-front, when solving the hard-to-solve competitive facility location problem described in [10]. A comprehensive computational study showed that FEMOEA was competitive, being able to reduce, in average, the computing time of the exact method by approximately 99%, and this offering good quality final solutions, i.e., all the points offered by the algorithm were (nearly) efficient and they were evenly distributed over the complete Pareto-front. Additionally, in [23], FEMOEA was compared to the well-known NSGA-II [6] and SPEA2 [33] algorithms, which have become the reference algorithms in the multi-objective evolutionary computation community. Besides, it was also compared to MOEA/D, a more recent algorithm which has proved to be one of the most competitive multi-objective evolutionary algorithms [30,31]. Several quality indicators were considered, namely, hypervolume, average distance, additive epsilon indicator, spread and spacing. According to the computational results and statistical analysis performed, the new algorithm outperformed, on average, the other three algorithms in all the quality indicators.

However, when the set approximating the Pareto-front must have many points (because a high precision is required), the computational time required by FEMOEA may not be negligible at all. Furthermore, the memory requirements needed by the algorithm when solving those instances may be so high that the available memory may not be enough. In those cases, parallelizing the algorithm and running it in a parallel architecture may be the best way forward. As far as the authors' knowledge is concerned, the development of parallel multi-objective evolutionary algorithms is a booming field, which has not been explored enough (see [1,15,16,26]).

The rest of the paper is organized as follows. In Section 2, the sequential algorithm is briefly described. In Section 3, the parallel version of FEMOEA, FEMOEA-Paral, is detailed. To show its applicability, a bi-objective competitive facility location and design problem is solved. The associated model and the corresponding computational study is shown in Section 4. Finally, in Section 5, the main conclusions are summarized.

2. The sequential method FEMOEA

This section is devoted to describing some concepts of FEMOEA which are needed to explain its parallelization. Details about the sequential version procedures are omitted. The interested reader is referred to [22,23] for further information.

The most important concept in FEMOEA is that of an individual. An individual is defined by a center and a radius. The center is a solution and the radius is a positive number which determines the subregion of the search space covered by that individual. The main aim of the radius is to focus the searching operators on the corresponding subregions. For further information about the concept as well as the calculation of the radius associated to each individual, see [19,22,23].

Apart from the center and the radius, an individual has two attributes which are related to the criterion space: the non-domination rank (d_{rank}) and the crowding distance (c_{dist}), see [6]. The non-domination rank indicates the number of individuals which dominate that particular individual. In this sense, a zero value means that such an individual is not dominated by any of the remaining ones in the current population. The crowding distance is an estimation of the density of solutions surrounding a particular solution in a population.

During the optimization process, two lists of individuals are kept by FEMOEA, whose maximum size M, the same for both lists, is a given input parameter. The parameter M refers to the desired number of solutions in the final Pareto-front. The first list, named *population_list*, is composed of M diverse individuals with different attributes, i.e. various radii, non-domination ranks and crowding distances. The second list, called *external_list*, can be understood as a deposit to keep non-dominated solutions.

Definition 3. A solution i is *preferable* to a solution i', $i \succ i'$, if $d^{i}_{rank} < d^{i'}_{rank}$ or $d^{i}_{rank} = d^{i'}_{rank}$ and $c^{i}_{dist} > c^{i'}_{dist}$.

The previous relation is known as *crowded comparison operator* (see [6]). To accelerate the selection process, both lists are always sorted according to such an operator.

In FEMOEA, each individual is intended to occupy an efficient solution. For this purpose, FEMOEA directs the individuals during the searching process towards the most suitable regions. Therefore, notice that a particular individual is not a fixed part of the search domain, but it can move through the space as the search proceeds. 'Individual-management' is one of the core parts of FEMOEA. It consists of procedures for creating and selecting individuals during the whole optimization process. Additionally, FEMOEA includes an improving method, which has been logically separated from the individual-management. This means that FEMOEA can easily be adapted to solve any other multi-objective problem, only adapting the improving technique. In [23], a new method to improve the efficiency of points, where no gradient information is used, was proposed, whereas in [22], a gradient-based method was designed.

It is important to mention that a single individual in the *population_list* can create a new offspring or be improved without participation of the remaining ones. Consequently, there exists an intrinsic parallelism, which consists of dividing the individuals among the number of available processors. Nevertheless, although there exists no relationship among the individuals in *the population_list*, the evolution of the population highly depends on the solutions stored in the *external_list*. Furthermore, the *external_list* may be modified (adding, removing or updating individuals) by procedures initially applied to the *population_list*. Then, the effectiveness of the parallel version may suffer a decrease if the embraced relationship between the *external_list* and the *population_list* is not taken into account.

3. The parallel algorithm FEMOEA-Paral

The programming paradigm used to parallelize FEMOEA may be considered a coarse-grain model, where each processor executes FEMOEA independently of the remaining ones most of the time but considering a smaller *population_list*. More precisely, the length of such a list will be equal to $M' = M/P$, assuming that P processors will be available. This list will be named *local_population_list* in the following. Therefore, the idea is that different processors work with a smaller and different list of individuals in such a way that, when merging all the local lists, a population list similar to that of the sequential version can be obtained. Nevertheless, although there exists no relationship among individuals in the *population_list*, and hence they can be distributed among the processors without problems, the *external_list* is not divided among the processors to prevent poor effectiveness. On the contrary, each processor has a local copy of it, which will be called *local_external_list* throughout this work.

An important issue to highlight is that, unlike the sequential version, those two lists are not sorted by the crowded comparison operator, but only by the first objective function value. Since the selection will be carried out in parallel, the maintenance of sorted lists by the crowded comparison operator is counter-productive in terms of efficiency.

Apart from these two lists, another list, called *auxiliary_external_list* is maintained during the optimization process. Such a list is only stored at the processor with identification number 0, P_0, and keeps the most preferable individuals found during the whole optimization process.

Algorithm 1 sketches the structure of the parallel algorithm. In the following, the different key stages are described.

Algorithm 1. FEMOEA-Paral

```
1: Init_individuals_lists_paral
2: while termination criteria are not satisfied
3:    Create_new_individuals_paral ($evals$)
4:    Select_individuals_paral ($local\_population\_list$)
5:    Improve_individuals_paral ($local\_population\_list$)
6:    Update_local_external_list
7:    Select_individuals_paral ($local\_external\_list$)
8:    Improve_individuals_paral ($local\_external\_list$)
9: end while
10: if length ($auxiliary\_external\_list$) $< M$
11:    Compose_pareto
12: end if
```

- *Init_individuals_lists_paral:* Initially, as many individuals as the parameter M' indicates are created at each processor. As in the sequential version, the center of the individuals are randomly computed, while their radii will be the radius associated at level 1. The *local_population_list* is initialized from this set of individuals, while the *local_external_list* will consist only of the non-dominated individuals.

After this procedure, a loop starts, which basically creates, selects and improves individuals. This loop is executed until a considerable improvement of the Pareto-front (placed in the *auxiliary_external_list*) is not obtained in three consecutive approximations or the maximum level L is achieved. L is an input parameter. Notice that the termination criteria is controlled by processor P_0.

- *Create_new_individuals_paral (evals):* The sequential *Create_new_individuals* procedure explores the search space to identify regions with good solutions. To achieve a balance between exploration and exploitation, the creation procedure applies a sequence of genetic operators, which use the accumulated experiences by making comparisons with the solutions stored in the *external_list*. The *Create_new_individuals_paral* method is similar to its sequential counterpart. The only difference appears in the population list length, which is equal to M for the sequential case and to $M' = M/P$ for the parallel version. The external list sizes are, in both cases, equal to M, although in the parallel version the *local_external_list* is used instead of the *external_list* of the sequential version.
- *Select_individuals_paral (list):* The sequential *Select_individuals* procedure reduces the *list* length when it reaches its maximum allowable capacity. Then, the most *preferable* individuals will be selected (see Definition 3). Notice that, in the parallel version, each processor maintains a *local* list. Performing selections by only considering the *local* information may be counterproductive in terms of effectiveness, since a solution which is not preferable in a particular processor could be preferable when taking all the local lists into account. Then, in order to prevent a decrement in the global effectiveness, selections are not carried out locally at each processor. Instead, a global selection considering all the lists at the P processors is accomplished. This may imply that large amounts of data must be frequently transferred among processors. To speed-up the communication overheads and hence the selection procedure, a hierarchical tree communication schema has been designed. Let us assume that the root of the tree is processor P_0. Then, processor P_0 may be understood as the collector of all the transferred information. The maximum width of the hierarchical tree is given by the number of available processors P, which are identified by P_{id} with $id \in [0, P-1]$. Its maximum number of stages is given by $stg^{\max} = \log_2(P)$. Each stage has associated a figure $stg \in [0, stg^{\max}]$.
  There exist three kinds of processors: senders, receivers and idle processors. As can be observed in Fig. 1, the role of each processor varies through the communication model. When a processor is a receiver, it obtains a list from a sender processor, composes a joint list considering the own individuals list and the received one and computes the d_{rank} value associated to each individual. Then, a selection which will vary depending on the transferred *list* and the stage of the communication model, is carried out.
  - If *list* refers to *local_population_list*. In this case, if $stg < stg^{\max}$, the selection is only carried out in terms of domination ranks. More precisely, the minimum non-domination rank $d_{rank}^{\min}$ in such a way that *at least* $2 \cdot M' \cdot 2^{stg+1}$ individuals exist with a non-domination rank smaller than or equal to $d_{rank}^{\min}$ is computed. Those individuals with $d_{rank} \geqslant d_{rank}^{\min}$ will be removed from the joint list. The remaining ones will be transferred through the communication hierarchical tree. In the last stage of the selection procedure, i.e. $stg = stg^{\max}$, the receiver processor (P_0) is the hierarchical tree root, and it will select the M most preferable individuals by using the crowded comparison operator. The resulting *population_list* will then be distributed back among all the processors, in such a way that M' individuals will be sent directly to each processor.

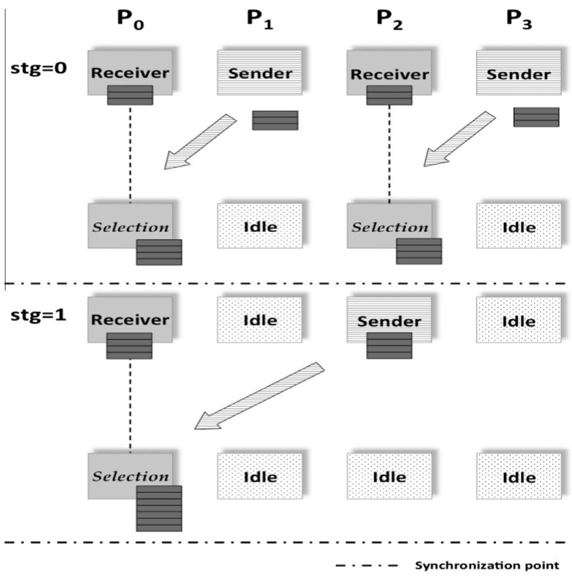

Fig. 1. Select_individuals_paral procedure.

– If *list* refers to *local_external_list* and $stg < stg^{max}$, only the non-dominated individuals will be maintained in the joint list and transferred to the next stage of the hierarchical communication tree. If $stg = stg^{max}$, processor P_0 (the receiver) will reduce the joint list to the M most preferable individuals and will store it in its own *local_external_list*. Then, P_0 distributes its *local_external_list* directly among all the processors as explained before.

- *Improve_individuals_paral (list):* This method is the parallelization of the sequential *Improve_individuals(list)* procedure, where each individual in the input *list* invokes an improving method. Notice that it is possible to implement any kind of local optimizer to work with the individual. When the calling individual belongs to the *population_list*, the local method allows, on the one hand, to push that individual towards the true Pareto-set and, on the other hand, to study its surrounding area to obtain indeterminate solutions, which may be inserted into the *external_list*. The inclusion of indeterminate points in the *external_list* may improve the quality of the final Pareto-front, but it increases the computational effort (the more elements in the list, the more computing time required to manage the *external_list*). Looking for a compromise between quality of the final Pareto-front and computational effort, indeterminate solutions are not inserted into the *external_list* when the *external_list* is considered as input in the *Improving_method*.
  Improve_individuals_paral executes the sequential procedure on the corresponding local lists. Then, in Step 5 of Algorithm 1, each processor improves its own *local_population_list*, with a length equal to M' and, as a consequence, the individuals may be substituted by solutions which dominate them and new points may be included in the *local_external_list* in case indeterminate individuals are found.
  On the contrary, in Step 8 of Algorithm 1, each processor applies the improving method to the *local_external_list* (with length equal to M') just sent by P_0. Notice that the received individuals belong to the *local_external_list* of P_0, which includes, at this moment, the most preferable solutions of all the local external lists. Once all the individuals have been optimized, they are sent back to processor P_0 by using the proposed communication model. If $stg < stg^{max}$, the selection procedure carried out by the receiver processors will maintain the non-domination individuals. On the contrary, if $stg = stg^{max}$, processor P_0 will join the *auxiliary_external_list* to the received improved individuals, and will apply a selection procedure, where only the M most preferable individuals are maintained. Finally, P_0 will directly send the whole *auxiliary_external_list* to each processor, which becomes their new *local_external_list*.
- *Update_local_external_list*: After the previous procedure, the *local_population_list* may contain individuals which deserve to be included in the *local_external_list*. Then, similar to the sequential version, the *local_external_list* is updated by copying the non-dominated individuals of the *local_population_list* to it. Of course, this implies that the points in the *local_external_list* dominated by the new ones have to be removed. Finally, a selection procedure is carried out over the *local_external_list*, where the M most preferable solutions are chosen.
- *Compose_pareto*: The solution provided by the algorithm must include M individuals. If the number of individuals in the *auxiliary_external_list* reaches this value, the set offered as an approximation of the Pareto-set will be the one kept in that list. Otherwise, the local population lists are sent to P_0 using the hierarchical tree communication schema explained before, which joins these lists to the *auxiliary_external_list*, and the M most preferable individuals will be offered as a result.

4. Experimental analysis

To show the applicability of FEMOEA-Paral, we have used it to solve a hard-to-solve competitive facility location (and design) problem. In particular, we revisit the bi-objective problem described in [10] which, for the sake of completeness, is briefly described in Section 4.1.

4.1. A bi-objective planar franchisor–franchisee facility location and design problem

A franchise wants to increase its presence in a given geographical region by opening one new facility. Both the franchisor (the owner of the franchise) and the franchisee (the actual owner of the new facility to be opened) have the same objective: maximizing their own profit. However, the maximization of the profit obtained by the franchisor is usually in conflict with the maximization of the profit obtained by the franchisee, as we will see. In the model, the *demand* is supposed to be *inelastic* (fixed) and concentrated at some demand points, which split their buying power among *all* the facilities proportionally to the *attraction* they feel for them (see [13]). The attraction (or utility) function of a customer towards a given facility depends on the distance between the customer and the facility as well as on other characteristics of the facility which determine its *quality*.

The following notation will be used throughout this paper. The variables of the new facility (i.e., the variables of the problem) are denoted by $nf = (x, \alpha)$, where x represents the location of the new facility, $x = (x_1, x_2)$, and α its quality ($\alpha > 0$). Let q be the index of demand points, $q = 1, \ldots, q_{\max}$, and r the index of existing facilities, $r = 1, \ldots, r_{\max}$. p_q is the location of the qth demand point; w_q refers to the demand (or buying power) at p_q; ef_r is the location of the rth existing facility; d_{qr} is the distance between p_q and ef_r; and α_{qr} is the quality of ef_r as perceived by p_q. With the previous notation, $\alpha_{qr}/g_q(d_{qr})$ is the attraction that p_q feels for ef_r, where $g_q(\cdot)$ is a non-negative non-decreasing function. Furthermore, let γ_q be the weight for the quality of the new facility as perceived by p_q, and d_{qx} be the distance between p_q and the new facility nf. Then $\gamma_q\alpha/g_q(d_{qx})$ gives the attraction that p_q feels for nf. Finally, let k be the number of existing facilities that are part of the franchise (the first k of the $r_{\max}$ facilities are assumed to be in this category, $0 < k < r_{\max}$). From the previous assumptions, the market share attracted by the franchisor is

$$MK(nf) = \sum_{q=1}^{q_{\max}} w_q \frac{\frac{\gamma_q\alpha}{g_q(d_{qx})} + \sum_{r=1}^{k} \frac{\alpha_{qr}}{g_q(d_{qr})}}{\frac{\gamma_q\alpha}{g_q(d_{qx})} + \sum_{r=1}^{r_{\max}} \frac{\alpha_{qr}}{g_q(d_{qr})}}.$$

We assume that the operating costs for the franchisor pertaining to the new facility are fixed. In this way, the profit obtained by the franchisor is an increasing function of the market share that it captures. Thus, maximizing the profit obtained by the franchisor is equivalent to maximizing its market share. This will be the first objective of the problem.

The second objective of the problem is the maximization of the profit obtained by the franchisee, to be understood as the difference between the revenues obtained from the market share captured by the new facility minus its operational costs. The market share captured by the new facility (franchisee) is given by

$$mk(nf) = \sum_{q=1}^{q_{\max}} w_q \frac{\frac{\gamma_q\alpha}{g_q(d_{qx})}}{\frac{\gamma_q\alpha}{g_q(d_{qx})} + \sum_{r=1}^{r_{\max}} \frac{\alpha_{qr}}{g_q(d_{qr})}}$$

and the profit is given by the following expression,

$$\pi(nf) = F(mk(nf)) - G(nf),$$

where $F(\cdot)$ is a strictly increasing function which determines the expected sales (i.e., income generated) for a given market share $mk(nf)$, and $G(nf)$ is a function which gives the operating costs of a facility located at x with quality α. In our computational studies we have considered F to be linear and G to be separable, in the form $G(nf) = G_1(x) + G_2(\alpha)$, where $G_1(x) = \sum_{q=1}^{q_{\max}} \Phi_q(d_{qx})$, with $\Phi_q(d_{qx}) = w_q/((d_{qx})^{\phi_{q0}} + \phi_{q1})$, $\phi_{q0}, \phi_{q1} > 0$, and $G_2(\alpha) = e^{\frac{\alpha}{\alpha_0}+\alpha_1} - e^{\alpha_1}$, with $\alpha_0 > 0$ and α_1 given values (other possible expressions for $G(nf)$ can be found in [8]).

The problem considered is

$$\begin{cases} \max & MK(nf), \\ \max & \pi(nf), \\ \text{s.t.} & d_{qx} \geqslant d_q^{\min} \ \forall q, \\ & \alpha \in [\alpha_{\min}, \alpha_{\max}], \\ & x \in FR \subset \mathbb{R}^2, \end{cases} \tag{2}$$

where the parameters $d_q^{\min} > 0$ and $\alpha_{\min} > 0$ are given thresholds, which guarantee that the new facility is not located over a demand point and that it has a minimum level of quality, respectively. The parameter $\alpha_{\max}$ is the maximum value that the quality of a facility may take in practice. By FR we denote the region of the plane where the new facility can be located. Notice that (2) is a particular case of (1), in which $y = nf, f_1(y) = -MK(nf), f_2(y) = -\pi(nf)$ and the feasible set S is given by the constraints in (2).

4.2. Computational results

All the computational studies have been run in a cluster with 18 nodes. Each node has 16 cores (Intel Xeon E5 2650), 64 GB of shared memory and 128 GB of solid-state drive. In total, 288 cores, 1151 GB of memory and 2304 GB of SSD. In this work, each instance has been solved with $P = 1, 2, 4, 8, 16, 32, 64$ processors. The executions with $P = 1, 2, 4, 8, 16$ processors were executed in a single node, selecting in each case a number of cores equal to P. The studies with 32 processors were run by using 2 nodes and 16 cores per node. Finally, for $P = 64$, 4 nodes were selected with 16 cores per node. The interconnection networks are Infiniband and Ethernet.

In order to have an overall view of the performance of the algorithm, different types of problems have been generated, varying the number $q_{\max}$ of demand points, the number $r_{\max}$ of existing facilities and the number k of those facilities belonging to the chain. The settings used were ($q_{\max} = 25, 50, r_{\max} = 2, k = 1$), ($q_{\max} = 25, 50, r_{\max} = 5, k = 1, 2$) and ($q_{\max} = 25, 50, r_{\max} = 10, k = 2, 4$). For every setting, one instance was generated by randomly choosing the parameters of the problems uniformly within pre-defined intervals (see [10]). This set of ten problems can be downloaded from `http://www.um.es/geloca/gio/AMC-testproblems.zip`. The number of points selected to approximate the Pareto-front was set to $M = 400, 800, 1600, 3200, 6400$. To counteract the randomness effect, each problem has been executed 10 times and average values have been computed.

The effectiveness has been tested, on the one hand, by checking whether the points in the solution set offered by FEMOEA-Paral approximating the efficient set are certainly efficient points (or are very close to efficient points). To this aim, similar to what was done in [22] with the sequential version, we check whether the solution points are included in the corresponding solution boxes offered by the interval branch-and-bound method iB&B described in [10]. On the other hand, for measuring the goodness of an approximation to the whole Pareto-front, the so-called *hypervolume* indicator [32] has also been computed. It measures the hypervolume of the portion of the criterion space that is weakly dominated by the approximation set (see Fig. 2-left). The higher the hypervolume, the better the approximation. In order to measure this quantity, a reference point that is dominated by all points is needed (see point RP in Fig. 2). For a given problem, the same reference point has to be used for all the algorithms and all the runs. In our computational studies, we have considered the points of all the approximation sets of the Pareto-front together (for fixed values of P and M we solve each problem 10 times, thus for every problem we have 350 approximation sets), and we have used as reference point the one whose lth component is the maximum of all the lth components of those points. It is an approximation of the Nadir Point.

For each particular problem j ($j = 1 \ldots, 10$), iB&B offers as a solution a list of boxes (multi-dimensional intervals) whose union contains the complete efficient set, and their images the corresponding Pareto-front. To compute the hypervolume of that solution, the upper-right and lower-left corners of the boxes in the image space are obtained (see Fig. 2-right). Those points form two approximation sets. An interval $[lowH_j, uppH_j]$ which contains the exact hypervolume of the true Pareto-front can be obtained as follows. Its lower bound $lowH_j$ is the hypervolume obtained with the set composed by the upper-right corners of the boxes and its upper bound $uppH_j$ is the hypervolume obtained with the set composed of the lower-left corners. It is important to mention that, prior to the computation of those hypervolumes, dominated points are removed from each set. For the sake of brevity, the particular intervals obtained for each problem are not shown. But in order to have a general overview, the averages of the lower and upper limits for the whole set of ten problems have been computed, $[lowH, uppH] = [361.9153, 364.0888]$. The average computing time spent by iB&B at solving the instances is 23.5 hours.

Notice that the hypervolume becomes larger as the number of points approximating the Pareto-front increases. In this sense, the behavior of both FEMOEA and FEMOEA-Paral when the number of points approximating the Pareto-front, M, increases has been studied.

It is worth mentioning that FEMOEA-Paral has approximated the Pareto front with 100% success, i.e., in all the runs and for all the problems, all the points were included in the corresponding boxes offered by iB&B. Furthermore, the hypervolume values have always been included in the interval $[lowH_j, uppH_j]$ for any particular instance $j, j = 1, \ldots, 10$. As expected, it can be observed (see Table 1) that the hypervolume increases with the number of points M in the Pareto-front. Besides, the hypervolume obtained by the parallel version is similar to the one obtained by the sequential method. This has been a challenge, since the sequential version has global control over both the population and the external lists, whereas the parallel

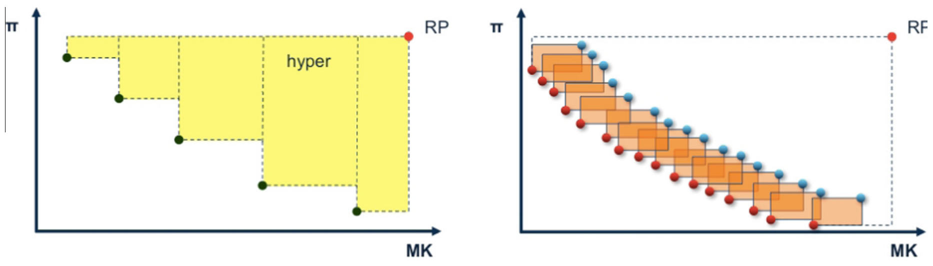

Fig. 2. Computation of the hypervolume value for an approximation set (left) and approximation sets considered for the iB&B algorithm (right).

Table 1
Average results obtained by FEMOEA grouped according to the values of M and P.

P	$M = 400$			$M = 800$			$M = 1600$			$M = 3200$			$M = 6400$		
	$\overline{Hyper}$	$\overline{SD}$	$\overline{Eff}$	$\overline{Hyper}$	$\overline{SD}$	$\overline{Eff}$	$\overline{Hyper}$	$\overline{SD}$	$\overline{Eff}$	$\overline{Hyper}$	$\overline{SD}$	$\overline{Eff}$	$\overline{Hyper}$	$\overline{SD}$	$\overline{Eff}$
1	363.0677	0.0073	1007	363.4107	0.0023	3528	363.5803	0.0005	14532	363.6609	0.0001	65219	363.6997	0.0000	343761
2	363.1025	0.0064	1.41	363.4256	0.0014	1.48	363.5852	0.0004	1.72	363.6626	0.0001	2.05	363.7003	0.0000	2.66
4	363.0996	0.0035	1.69	363.4261	0.0020	1.80	363.5851	0.0006	2.24	363.6626	0.0002	2.86	363.7003	0.0000	4.12
8	363.1018	0.0050	1.60	363.4264	0.0016	1.66	363.5846	0.0007	2.60	363.6625	0.0002	3.79	363.7003	0.0000	4.99
16	363.1018	0.0060	1.18	363.4252	0.0023	1.43	363.5850	0.0004	2.02	363.6624	0.0002	2.98	363.7003	0.0000	3.92
32	363.1029	0.0062	0.71	363.4257	0.0014	0.92	363.5851	0.0005	1.23	363.6625	0.0001	1.83	363.7002	0.0000	2.66
64	363.1029	0.0042	0.37	363.4256	0.0018	0.48	363.5846	0.0004	0.65	363.6623	0.0002	0.96	363.7002	0.0000	1.46

version carries out some decisions based on local knowledge. The designed selection procedure has helped to counteract this drawback.

To measure the computational effort in obtaining the solutions, the efficiency measure, *Eff*, which estimates how well-utilized the processors are in solving the problem, has been computed. The efficiency of a parallel version (run over P processors) with respect to the sequential one is computed as: $Eff(P) = \frac{T(1)}{P \cdot T(P)}$, where $T(1)$ is the execution time on a uni-processor and $T(P)$ is the execution time on P processors. Notice that the ideal efficiency is 1.

Furthermore, the *scalability* of the parallel version has also been tested. Broadly speaking, this concept can be understood as the ability of a system, algorithm, or process, to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth. The use of performance metrics such as the efficiency allows us to determine whether an algorithm is scalable: an algorithm that scales well will be able to maintain or even increase its level of performance or efficiency when tested on more demanding instances.

For the sake of brevity, the particular results for each particular problem are not shown; but in order to have a general overview, the average efficiency results for the different values of P and grouped them according to the value of M are detailed (see columns $\overline{Eff}$ in Table 1). Notice that for $P = 1$, the computing time (in seconds) has been depicted instead. The average hypervolume and the corresponding average of the standard deviations have also been included. To be more precise, for fixed M and P values, for each of the 10 problems (each of them executed 10 times) we compute its mean hypervolume and its corresponding standard deviation $(\overline{Hyper}_j, SD_j)$, $j = 1 \ldots, 10$. The values in Table 1 are $\overline{Hyper} = \sum_{j=1}^{10} \overline{Hyper}_j / 10$ and $\overline{SD} = \sum_{j=1}^{10} SD_j / 10$.

Observe that, as expected, $\overline{Hyper}$ is always included in the interval $[lowH, uppH]$. Notice also that the higher the M, the smaller the standard deviation. This is not surprising, since the more points in the set approximating the Pareto-front, the better the approximation is, the smaller the difference between the approximation sets can be, and the smaller the difference in their corresponding hypervolume is. Concerning efficiency, it increases up to 8 processors, and then it starts to decrease. Even so, promising results have been obtained and values very superior to the ideal case have been achieved in most of the cases. Furthermore, FEMOEA-Paral scales, since the efficiency values improve as the computational load of the problem to be solved increases (with the value of M).

The behavior of the parallel version and hence, the obtained efficiencies, are mainly due to: (i) the distribution of the lists among the available processors, which reduces the cache misses and hence, the memory access time; and (ii) the overheads imposed by the exchange of information among processors, which takes place at the *Select_individuals_paral (list)* method. These conclusions have been inferred from a study conducted by using the TAU (Tuning and Analysis Utilities) Performance System [18] and PAPI (Performance Application Programming Interface) [27]. It is important to mention that the tendencies observed in Table 1, are not a consequence of computing average values. On the contrary, the behavior is similar for all the particular instances. That is why only the results obtained for a particular problem (the one with setting $(50, 10, 4)$) will be shown next, when it is analyzed using TAU Performance System.

One of the key points studied using TAU and PAPI has been the total cache misses obtained by both FEMOEA ($P = 1$) and FEMOEA-Paral ($P > 1$), for every value of M. Fig. 3 depicts those results for the instance $(50, 10, 4)$. As can be seen, given a value of M, the number of cache misses decreases exponentially for up to 8 processors. From that point on, such a decrement is not as important, but much slower. Besides, notice that the larger the value of M, the higher the number of cache misses of the sequential version and therefore, the greater the time savings obtained by the parallel versions due to the reduction of the cache faults. Of course, these facts are clearly related to the efficiencies obtained, which are superior to the ideal case and higher as the value of M increases.

The computing time associated to the *Select_individuals_paral (list)* method has also been studied using TAU. The behavior of such a method and hence, the conclusions that can be inferred, can be extrapolated to any input *list* (with any length). Then, for the sake of brevity, only a particular example will be shown. More precisely, the case with setting $(50, 10, 4)$, $M = 3200$ and *list* referring to *local_population_list*, will be considered. Fig. 4 shows the computing time employed by each processor (named node in the figure) to execute this method, for that particular instance and when $P = 2, 4, 8, 16$. The graphics associated with $P = 32, 64$ have been omitted to reduce the length of the paper, although the same tendency was observed. Furthermore, notice that some additional figures are provided, i.e. the maximum, mean and minimum execution times spent by the processors are shown. The standard deviation is also depicted. As can be seen, there exists a great imbalance in the computing

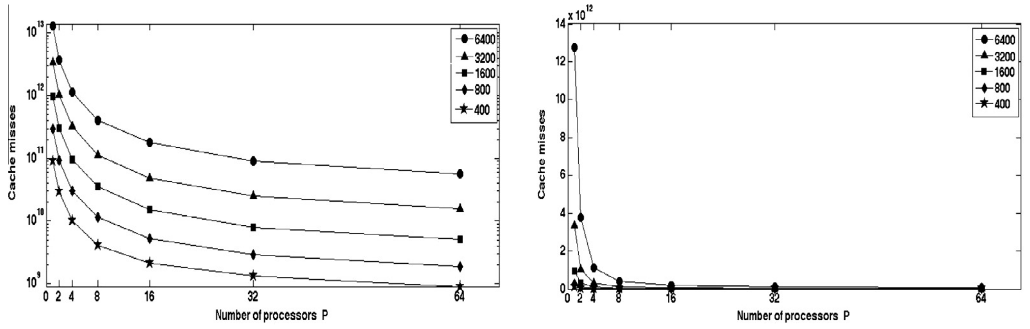

Fig. 3. Setting (50, 10, 4). Average total cache misses obtained by FEMOEA and FEMOEA-Paral with 400, 800, 1600, 3200and 6400 points in the Pareto-front. (a) On the left in linear scale, (b) on the right in logarithmic scale.

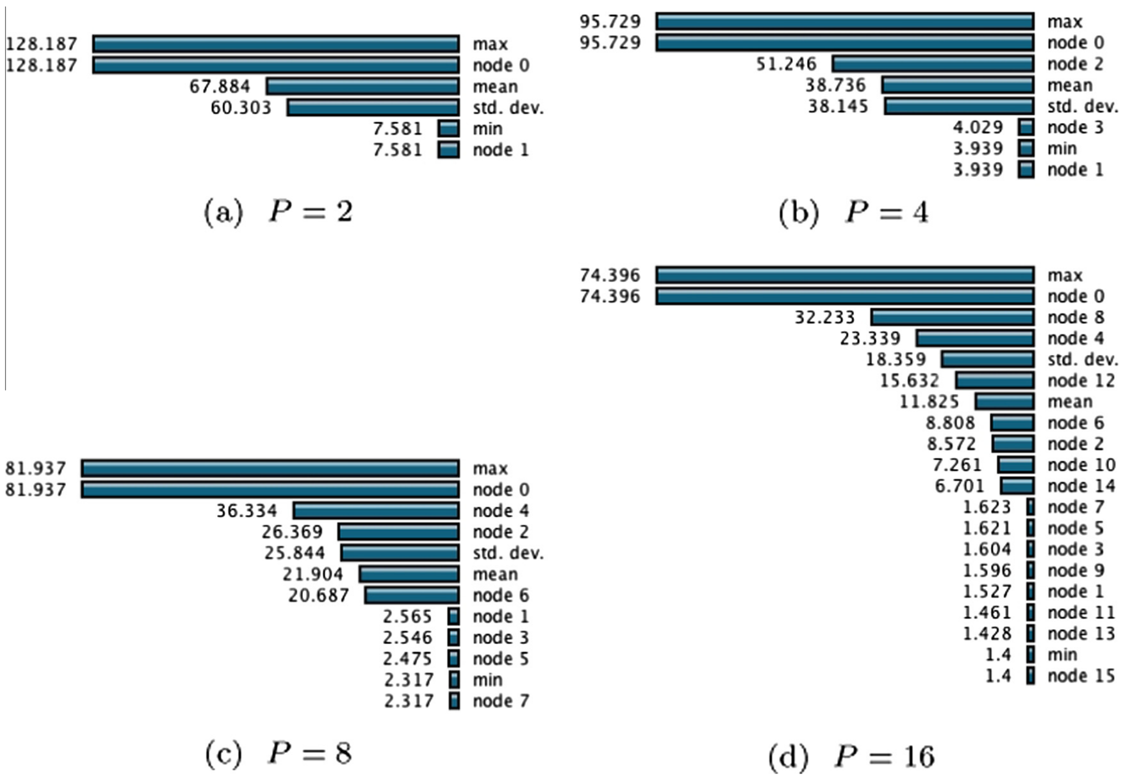

Fig. 4. Setting (50, 10, 4). Computational time employed in Select_individuals_paral (*local_population_list*) with $P = 2, 4, 8, 16$ and $M = 3200$.

times associated to each processor. Bear in mind the communication tree schema to understand the reasons behind such an imbalance. Following with the notation previously defined in this paper, processors with an odd identification number will initially be receivers while the ones with an even number will be senders. Processor P_0 (node 0 in the figure) represents the collector. As expected, it is always active and hence, its computing time is always the maximum one. Besides, the number of idle processors increases with the value of P. These results clearly show that the selection procedure is a bottleneck and provokes a reduction in the efficiency of the parallel version. Nevertheless, notice that, in spite of the time costs (due to communications and waiting times imposed by this method), good efficiency values are obtained. This is because these overheads are compensated by the time savings associated to the memory accesses previously commented. Notice that, when $P > 8$, those savings are not as spectacular and therefore, the effects of both communication and waiting times have a greater influence on the efficiency. This fact explains the existence of an inflexion point in the efficiency values when $P = 8$, i.e. the efficiency increases for up to 8 processors and then it begins to decrease. The selection procedure was designed to be able to maintain the *effectiveness* of the sequential version; other decisions which may improve this deficiency may in turn reduce the quality of the obtained solutions.

5. Conclusions

To deal with hard-to-solve multi-objective optimization problems, as most competitive location problems are, a parallel version of FEMOEA, called FEMOEA-Paral, has been developed and analyzed. A comprehensive computational study has shown that FEMOEA-Paral maintains the effectiveness of the sequential version, i.e. both versions approximate the Pareto-front with 100% success in all the instances, their hypervolume values are always included in the interval provided by iB&B and they both obtain similar hypervolume values for any particular instance. The maintenance of the effectiveness values is made possible thanks to the implemented selection procedure, which allows us to concurrently choose the most preferable solutions. The efficiency of the parallel version has also been tested. The distribution of the computational load carried out by FEMEOA-Paral allows us to highly accelerate the sequential computational times, in such a way that FEMOEA-Paral has been able to obtain super efficiency values. Additionally, the scalability of the parallel version has also been shown by solving instances with a larger computational burden.

Efficiency results have been analyzed by using TAU. It can be concluded that the parallelization overhead increases with the number of processors. Furthermore, it was found that whereas the distribution of the population among the processors saves CPU time as compared to the sequential version (the number of cache misses is smaller), the opposite holds with the selection procedure, which acts as a bottleneck in the algorithm. New selection procedures, able to maintain the effectiveness, but without causing load imbalance, should be researched. Additionally, methods to reduce the cache misses for the sequential as well as for the parallel version should be studied. An analytical study should also be carried out to try to characterize the improvements that can be achieved with the parallelization.

References

[1] E. Alba, G. Luque, S. Nesmachnow, Parallel metaheuristics: recent advances and new trends, Int. Trans. Oper. Res. 20 (1) (2013) 1–48.
[2] E.E. Ammar, E.A. Youness, Study on multiobjective transportation problem with fuzzy numbers, Appl. Math. Comput. 166 (2) (2005) 241–253.
[3] C.A.C. Coello, G.B. Lamont (Eds.), Applications of Multi-objective Evolutionary Algorithms, World Scientific, Singapore, 2004.
[4] C.A.C. Coello, G.B. Lamont, D.A. Van Veldhuizen, Evolutionary algorithms for solving multi-objective problems, second edition., Genetic and Evolutionary Computation. Springer, New York, 2007.
[5] A.L. Custódio, J.F.A. Madeira, A.I.F. Vaz, L.N. Vicente, Direct multisearch for multiobjective optimization, SIAM J. Optim. 21 (3) (2011) 1109–1140.
[6] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[7] Y.G. Evtushenko, M.A. Posypkin, Nonuniform covering method as applied to multicriteria optimization problems with guaranteed accuracy, Comput. Math. Math. Phys. 53 (2) (2013) 144–157.
[8] J. Fernández, B. Pelegrín, F. Plastria, B. Tóth, Solving a Huff-like competitive location and design model for profit maximization in the plane, Eur. J. Oper. Res. 179 (3) (2007) 1274–1287.
[9] J. Fernández, B. Tóth, Obtaining an outer approximation of the efficient set of nonlinear biobjective problems, J. Global Optim. 38 (2) (2007) 315–331.
[10] J. Fernández, B. Tóth, Obtaining the efficient set of nonlinear biobjective optimization problems via interval branch-and-bound methods, Comput. Optim. Appl. 42 (3) (2009) 393–419.
[11] J. Figueira, S. Greco, M. Ehrgott (Eds.), Multiple Criteria Decision Analysis: State of the Art Surveys, Kluwer, New York, 2005.
[12] D. Fotakis, E. Sidiropoulos, A new multi-objective self-organizing optimization algorithm (MOSOA) for spatial optimization problems, Appl. Math. Comput. 218 (9) (2012) 5168–5180.
[13] S.L. Hakimi, Locations with spatial interactions: competitive locations and games, in: R.L. Francis, P.B. Mirchandani (Eds.), Discrete Location Theory, Wiley/Interscience, 1990, pp. 439–478.
[14] D. Jaeggi, G. Parks, T. Kipouros, J. Clarkson, A multi-objective tabu search algorithm for constrained optimisation problems, in: C.A.C. Coello, A.H. Aguirre, E. Zitzler (Eds.), Evolutionary Multi – A Criterion Optimization. Third International Conference, EMO 2005, Lecture Notes in Computer Science, vol. 3410, Springer, 2005, pp. 490–504.
[15] A. Lančinskas, J. Žilinskas, Approaches to parallelize Pareto eanking in NSGA-II algorithm, in: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Waśniewski (Eds.), Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 7204, Springer, Berlin, Heidelberg, 2012, pp. 371–380.
[16] A. Lančinskas, J. Žilinskas, Solution of multi-objective competitive facility location problems using parallel NSGA-II on large scale computing systems, in: Pekka Manninen, Per Öster (Eds.), Applied Parallel and Scientific Computing, Lecture Notes in Computer Science, vol. 7782, Springer, Berlin, Heidelberg, 2013, pp. 422–433.
[17] A.J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J.J. Durillo, A. Beham, AbYSS: adapting scatter search to multiobjective optimization, IEEE Trans. Evol. Comput. 12 (4) (2008) 439–457.
[18] Dpt. of Computer and Information Science University of Oregon, Advanced Computing Laboratory LANL NM, and Research Centre Julich ZAM Germany. TAU (Tuning and Analysis Utilities). <http://www.cs.uoregon.edu/research/tau/home.php>.
[19] P.M. Ortigosa, I. García, M. Jelásity, Reliability and performance of UEGO, a clustering-based global optimizer, J. Global Optim. 19 (3) (2001) 265–289.
[20] M.S. Osman, M.A. Abo-Sinna, A.A. Mousa, An effective genetic algorithm approach to multiobjective routing problems (MORPs), Appl. Math. Comput. 163 (2) (2005) 769–781.
[21] P.M. Pardalos, I. Steponavičė, A. Žilinskas, Pareto set approximation by the method of adjustable weights and successive lexicographic goal programming, Optim. Lett. 6 (4) (2012) 665–678.
[22] J.L. Redondo, J. Fernández, J.D. Álvarez, A.G. Arrondo, and P.M. Ortigosa, Approximating the Pareto-front of a planar bi-objective competitive facility location and design problem, Comput. Oper. Res. doi:http://dx.doi.org/10.1016/j.cor.2014.02.013.
[23] J.L. Redondo, J. Fernández, P.M. Ortigosa, FEMOEA: a fast and efficient multi-objective evolutionary algorithm, submitted for publication. http://www.um.es/geloca/gio/FEMOEA.pdf.
[24] D. Scholz, Deterministic global optimization: geometric branch-and-bound methods and their applications, Optimization and its Applications, Springer, 2012.
[25] P. Shelokar, V. Jayaraman, B. Kulkarni, Ant algorithm for single and multiobjective reliability optimization problems, Qual. Reliab. Eng. Int. 18 (6) (2002) 497–514.
[26] R.G. Strongin, Y.D. Sergeyev, Global optimization with non-convex constraints: sequential and parallel algorithms, Nonconvex Optimization and its Applications, Kluwer Academic Publishers, Dordrecht, 2000.
[27] Innovative Computing Laboratory University of Tennessee (U.S.A.), PAPI (Performance Application Programming Interface). <http://icl.cs.utk.edu/papi/>.
[28] A. Žilinskas, A one-step worst-case optimal algorithm for bi-objective univariate optimization, Optim. Lett. (2013), http://dx.doi.org/10.1007/s11590-013-0712-8.
[29] D.J. White, A bibliography on the applications of mathematical programming multiple-objective methods, J. Oper. Res. Soc. 41 (8) (1990) 669–691.

[30] Q. Zhang, H. Li, MOEA/D: a multiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (6) (2007) 712–731.
[31] Q. Zhang, H. Li, Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II, IEEE Trans. Evol. Comput. 13 (2) (2009) 284–302.
[32] E. Zitzler, Evolutionary algorithms for multiobjective optimization: methods and applications [Master's thesis], Swiss Federal Institute of Technology (ETH) Zurich, Shaker Verlag, Germany, 1999.
[33] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: improving the strength Pareto evolutionary algorithm for multiobjective optimization, in: K.C. Giannakoglou, D.T. Tsahalis, J. Périaux, K.D. Papailiou, T. Fogarty (Ed.), Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems, International center for numerical methods in engineering (CIMNE), Athens, Greece, 2002, pp. 95–100.