Distributed multi-objective metaheuristics for real-world structural optimization problems

Francisco Luna[1], Gustavo R. Zavala[2], Antonio J. Nebro[3], Juan J. Durillo[4] and Carlos A. Coello Coello[5]

[1]*Departamento de Sistemas Informáticos y Telemáticos, Centro Universitario de Mérida, Universidad de Extremadura, Spain*
[2]*Khaos Research Group, University of Málaga, Spain*
[3]*Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain*
[4]*Institute of Computer Science, University of Innsbruck, Austria*
[5]*CINVESTAV-IPN, México*
Email: fluna@unex.es, grz@lcc.uma.es, antonio@lcc.uma.es, juan@dps.uibk.ac.at, ccoello@cs.cinvestav.mx

The design of bar structures in civil engineering is a complex problem when dealing with real-world structures. An approach to deal with these problems is to apply metaheuristics, which are stochastic methods based on iteratively producing and evaluating tentative solutions. In particular, we focus on multi-objective metaheuristics, as we consider two goals to be minimized: the weight and the deflection of the structure. When applying these techniques to real-world problems, running a metaheuristic for several thousands of evaluations may require many days on a single processor. In this paper, we develop distributed master/slave versions of four multi-objective metaheuristics which are representative of the state-of-the-art and apply them to optimize the design of two instances of a cable-strayed bridge. Our study reveals that our parallel proposals are able to effectively use up to 450 cores, providing accurate solutions in a short time.

Keywords: Multi-objective Optimization; Parallelism; Large Scale Structural Optimization;

Received 00 February 2014; revised 00 Month 2014

1. INTRODUCTION

In the process of designing civil structures, engineers have frequently to face very hard optimization problems before the infrastructure becomes a reality. The hardness of these optimization problems emerges from different sources such as their inherent complexity, the presence of multiple conflicting design objectives, or the computational cost required for their resolution [1]. Of course, this kind of optimization problems are not specific to civil engineering and they also appear in many other disciplines such as telecommunications, biology, finance, chemistry, etc. [2, 3].

The aim is this paper is to address two instances of a real-world structural design problem, namely, the design of a cable-strayed bridge. The complexity of the problem lies mainly in the number of components to optimize (the dimensions of hundreds of bars) and on the number of side-constraints that feasible solutions must fulfill. The design problem is stated as that of optimizing two conflicting criteria: minimizing the financial cost of the structure and maximizing its safety. It is therefore a multi-objective optimization problem (MOP), in which the goal is to find a set of compromise solutions with different trade-offs among different criteria. This set of compromise solutions is known as *Pareto optimal set*, and its projection in objective function space is called *Pareto front*.

Many techniques have been proposed in the multi-objective research community to address the resolution of these problems. Unlike classical mathematical programming approaches, metaheuristics [4] in general, and Evolutionary Algorithms (Multi-Objective Evolutionary Algorithms or MOEAs, in the multi-objective domain) [5] in particular, have been widely used in the last decade because of two main facts. First, they have the ability to approximate the entire Pareto optimal set in one single run, as opposed to classical multi-criteria decision making techniques. They are also less sensitive to the shape of the Pareto front and, therefore, can deal with a large variety of multi-objective optimization problems. Second, as randomized black-box algorithms,

MOEAs can address optimization problems with non-linear, non-differentiable, or noisy objective functions.

In spite of these advantages, multi-objective metaheuristics might demand high computational resources because, on the one hand, they need to approximate a set of solutions rather a single one (i.e., the Pareto optimal set); on the other hand, and even more importantly, many real-world multi-objective problems typically use computationally expensive methods for computing the objective functions and constraints. As the working principle of MOEAs is based on generating and evaluating thousands of tentative solutions, the running time of the optimizer can become a challenge when the instance size increases. This last fact is precisely the focus in this work, because computing the objective functions requires applying a matrix stiffness method [6], and making this iteratively can lead easily to weeks and months of computing time.

The main goal of this work is to address this issue by using parallel computing platforms to speedup the search of several multi-objective metaheuristics [7]. We have developed distributed master/worker versions of four multi-objective solvers, namely NSGA-II [8], SMS-EMOA [9], MOCell [10], and SMPSO [11]. These algorithms are a representative sample of the state-of-the-art that includes classical algorithms (NSGA-II), plus recent proposals with different enhanced search features: SMS-EMOA is an indicator-based MOEA, MOCell is a cellular MOEA with a structured population, and SMPSO is a swarm-based multi-objective metaheuristic. They have been engineered for evaluating in parallel the objective functions of two real-world structural design problems. The parallelization devised is aimed at taking full advantage of a massive parallel computing platform but without setting the algorithms with non-customary operating conditions (e.g., extremely large populations for using a large number of workers). The idea is to break down the synchronism induced by the metaheuristic search loop and the configured population size to incorporate a many workers as possible. The experimentation conducted is in the line of showing that the proposed parallelization can incorporate hundreds of processing elements (workers) without relevant impact on the final solution quality. The two real-world problems addressed model two variants of a cable-strayed bridge, which is characterized by having two towers from which steel cables hold the bridge deck. These problem instances have been named 133N_221B and 837N_1584B, reflecting the number of nodes (133 and 837, respectively) and bars (221 and 1584, respectively) of the structure. For the former instance, which is considerably smaller than the latter, an extensive and thorough experimentation with different parallel settings has been conducted to assess our working hypothesis; the latter is then used as a case study using the obtained findings.

Thanks to the adopted parallel approaches, we have been able to optimize the 837N_1584B instance in around 10 hours using more than 400 workers. This paper is an extension of [12], which was presented at the *Second International Workshop on Soft Computing Techniques in Cluster and Grid Computing Systems* (SCCG 2013). That conference paper introduced the distributed versions of SMS-EMOA and NSGA-II, and tackled the 837N_1584B instance. The novel scientific contributions of this extension, built upon such conference paper, are:

- To the best of our knowledge, it is the first master/worker parallelization of MOCell and SMPSO capable to incorporating hundreds of processing nodes.
- The solution of two real-world structural design problems with up to 450 processors.
- A detailed analysis of the quality of the solutions reached by the parallel algorithms when using an increasing number of workers.

In the survey of multi-objective metaheuristics applied to structural problems carried out by Zavala *et al.* [13] some of the final conclusions were that the most widely used algorithm was NSGA-II and that applying parallelism was an open issue; furthermore, most of the addressed problems in the works reviewed in that survey were academic. In this paper, we make progress in these open research lines, by considering modern algorithms (SMS-EMOA, SMPSO, and MOCell), which are parallelized to solve two real-world structural design problems.

The rest of this paper is structured as follows. Section 2 and 3 are devoted to introduce background concepts related to multi-objective optimization, parallelism and multi-objective metaheuristics, respectively. The structural design problems we are dealing with are detailed in Section 4. Section 5 presents both the four algorithms and their parallelization. The obtained results are analyzed in Section 6. Finally, Section 7 summarizes the paper and outlines some lines of further research.

2. BACKGROUND ON MULTI-OBJECTIVE OPTIMIZATION

In this section, we provide some basic background on fundamentals of multi-objective optimization. We start by defining key concepts, such as *multi-objective optimization problem*, *Pareto optimality*, *Pareto dominance*, *Pareto optimal set*, and *Pareto front*, followed by a discussion of the goals of multi-objective optimization. It is assumed in the following, without loss of generality, that all the objective functions are to be minimized.

The definition of a general multi-objective optimization problem (MOP) can be formally formulated as:

Definition 2.1. *(MOP) Find a vector* $\vec{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]$ *which satisfies the* m *inequality con-*

straints $g_i(\vec{x}) \geq 0, i = 1, 2, \ldots, m$, *the p equality constraints* $h_i(\vec{x}) = 0, i = 1, 2, \ldots, p$, *and minimizes the vector function* $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T$, *where* $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ *is the vector of decision variables.*

Those values satisfying the constraints define the *feasible region* Ω and any point $\vec{x} \in \Omega$ is a *feasible solution*. To determine those points that are the optimal solutions to a given MOP, the concept of *Pareto optimality* has is introduced:

DEFINITION 2.2. *(Pareto Optimality) A point* $\vec{x}^* \in \Omega$ *is Pareto Optimal if for every* $\vec{x} \in \Omega$ *and* $I = \{1, 2, \ldots, k\}$ *either* $\forall_{i \in I}(f_i(\vec{x}) = f_i(\vec{x}^*))$ *or there is at least one* $i \in I$ *such that* $f_i(\vec{x}) > f_i(\vec{x}^*)$.

This definition states that $\vec{x}^*$ is Pareto optimal if no feasible vector $\vec{x}$ exists which would improve some criteria without causing a simultaneous worsening in at least one other criterion.

Pareto optimality can be expressed in terms of the concept of *Pareto dominance*:

DEFINITION 2.3. *(Pareto Dominance) A vector* $\vec{u} = (u_1, \ldots, u_k)$ *is said to dominate* $\vec{v}=(v_1, \ldots, v_k)$ *(denoted by* $\vec{u} \preccurlyeq \vec{v}$*) if and only if* $\vec{u}$ *is partially less than* $\vec{v}$*, i.e.,* $\forall i \in \{1, \ldots, k\},\ u_i \leq v_i \ \wedge\ \exists i \in \{1, \ldots, k\} :\ u_i < v_i$.

Thus, a Pareto optimal point is that which is not dominated by any other point in Ω. Pareto dominance allows to compare two solutions, allowing to know if one solution *dominates* another or that both solutions do not dominate each other (i.e., they are *non-dominated* solutions). Many multi-objective algorithms are based on Pareto dominance.

Therefore, the optimization of a MOP involves finding the set of all (or as many as possible) the Pareto optimal solutions. This is the so-called *Pareto optimal set*, or simply Pareto set, and it is defined as follows:

DEFINITION 2.4. *(Pareto Optimal Set) For a given MOP* $\vec{f}(\vec{x})$*, the Pareto optimal set is defined as* $\mathcal{P}^* = \{\vec{x} \in \Omega | \neg\exists \vec{x'} \in \Omega, \vec{f}(\vec{x'}) \preccurlyeq \vec{f}(\vec{x})\}$.

Each vector in the Pareto optimal set has a correspondence in objective function space, leading to the so-called *Pareto front*:

DEFINITION 2.5. *(Pareto Front) For a given MOP* $\vec{f}(\vec{x})$ *and its Pareto optimal set* $\mathcal{P}^*$*, the Pareto front is defined as* $\mathcal{PF}^* = \{\vec{f}(\vec{x}) | \vec{x} \in \mathcal{P}^*\}$.

The main goal of multi-objective optimization algorithms is to obtain the Pareto front of a given MOP. In general, multi-objective optimization problems can have a Pareto front composed by a huge (possibly infinite) number of solutions. When using stochastic techniques, such as metaheuristics, the goal is thus to obtain a *Pareto front approximation* (also called approximation set), i.e., a subset of solutions that represents the optimal Pareto front. The size of this subset (a typical value is 100) has to be large enough to allow the expert to have an accurate set of solutions to choose from. This implies that a Pareto front approximation has to fulfill two properties: (1) *convergence* (i.e., the solutions must be as close as possible to the optimal Pareto front) and (2) *diversity* (i.e., the solutions are uniformly spread along the entire Pareto set, and not only clustered around a few specific parts of it).

Although obtaining Pareto front approximations with good convergence and diversity is important, there is also a very important goal in practice, which is to obtain these approximations in a reasonable amount of time. This issue is tackled in this paper, and it has to do with the fact that running a metaheuristic to solve a real-world optimization problem can take several days (or even weeks or months), making its use impractical. A way to deal with this drawback is to design parallel metaheuristics that are able to take advantage of hundreds of processors/cores.

3. MULTI-OBJECTIVE METAHEURISTICS AND PARALLELISM

Due to their population-based approach, EAs are very suitable for parallelization because their main operations (i.e., crossover, mutation, and, in particular, function evaluation) can be independently performed on different individuals. There is a vast amount of literature on how to parallelize EAs (e.g., see the surveys [14, 15, 16]). However, parallelism here is not only a way to solve problems in shorter times, but also to engineer new and more effective search models: a parallel EA can be more effective than a sequential one, even when executed on a single processor. The advantages that parallelism offers to single objective optimization also hold in multi-objective optimization.

The most well-known models for parallel MOEAs have been directly inherited from the single-objective parallel EA community, where two parallelizing strategies are defined for population-based algorithms: (1) parallelization of computation (Figure 1a), in which the operations commonly applied to each individual are performed in parallel, resulting in the well-known *Master/Slave* or *global parallelization* algorithms; and (2) parallelization of population, in which the population is split into different parts, each one evolving in semi-isolation (individuals can be exchanged between subpopulations). Two main models emerge from this latter parallelization strategy: the *distributed* EA (dEA, or coarse-grain) and *cellular* EA (cEA, fine-grain or diffusion). In dEAs (Figure 1b), the population is partitioned into a set of islands in which isolated EAs run in parallel. Sparse individual exchanges are performed among these islands, aiming at inserting genetic diversity into the subpopulations, thus avoiding them getting stuck in local optima. In the case of cEAs (Figure 1c), subpopulations are

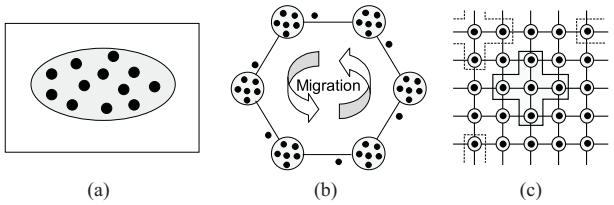


FIGURE 1. Different models of parallel EAs: (a) global parallelization, (b) distributed EAs, and (c) cellular EAs.

typically composed of one individual, which may only interact with its nearby neighbors in the breeding loop as they are arranged in a toroidal grid, i.e., the concept of *neighborhood* is introduced. These neighborhoods are overlapped, which implicitly defines a migration mechanism and allows a smooth diffusion of the solutions throughout the population.

This taxonomy holds as well for parallel MOEAs [17, 18, 19], so we can consider Master/Worker MOEAs (*mwMOEAs*), distributed MOEAs (*dMOEAs*), and cellular MOEAs (*cMOEAs*). Nevertheless, these two decentralized population approaches need a further particularization for MOPs [20]. As we stated before, the main goal of any multi-objective optimization algorithm is to find the optimal Pareto front for a given MOP. It is clear that in msMOEAs, the management of this Pareto front is done by the master processor. But, when the search process is distributed among different subalgorithms, as happens in dMOEAs and cMOEAs, the management of the nondominated set of solutions during the optimization procedure becomes a capital issue. Hence, it can be distinguished when the Pareto front is distributed and locally managed by each subEA during the computation, or it is a centralized element of the algorithm. They have been called *Centralized Pareto Front* (CPF) structured MOEAs and *Distributed Pareto Front* (DPF) structured MOEAs, respectively [7]. A detailed review or parallel MOEAs is outside the scope of this paper, but interested readers are referred to [21].

After providing the reader with a short overview of parallel evolutionary multi-objective optimization, we will now focus on the work that is closely related to the algorithmic proposals presented here. Because of the computationally expensive cost of running a MOEA over thousands of evaluations on the considered target problems, the approach used here is to propose and analyze several master/worker parallelizations of several state-of-the-art MOEAs for which, to the best of our knowledge, such parallel versions do not exist. Indeed, this work introduces the first master/worker parallelization of MOCell and SMPSO (the contribution in the conference paper was the master/slave parallelization of SMS-EMOA). From the point of view of parallelism, we have modified the evolutionary loop of these MOEAs so that they are capable of incorporating as many workers as possible in the computation. In the experiments performed, up to 450 cores from a heterogeneous distributed computing platform successfully evaluate the fitness function in parallel. In order for the workers to be deployed, we have relied on the Condor high throughput computing system [22].

4. STRUCTURAL DESIGN PROBLEM: CABLE-STRAYED 3D BRIDGES

The target problem of our work is the design of two cable-strayed 3D bridges, formed by spatial bars having different cross-section shapes and sizes. The bridge design problem is formulated as a bi-objective optimization problem. The first objective is to minimize the total weight of the structure; the second objective is to minimize the summation of the deformations in some selected nodes. Each bar can have one out of three different cross-sections: hollowed rectangle, I-beam, or circular. Each kind of bar is featured by a set of measurements, which lead to the encoding that will be used by the metaheuristic algorithms. In particular, a solution to the problem is an array of values (one per bar) featuring a concrete instance of the bridge (see Section 6.2).

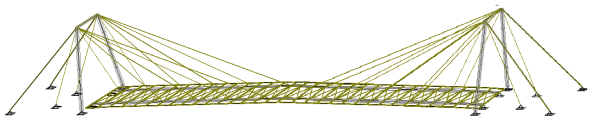


FIGURE 2. Bridge 133N_221B.

We detail next the features of the two bridges considered in our study. The smallest bridge, the 133N_221B instance, has two independent pillars that support part of the weight of the deck, two independent pillars that support part of the weight of the board, and the rest of the board weight of the board is supported by the parallel tension of the cables. It has a one-way roadway and a pedestrian circulation lane which is illustrated in Fig. 2. The bridge has a total length of $44.00m$ and the deck length and width are $32.00m$ and $6.40m$, respectively. In addition, the bridge contains two pillars (towers) of $6.00m$ height anchored by four cables with an upper beam. The main longitudinal beam, made of malleable steel, is suspended using ten high resistant tensioners; therefore, two materials having different elastic properties are used. From a mechanical point of view we only consider one half of the bridge and symmetric loads. The bridge is composed of 133 nodes and 221 bars. For the sake of simplifying the construction, we have congured groups of bars, in such a way that the elements composing the groups have the same shape, same material, equivalent position in the structure, and taking into account the inner forces that might appear in the structure must keep smooth transitions between groups in the geometric continuity of the structure; the 133N_221B bride has 33 groups of

Shape	Bars	Groups (gr)	Variables (var)	Geometric constraints	Mechanic constraints	Deflection constraints
Circle	46	8	1var/gr x 8gr = 8		3const/gr x 8gr = 24	
I-beam	39	4	4var/gr x 4gr = 16	4const/gr x 4gr = 16	3const/gr x 4gr = 12	68
Hollowed rectangle	136	21	4var/gr x 21gr = 84	4const/gr x 21gr = 84	3cont/gr x 21gr = 63	
Total	221	33	108	267		
$F_1 = \Sigma_{i=1}^{b}\gamma_i l_i \Omega_i$, $F_2 = \Sigma_{j=1}^{n}\delta_j$						

TABLE 1. Variables and constraints of the 133N_221B bridge.

bars. After applying this grouping strategy, the total number of decision variables is 108, the total number of geometric constraints is 100, and the sum of mechanical and deflection constraints are 99 and 68, respectively (see Table 1).

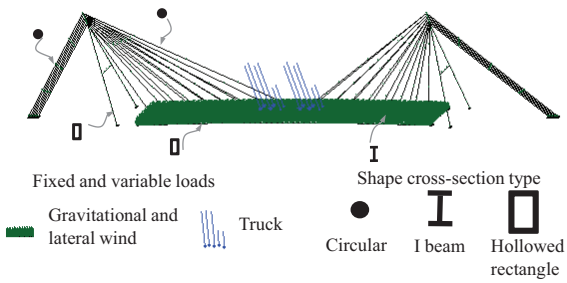


FIGURE 3. Bridge 837N_1584B.

The second problem, referred to as 837N_1584B, has two pillars standing together in the middle, and the tension cables are arranged diagonally. It has a two-way roadway and a pedestrian circulation lane which is illustrated in Fig. 3. The bridge has a total length of $162.00m$ and the deck length and width are $90.00m$ and $9.00m$, respectively. In addition, the bridge contains two pillars of $23.00m$ of height joined in the top and anchored by cables. The main longitudinal beam, made of malleable steel, is suspended using ten high resistant tensioners; therefore, two materials having different elastic properties are used. From a mechanical point of view, we consider the complete bridge and asymmetric loads. The 837N_1584B bridge is composed of 837 nodes, 1 584 bars, and 60 groups of bars. After applying the grouping strategy, the total number of decision variables is 207, the total number of geometric constraints is 392, and the sum of mechanical and deflection constraints are 180 and 16, respectively (see Table 2).

In both problem instances, the two objectives are to minimize the total weight of the structure (in meganewtons or MN) and the sum of deformations of selected nodes (in meters). The formulation of these objectives is as follows:

- $F_1 = \Sigma_{i=1}^{b}\gamma_i l_i \Omega_i$.
- $F_2 = \Sigma_{j=1}^{n}\delta_j$.

where:

- σ_i: stress calculation for i^{th} bar.
- γ_i: specific weight of the material for i^{th} bar.
- l_i: bar length.
- Ω_i: optimized cross-section of the i^{th} bar.
- δ_j: deflection in selected node j.

5. PARALLEL PROPOSALS

In this section, we first introduce briefly the four state-of-the art solvers, NSGA-II, SMS-EMOA, MOCell, and SMSPO. Then, the common parallelization framework used to engineer the parallel versions of these algorithms is described. Finally, we describe how the parallel algorithms are designed within such framework.

5.1. Sequential MO solvers

In this section, we briefly describe the four metaheuristics used in this study, namely NSGA-II, SMS-EMOA, MOCell, and SMPSO. They all are population-based metaheuristics [4], i.e., they operate on a set of solutions at each iteration. A general template for a multi-objective metaheuristic is displayed in Algorithm 1. The general operation of these algorithms begins by generating a set of initial solutions, S (which is referred to as *population* in EAs and as *swarm* in PSO algorithms), and creating a set A to store non-dominated solutions which, at the beginning of the execution, is empty (lines 1 and 2). This set can be used in an implicit way (as in NSGA-II and SMS-EMOA) or explicitly (as in MOCell and SMPSO); in the latter case, it is usually referred to as an *external archive*. The solutions in S are evaluated and the set A is updated (lines 3 and 4). Then, the search loop starts. The loops involves a stochastic variation of the solutions included in S and

Algorithm 1 Template of a generic multi-objective metaheuristic.

1: $S(0) \leftarrow$ GenerateInitialSolutions()
2: $A(0) \leftarrow \emptyset$
3: Evaluation(S)
4: $A(0) \leftarrow$ Update($A(0)$, $S(0)$)
5: $t \leftarrow 0$
6: **while not** StoppingCriterion( ) **do**
7: $t \leftarrow t + 1$
8: $S(t) \leftarrow$ Variation($A(t-1)$, $S(t-1)$)
9: Evaluate($S(t)$)
10: $A(t) \leftarrow$ Update($A(t)$, $S(t)$)
11: **end while**
12: **Output:** A

Shape	Bars	Groups (gr)	Variables (var)	Geometric constraints	Mechanic constraints	Deflection constraints
Circle	42	11	1var/gr x 11gr = 11		3const/gr x 11gr = 33	
I-beam	252	41	4var/gr x 41gr = 164	4const/gr x 41gr = 164	3const/gr x 41gr = 123	16
Hollowed rectangle	1 280	8	4var/gr x 8gr = 32	4const/gr x 8gr = 32	3cont/gr x 8gr = 24	
Total	1 584	23	207	392		
$F_1 = \Sigma_{i=1}^{b} \gamma_i l_i \Omega_i$, $F_2 = \Sigma_{j=1}^{n} \delta_j$						

TABLE 2. Variables and constraints of the 837N_1584B bridge.

A, and the generation of a new set of solutions (line 8) from which those that are non-dominated are retrieved (line 10). The matching of this general scheme on the four algorithms used in this work is briefly presented in the following subsections (for a detailed description, interested readers are referred to the references provided for each one).

5.1.1. NSGA-II

The Non-Dominated Sorting Genetic Algorithm II, NSGA-II, was proposed by Deb *et al.* [8]. It is a genetic algorithm based on generating a new population from the original one by applying the typical genetic operators (selection, crossover, and mutation); then, the individuals in the new and old population are sorted according to their rank, and the best solutions are chosen to create a new population. In case of having to select some individuals with the same rank, a density estimation based on measuring the crowding distance to the surrounding individuals belonging to the same rank is used to get the most promising solutions. From Algorithm 1, S and A are considered to be one single set $P = S \cup A$ so that, at each iteration, the non-dominated solutions found are used to generate new solutions within the evolutionary loop.

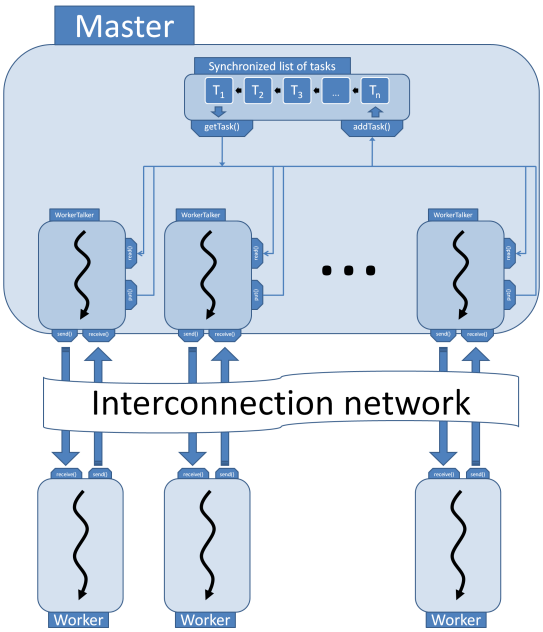


FIGURE 4. Underlying parallel software architecture for both SMS-EMOA and NSGA-II

5.1.2. SMS-EMOA

The general idea of SMS-EMOA is to use a quality indicator to guide the search of the algorithm [9]. In other words, the algorithm aims to compute a Pareto front optimizing the value of that quality indicator. SMS-EMOA makes use of the Hypervolume, I_{HV}, which is the only unary performance measure that is known to be Pareto-compliant [23]. It calculates the volume, in objective function space, covered by members of a non-dominated set of solutions. Therefore, the higher the value of I_{HV}, the better will be our approximation of the Pareto front.

SMS-EMOA is based on the NSGA-II algorithm, but introduces two main modifications: firstly, SMS-EMOA is a steady-state evolutionary algorithm, while NSGA-II is a generational one; secondly, instead of using the crowding distance as density estimator, SMS-EMOA considers the contribution of the solutions to I_{HV} in the current approximation of the Pareto front. This way, in every iteration the algorithm discards the solution contributing the least to the hypervolume.

5.1.3. MOCell

The Multi-Objective Cellular Genetic Algorithm, MOCell, is a cellular genetic algorithm (cGA) [10]. Like many multi-objective metaheuristics, it includes an external archive to store the nondominated solutions found so far. The matching with Algorithm 1 is therefore straightforward. The archive is bounded and uses the crowding distance of NSGA-II to maintain diversity in the Pareto Front. We have used here an asynchronous version of MOCell, called aMOCell4 in [24], in which the cells are explored sequentially (asynchronously). The selection is based on taking an individual from the neighborhood of the current solution (called *cell* in cGAs) and another one randomly chosen from the archive. After applying the genetic crossover and mutation operators, the new offspring is compared with the current one, replacing it if better; if both solutions are non-dominated, the worst individual in the neighborhood is replaced by the current one. In these two cases, the new individual is inserted into the archive.

5.1.4. SMPSO

SMPSO (Speed-constrained Multi-Objective PSO algorithm) is a particle swarm optimization algorithm for solving MOPs [11]. From a high level of abstraction, in a PSO algorithm, a set (swarm) of candidate solutions (particles) to the problem navigate through the search space of an optimization problem. This navigation takes place attending to a velocity equation, which rules the way in which particles change their position. Among the factors that govern that velocity equation, two of them can be highlighted: the current position of the particle and the best positions visited so far, also referred to as leaders. Usually, the best position visited by a particle (local leader) and the best particle visited by any particle in the swarm (global leader) are considered. The main innovation of SMPSO is the incorporation of a constraining mechanism already applied in single-objective PSO algorithms, which modulates the speed at which particles fly [25]. SMPSO uses an external archive to store the non-dominated solutions, so it fits into the template described in Algorithm 1.

5.2. Parallelization framework

The general architecture used for parallelizing the four MOEAs is outlined in Fig. 4. The idea is to follow a master/worker distributed scheme. As can be seen, a multi-threaded master has been devised where there is a talker thread handling the communication with each worker. Talkers and workers communicate via tasks, which are just containers of tentative solutions to be evaluated remotely. Tasks are stored in a shared FIFO list which is accessed concurrently by all the talkers (in mutual exclusion). A talker can both, remove and add, tasks to this list. If this list gets empty it means that all the tasks have been processed, i.e., all the solutions have been evaluated by the workers, so the master stops (as well as all the workers) because the algorithm has finished.

Algorithm 2 Pseudo-code of the master thread

```
1: population ← GenerateInitialPopulation()
2: taskList ← addTasks(population)
3: threads ← runTalkerThreads(population, taskList)
4: waitForAllThreadsToComplete(threads)
```

The mapping strategy of NSGA-II, SMS-EMOA, MOCell, and SMPSO follows common guidelines which are described next, and the particular design decisions adopted for each algorithm are detailed in the corresponding subsections. Initially, the master thread randomly generates as many parallel tasks (enclosing the initial solutions of the algorithm) as the population (or swarm) size, aiming at evaluating the initial set of solutions in parallel (lines 1 and 2 in Algorithm 2). That is, this set of tasks are inserted into the task list, which means that are ready to be remotely evaluated by a worker. The processing is then delegated to the talker threads, which handle the communication with the workers concurrently (thus, the master can profit from multi-core computers). Talkers pick up tasks from the list, send them to the workers, and wait for the evaluated individual to be returned. The operation in the workers is fairly simple: upon reception of a task, the enclosed solution is retrieved, evaluated, and sent back to the master (a pseudo-code is shown in Algorithm 3).

Algorithm 3 Pseudo-code of a worker

```
1: while (not receive finalization notification) do
2:    task ← receiveFromTalker()
3:    solution ← task.solution
4:    evaluate(solution)
5:    newTask ← new Task(solution)
6:    sendToTalker(newTask)
7: end while
```

The cornerstone of the parallel implementation relies on breaking down the synchronization requirements imposed by the evolutionary loop of the sequential algorithms, which are determined by two main factors: the number of solutions manipulated (i.e., the population size) and the selection scheme (either generational or steady-state) [26]. Let's assume that the corresponding sequential MOEA has a population size of P. In the generational selection scheme, P new solutions are generated within the evolutionary loop (by means of the genetic operators). This means that, at most, P solutions have to be evaluated in parallel at each iteration, so only P workers are required. In the steady-state case, the scenario is even worse for an efficient parallelization, as only one single solution is generated at each iteration. Then a hard sync point is reached: the evolutionary loop does not proceed to the next generation until all the P solutions are evaluated. In a parallel master/worker setting this means that the master has to wait for the slower worker to start sending the newly generated individuals of the next generation for remote evaluation, leaving the faster ones idle. In case of parallel heterogeneous platforms, this issue has a deep impact on the parallel performance of the resulting algorithm.

The proposed parallelization deals with these two previous issues in such a way that its efficiency does not depend either on the population size or on the selection scheme. It is based on relaxing two main operational aspects within the evolutionary loop: on the one hand, the population incorporates any newly incoming solution evaluated remotely, regardless of when it was generated, i.e., at iteration t, the evolutionary loop could be receiving solutions generated at iteration $t - k, k > 1$ (e.g., by very slow workers); on the other hand, it generates more solutions than the value imposed by the population size or the selection scheme. All this is done in the talker threads as shown in Algorithm 4. In order to deal with the first

relaxation, talkers extract the first task of the master's task list (line 3), send it out to its corresponding worker for remote evaluation (line 4), and wait for the evaluated solution (line 5). This newly incoming solution is inserted into an auxiliary population (no matter when it was added to the tasklist). The size of this auxiliary population, $auxS$, allows to define the selection strategy: if $maxAuxSize = 1$ (line 8), a steady-state scheme is adopted; on the other hand, if $maxAuxSize = auxS.size()$, a generational selection scheme emerges. When $auxS$ is filled (i.e., the algorithm already has enough solutions evaluated to proceed with a new iteration), the archive with the non-dominated solutions is updated by using the previous contents of the archive, the current MOEA population, and the auxiliary population recently evaluated by the workers (line 10). Then, the talker threads deal with the second relaxation: talker threads will be generating solutions, and packaging them into tasks for remote evaluation by a worker, while the size of the task list is lower than the number of workers involved in the parallel computation (lines 11 to 15). The aim is to generate enough workload to keep the workers busy, thus increasing the parallel performance.

Algorithm 4 Pseudo-code of a talker thread

```
Require: S // the current MOEA population
Require: A // the current MOEA archive
Require: taskList // the tasks to be remotely executed
 1: auxS ← ∅
 2: while (not stopping condition is met) do
 3:    task ← getTask(taskList)
 4:    sendToWorker(task)
 5:    processedTask ← receiveFromWorker()
 6:    s ← processedTaks.solution
 7:    auxS ← auxS.add(s)
 8:    if (auxS.size() == maxAuxSize) then
 9:       t ← t + 1
10:       A(t) ← Update(A(t − 1), S(t − 1), auxS)
11:       if (taskList.size ≤ numWorkers) then
12:          S(t) ← Variation(A(t), S(t − 1))
13:          newTasks ← generateTasks(S(t))
14:          taksList.add(newTasks)
15:       end if
16:       auxS ← ∅
17:    end if
18: end while
```

5.3. Parallel algorithms

The next subsections describe the way in which the four MOEAs used in this work are mapped on a generic parallel framework. The parallel versions are named by adding the prefix "mw" to the names of the algorithms.

5.3.1. mwNSGA-II

This algorithm fits perfectly on the general outline presented in Algorithm 4 by just making $maxAuxSize$ equals to the population size of the population. Indeed, NSGA-II is a generational MOEA. The external archive A and the population S merge into a single population of solutions that is handled by the $Update()$ function. The NSGA-II parallelization developed here follows the same structure as the asynchronous generational NSGA-II version presented in [27].

5.3.2. mwSMS-EMOA

SMS-EMOA is a steady-state algorithm. Within the general framework presented above, this can be mapped by just making $maxAuxSize = 1$. Then, whenever a newly evaluated solution arrives, $auxS$ is completed, and the algorithm starts processing it further by measuring its contribution to the hypervolumen indicator and re-computing the contributions of the solutions already in the population. A ranking procedure is applied to $S \cup auxS$ and the solution that contribute the least to the hypervolume is removed so as to keep the population size constant.

5.3.3. mwMOCell

The parallelization of MOCell has required several tricky steps as the population of the algorithm is structured. That is, newly incoming solutions must be inserted in the corresponding grid position so as to maintain the smooth diffusion of genetic material throughout the grid as similar as possible to that of the sequential MOCell. The extra workload to keep workers busy is generated by traversing the structured population randomly, trying to avoid evaluating more solutions from specific areas that may lead to very different evolved solutions.

5.3.4. mwSMPSO

In mwSMPSO, we may just rename "population" with "swarm" and "archive" with "leaders", and the mapping onto the general parallel framework is direct as SMPSO does have an external archive, and the swarm uses a generational scheme. As such, whenever the auxiliary swarm is filled, the new particle speeds and positions are computed. The mutated particles are then packaged into a task for remote evaluation in the workers.

5.4. Parallel Computing Platform

All the algorithms and the distributed computing platform have been developed in Java. In particular, the jMetal framework for multi-objective optimization with metaheuristics [28] has been used to provide the sequential versions of the four selected metaheuristics. These algorithms have been adapted to use the master/worker architecture. For the deployment of the workers we have used the Condor system [22], which helps in simplifying the finding of idle computers.

The parallel computing system we have used is composed by the computers of the teaching labs of the Department of Computer Science at the University of

Málaga. We have been able to use up to 192 machines with two cores, 8157 of RAM, Windows 7 (64 bits), which are only available from 10:00PM to 8:00AM during the working days, and the entire weekends. The interconnection network is a 10GB ethernet.

6. EXPERIMENTATION

6.1. Methodology

We are evaluating the proposed parallel MOEAs in two separate dimensions: their parallel performance and the quality of the approximated Pareto fronts that they produce. The former is basically achieved by looking at the execution time when an increasing number of workers are involved in the parallel computation, whereas the latter is assessed by measuring two quality indicators: the hypervolume (HV) [23] and the attainment surfaces [29].

The HV is considered as one of the more suitable performance indicators in the multi-objective community since it provides a measure that takes into account both the convergence and the maximum spread of the obtained approximation set. Higher values of the hypervolume are desirable. Since this indicator is not free from an arbitrary scaling of the objectives, we have built up a reference Pareto front (RPF) for each problem composed of all the nondominated solutions found for each problem instance by all the algorithms. Then, the RPF is used to normalize each approximation prior to computing the HV value by mapping all the nondominated solutions to $[0, 1]$. This way, the reference point to compute the HV values is (1,1), which results from the mapping of the extreme solutions of the RPF.

While the HV allows one to numerically compare different algorithms, from the point of view of a decision maker, knowing about the HV value might not be enough, because it gives no information about the shape of the front. The empirical attainment function (EAF) [29] has been defined to do this. EAF graphically displays the expected performance and its variability over multiple runs of a multi-objective algorithm. In short, the EAF is a function α from the objective space $\mathbb{R}^n$ to the interval $[0, 1]$ that estimates for each vector in the objective space the probability of being dominated by the approximated Pareto front of one single run of the multi-objective algorithm. Given the r approximated Pareto fronts obtained in the different runs, the EAF is defined as:

$$\alpha(z) = \frac{1}{r} \sum_{i=1}^{r} I(A^i \preccurlyeq \{z\}) \qquad (1)$$

where A^i is the i-th approximated Pareto front obtained with the multi-objective algorithm and I is an indicator function that takes a value of 1 when the predicate inside it is true, and of zero, otherwise. The predicate $A^i \preccurlyeq \{z\}$ means that A^i dominates solution z. Thanks to the attainment function, it is possible to define the concept of k%-attainment surface [29]. The attainment function α is a scalar field in $\mathbb{R}^n$ and the k%-attainment surface is the level curve with value $k/100$ for α. Informally, the 50%-attainment surface in the multi-objective domain is analogous to the median in the single-objective one and it is the default value used in the following sections.

As metaheuristics are stochastic algorithms, their results must be provided with statistical confidence ([30]). To cope with this matter, 30 independent runs have been carried out. Then, a Kolmogorov-Smirnov test is performed in order to check whether the samples are distributed according to a normal distribution or not. If so, an ANOVA I test is performed; otherwise a Kruskal-Wallis test is performed. Since more than two algorithms are involved in the study, a post-hoc testing phase which allows for multiple comparisons of samples (multicompare) has been conducted. All the statistical tests are performed with a confidence level of 95%.

6.2. Solution encoding, genetic operators, and parameterization

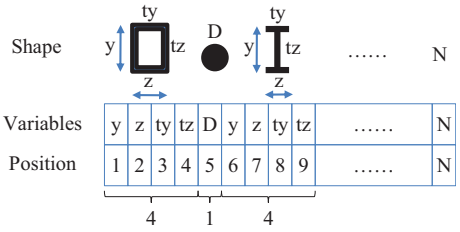


FIGURE 5. Encoding of solutions.

The encoding used to represent the solutions is depicted in Fig. 5. Each bar, depending on their cross-section shape, has a number of parameters to be optimized; they are then grouped as detailed in Tables 1 and 2 and, finally, the parameters for each group (diameter, height, etc,.), which are real-valued numbers, are all arranged in a vector. With such a representation, we adopted simulated binary crossover (SBX) and polynomial-based mutation, which have been frequently used in the multi-objective optimization community.

The detailed settings for each of the four algorithms are included in Table 3. All of them are configured to obtain 100 non-dominated feasible solutions at most (those solutions not fulfilling the problem constraints are discarded). Finally, we want to clarify two relevant points. On the one hand, we have not paid attention to the particular parameterization of the algorithms as we have used the standard values given in the seminal works in which they were presented. On the other hand, we want to remark again that the comparison is fair in terms of both the numerical performance (i.e., the size of the sampling in the search space) and the maximum

[!t]

TABLE 3. Parameterization of the algorithms. L is the individual length.

Parameterization used in mwNSGA-II	
Population Size	100 individuals
Selection of Parents	binary tournament + binary tournament
Recombination	simulated binary, $p_c = 0.9$
Mutation	polynomial, $p_m = 1.0/L$
Parameterization used in mwSMS-EMOA	
Population Size	100 individual
Selection of Parents	binary tournament + binary tournament
Recombination	simulated binary, $p_c = 0.9$
Mutation	polynomial, $p_m = 1.0/L$
Archive Size	100
Parameterization used in mwMOCell	
Population Size	100 individuals (10×10)
Neighborhood	1-hop neighbors (8 surrounding solutions)
Selection of Parents	binary tournament + binary tournament
Recombination	simulated binary, $p_c = 0.9$
Mutation	polynomial, $p_m = 1.0/L$
Archive Size	100 individuals
Parameterization used in mwSMPSO	
Swarm Size	100 individuals
Leaders Size	100 individuals
Mutation	polynomial, $p_m = 1.0/L$

TABLE 4. Mean and standard deviation of the execution time for the four MOEAs and the 133N_221B instance when using 100, 200, and 300 workers.

	Number of workers		
	100	200	300
MOEA	$\mu_{\pm sigma_n}$	$\mu_{\pm sigma_n}$	$\mu_{\pm sigma_n}$
mwNSGAII	$1015.57_{\pm 10.30}$	$943.32_{\pm 7.22}$	$946.23_{\pm 23.77}$
mwSMSEMOA	$1030.29_{\pm 24.73}$	$952.15_{\pm 12.68}$	$948.10_{\pm 24.44}$
mwMOCell	$1017.90_{\pm 5.64}$	$948.64_{\pm 19.66}$	$940.24_{\pm 6.37}$
mwSMPSO	$1019.69_{\pm 11.25}$	$942.80_{\pm 8.11}$	$941.07_{\pm 7.40}$

size of the approximated fronts (i.e., no algorithm is given more chance to cover regions of the Pareto front by using non-dominated sets of unbounded size).

6.3. Scalability analysis

This section discusses the scalability of the parallelization proposed for the four MOEAs in terms of the number of workers that may be involved in the parallel computation at the same time. The experimental conditions vary depending on the instance considered: 150,000 function evaluations are performed for 133N_221B (the smaller one), and only 1,000 for 837N_1584B (the larger one). We were forced to reduce drastically the exploration of the search space in the latter, because of the extremely long time required to evaluate one single solution and the intermittent availability of the computing platform (only during nights and weekends), which would have make the experiments to be unaffordable for the given timeframe. The aim is just to illustrate the parallel performance that can be reached.

Table 4 shows the mean, μ, and standard deviation, σ_n, of the runtime of the four MOEAs for the 133N_221B instance when using 100, 200, and 300 workers in the computation. The results show that the size of this instance is not large enough to make the ratio computation/communication favorable and, as a consequence, the runtime is slightly reduced when using more workers. Indeed, the algorithms take 1020.86 seconds on average when using 100 workers, and 946.72 and 943.91 seconds with 200 and 300 workers, respectively. This means a reduction of 7.26% and 7.54% in spite of using twice and three times more computational power. These are, however, expected results. We will provide this fact with a concise explanation below, using the stats reported by the Condor system. But let us now illustrate the benefits of the parallelization proposed for four MOEAs by analyzing the results of the 837N_1584B instance, which are displayed in Table 5. The parallel performance has increased dramatically: from the 878.35 seconds required on average by the four MOEAs with 100 workers, the runtime has been reduced to 481.12 and 317.79 seconds when using 200 and 300 workers, respectively. That is, doubling and tripling the computational resources has allowed the parallel algorithms to reduce the computational time to almost to a half (45.22%) and a third (63.82%). We can claim that the proposed parallel MOEAs can therefore take full advantage of an increasing number of workers if computing the fitness function takes long enough.

TABLE 5. Mean and standard deviation of the execution time for the four MOEAs and the 837N_1584B instance when using 100, 200, and 300 workers.

	Number of workers		
	100	200	300
MOEA	$\mu_{\pm sigma_n}$	$\mu_{\pm sigma_n}$	$\mu_{\pm sigma_n}$
mwNSGA-II	$863.00_{\pm 39.69}$	$492.60_{\pm 10.07}$	$313.50_{\pm 6.69}$
mwSMS-EMOA	$874.60_{\pm 12.04}$	$488.40_{\pm 9.83}$	$320.40_{\pm 9.24}$
mwMOCell	$894.60_{\pm 43.78}$	$496.00_{\pm 6.26}$	$320.60_{\pm 9.24}$
mwSMPSO	$881.20_{\pm 18.60}$	$447.50_{\pm 40.50}$	$316.66_{\pm 8.01}$

Table 6, which includes the average utility of the workers reported by the Condor system, may help to better explain the previous claims. The utility is computed as the time the workers have been evaluating solutions for the master divided by the total time they have been involved in the computation (i.e., allocated by Condor). In other words, it is the percentage of time they have not been idle. It can be seen that the average utility of the workers for the 133N_221B instance (upper part of the table) is, at most, 72.81% (averaging over the four MOEAs) when using 100 workers, and decreases rapidly to 32.79% and 19.63% for 200 and 300 workers, respectively. That is, the master is not able to provide such a number of workers with enough workload because solutions are arriving very quickly and the network bandwidth is limited (10 GB Ethernet). For the 837N_1584B instance, the utility values are always over 81%, even when the parallel MOEA uses 300 workers, reaching a 96.62% on average

TABLE 6. Average utility reported by the Condor system for the two instances when using 100, 200, and 300 workers.

		Number of workers		
Instance	MOEA	100	200	300
133N_221B	msNSGA-II	72.84%	32.09%	19.61%
	msSMS-EMOA	71.40%	32.92%	19.72%
	msMOCell	73.24%	32.67%	19.62%
	msSMPSO	73.76%	33.50%	19.60%
837N_1584B	msNSGA-II	95.96%	87.10%	82.17%
	msSMS-EMOA	96.88%	87.57%	81.87%
	msMOCell	96.80%	86.90%	81.55%
	msSMPSO	96.86%	86.07%	81.72%

TABLE 7. Mean and standard deviation of the HV indicator for the four parallel MOEAs when running with 100, 200, and 300 workers (133N_221B instance).

	Number of workers		
	100	200	300
MOEA	$\mu_{\pm sigma_n}$	$\mu_{\pm sigma_n}$	$\mu_{\pm sigma_n}$
mwNSGA-II	$0.864_{\pm 0.027}$	$0.800_{\pm 0.043}$	$0.736_{\pm 0.051}$
mwSMS-EMOA	$0.792_{\pm 0.049}$	$0.746_{\pm 0.066}$	$0.708_{\pm 0.045}$
mwMOCell	$0.643_{\pm 0.083}$	$0.601_{\pm 0.074}$	$0.553_{\pm 0.094}$
mwSMPSO	$0.001_{\pm 0.004}$	$0.001_{\pm 0.005}$	$0.000_{\pm 0.000}$

for 100 workers. We want to remark here that such utility values have been reached by performing only 1,000 function evaluations, from which the very first 100 are computed to fill the initial population, and the parallelization proposed only uses 100 workers. That is, when using 200 and 300 workers, there are, respectively, 100 and 200 workers that remain idle until the initial population is evaluated. As we will show below in Section 6.5, where this instance is thoroughly analyzed, longer runs improve this utility value.

6.4. Numerical results

This section analyzes the quality of the approximated Pareto fronts reached by the four parallel MOEAs in terms of the HV indicator, only for the 133N_221B instance. We wanted to evaluate the impact of involving a higher number of workers in the parallel computation. The results are included in Table 7. Recall here that, for the HV indicator, the higher the value, the better it is.

The first clear conclusion is that using a lower number of workers (100) has allowed all the four MOEAs to reach approximated Pareto fronts with higher HV values. The statistical analysis performed (see Figure 6) confirms this claim with statistical confidence in many cases. The stats output is shown in tabular form, as a head-to-head comparison between pairs of <Algorithm>.<Number of workers>; a black upwards triangle states that the setting of the row has statistically higher values than the configuration of the column, a white downwards triangle states that the configuration in the row has statistically lower values than the configuration in the column. When no

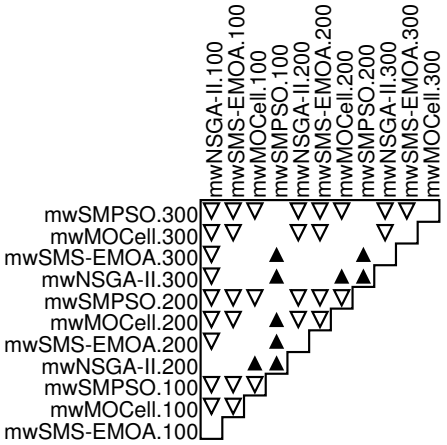
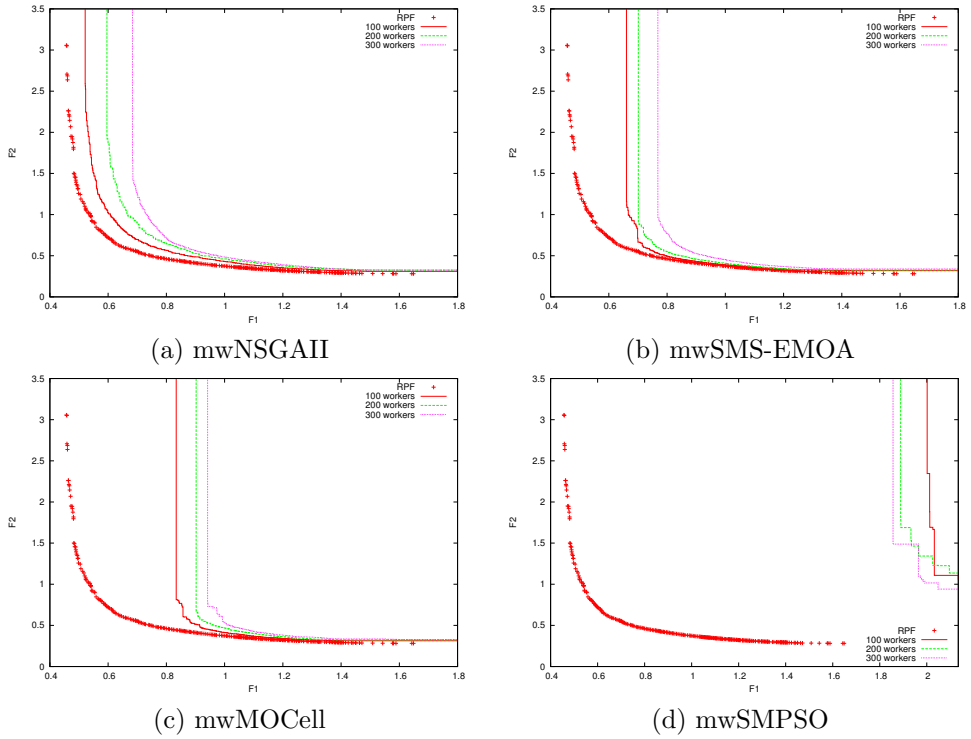


FIGURE 6. Result of the statistical test

statistically significant differences are found, the spot is left empty. The 50%-attainment functions included in Figure 7 illustrate this fact. It can be seen how the average surface covered by the algorithms with 100 workers dominates both that of 200 and 300 workers, specially in the left hand side of the approximated fronts (lower values of F1). This behavior is specially relevant in mwNSGA-II, which is the algorithm that better explores this portion of the search space. We do have a explanation for this issue. It has to do with the design decisions adopted to improve the parallel performance of the algorithms (see Section 5.2): a larger number of workers has a direct impact on the solution diversity, because the algorithms generate more solutions than their sequential counterparts within the evolutionary loop. This is specially critical at the beginning of the execution, until enough workload is generated for all the workers. As it occurs in the early stage of the search, solutions have been scarcely evolved and their offspring may not be of high quality either. This increased diversity is clearly beneficial for longer runs.

Even though the HV is a highly reliable indicator, the 50%-attainment surfaces allow us to extract further advantages of the parallelization proposed. If the decision maker is interested in those regions of the Pareto front for which there are no differences in the algorithms (larger values of F1), then all the algorithms perform the same, regardless of the number of workers. Therefore, massively parallel algorithms could be safely used without compromising the solution quality.

The final analysis is devoted to compare the four MOEAs when the same number of workers is used (a by-column analysis of Table 7). The HV indicator shows that mwNSGA-II is the best performing algorithm for 100, 200, and 300 workers, but with tight differences with respect to mwSMS-EMOA (no statistical confidence exist in Figure 6). MOCell does have competitive results and SMPSO is, clearly, the worst performer. This latter fact has to do with the



(a) mwNSGAII (b) mwSMS-EMOA

(c) mwMOCell (d) mwSMPSO

FIGURE 7. Three 50%-attainment surfaces (one for each number of workers) of the four algorithms for the 133N_221B instance. Each subfigure also plots the reference Pareto front.

constrained features of the problem: it is fairly difficult for all the algorithms to reach the feasible region and, in general, PSO algorithms do not deal with this kind of problems properly. We want to dive a little bit more into these results, supported by the 50%-attainment surfaces displayed in Figure 8 (SMPSO surfaces are not shown due to their bad performance). As explained above, mwNSGA-II has reported the best HV values as it is able to better explore the region of solutions with lower values of the F1 function (weight). However, the central part of the approximated fronts is dominated, specially, by that of mwSMS-EMOA, and in the three parallel settings. An explanation of these differences is that the solutions in the central regions of the fronts usually have a higher contribution to the HV indicator than those in the extreme regions, so SMS-EMOA tends to promote solutions from this central part. The same happens to MOCell, but to a lesser extent.

6.5. Case study: the bridge 837N_1584B

As explained before, the study of this instance has been addressed in a separate section, as we have not been able to provide the rigorous statistical study required in an experimental work like this because the parallel computing platform cannot be used in exclusivity. Assuming this scenario, we decided to increase the sample of the search space from 150000 to 300000 function evaluations so as to show the potential benefits of the parallelization scheme proposed. We have performed one single run for each of the four mwMOEAs developed in this work.

The analysis of the runtime and the parallel performance clearly motivates and supports this work: on average over the four runs, and using up to 420 cores, the wall clock time of the executions is 66112 seconds (18.36 hours), and the accumulated time reported by Condor is 8456 hours (more than 311 days). The utility value reported is, on average, 98.75%, which supports the claims stated above.

The approximated Pareto fronts obtained are displayed in Figure 9 (only those of NSGA-II, SMS-EMOA, and MOCell are included as SMPSO was not able to reach the feasible region). It can be seen that the same conclusions hold as when using the small instance (133N_221B). NSGA-II better explores the search space with lower values for F1 (weight), SMS-EMOA finds solutions that dominate those from NSGA-II and MOCell in the central part of the front and, finally, MOCell seems to have performed better here as it also dominates solutions from NSGA-II in this region.

6.6. Visualization of Solutions

After carrying out the experimental study, the last step is to provide the Pareto fronts approximation to an expert in order to choose some of the solutions. From the point of view of the civil engineer, the most attractive solutions are those having the less weight, so the fronts returned by NSGA-II are the most interesting.

Once a solution is selected, the concrete dimensions

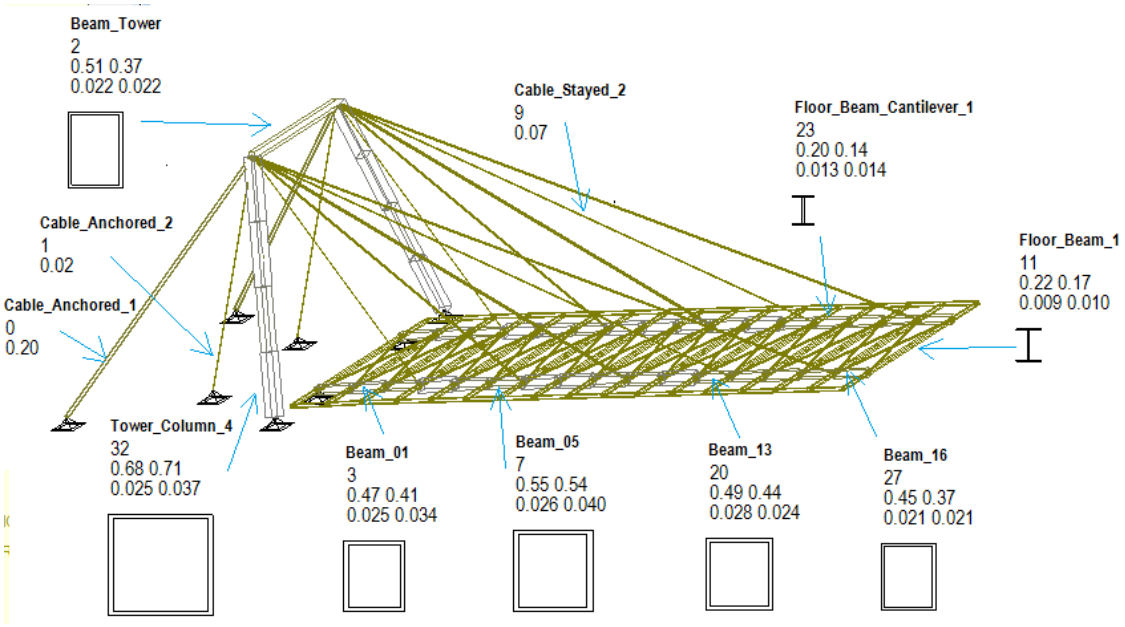


FIGURE 10. Visualization of the bar dimensions of a solution for the 133N_221B problem.

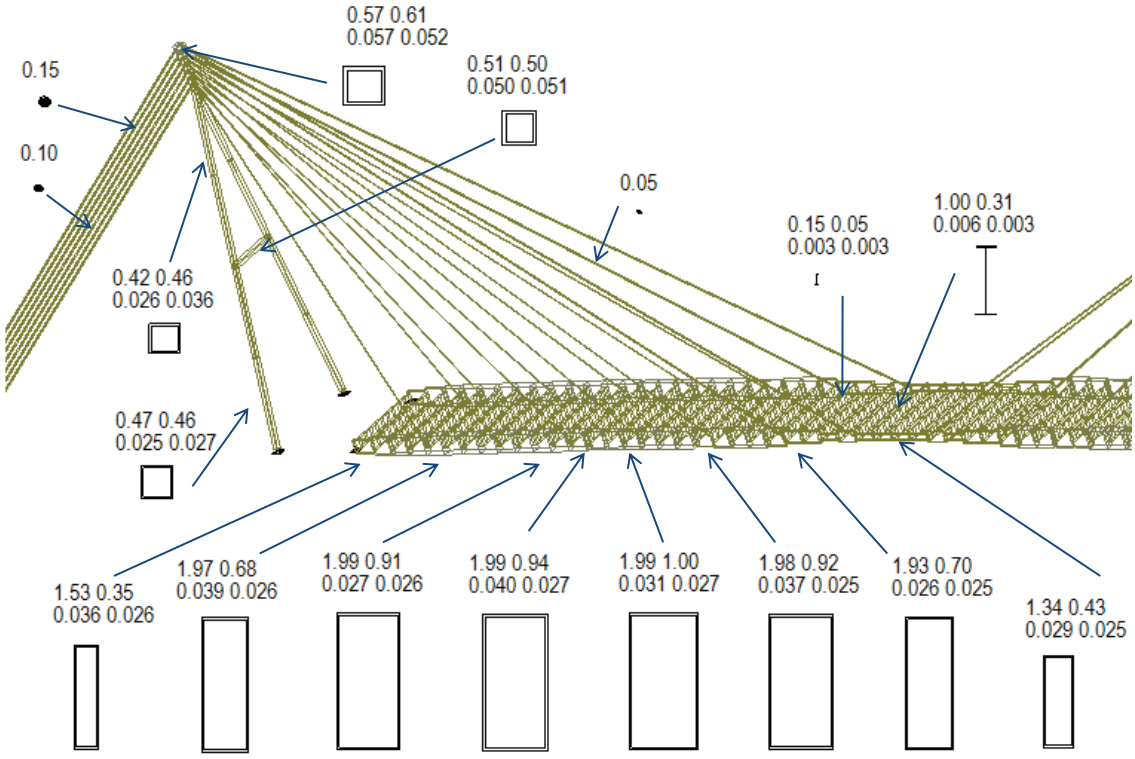
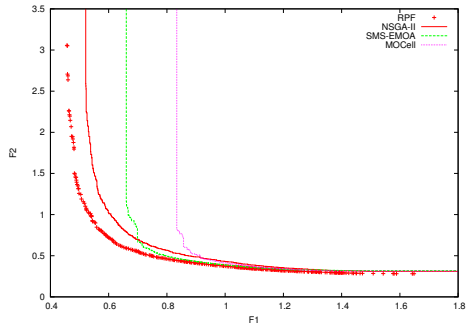


FIGURE 11. Visualization of the bar dimensions of a solution for the 837N_1584B problem.

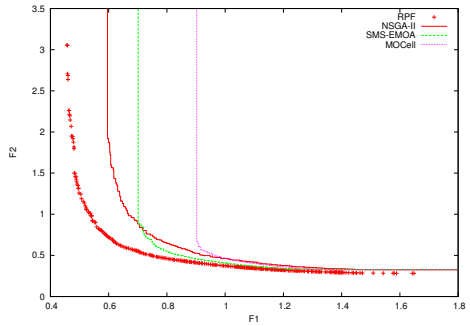
of each bar can be obtained and visualized with a structural design tool, as depicted in Figures 10 and 11. We can observe the particular dimensions of the bars of each bridge.

7. CONCLUSIONS AND FUTURE WORK

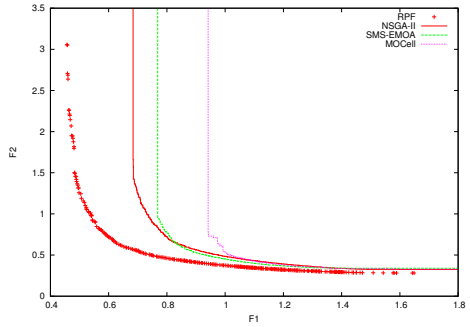
In this paper, we have developed four distributed multi-objective metaheuristics to solve two instances of a real-word structural problem: the design of a cable-strayed bridge. The problem is formulated with two objectives to be minimized at the same time: the total weight



(a) 100 workers



(a) 200 workers



(c) 300 workers

FIGURE 8. 50%-attainment surfaces that compare the MOEAs when using (a) 100, (b) 200, and (c) 300 workers. Each subfigure also plots the reference Pareto front.

and the deformation in concrete points. The two target problems are named 133N_221B and 837N_1584B (i.e., 133 and 837 nodes, 221 and 1584 bars), and they are featured by having two towers supporting a deck, being the total bridge length 44m and 162m, respectively.

Given the multi-objective nature of the bridge design problem, we have addressed its optimization by using four state-of-the-art multi-objective metaheuristics, namely NSGA-II, SMPSO, MOCell, and SMS-EMOA. The estimated time to run the algorithms during 150000 evaluations indicate that about 11 hours are needed in the case of the bridge 133N_221B and more than 170 days in the case of 837N_1584B, so applying parallelism arises as a natural way to reduce these computing times. To take advantage of hundreds of processors/cores, we have applied a master/slave scheme to the sequential metaheuristics, yielding as a result four distributed algorithms, where the master rules the logic of the

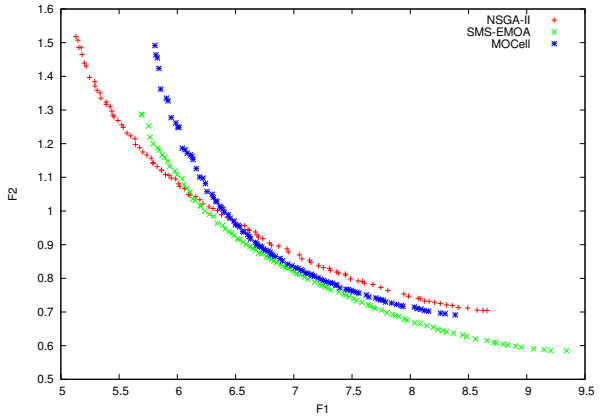


FIGURE 9. Approximated fronts of NSGA-II, SMS-EMOA, and MOCell for 837N_1584B.

metaheuristics and the workers perform the evaluations in parallel. By using the Condor system to deploy the workers, we have been able to use up to 450 cores, which has allowed us to drastically reduce the computing time of the algorithms.

We have made a rigorous experimental comparison of the four distributed algorithms on the smallest bridge, by making 30 independent runs per configuration. Also, we adopted well-stablished quality indicators, and applied tests to ensure the statistical significance of the results. The study reveals that the distributed version of NSGA-II is the best performing algorithm. So, we have used this technique to solve the 837N_1584B instance. The obtained Pareto front approximations have been analyzed by an expert in the field, concluding that the results provide accurate designs to the two bridge design problems.

Future research works include developing distributed versions of other state-of-the-art multi-objective algorithms (e.g., MOEA/D) and applying these techniques to other engineering optimization problems requiring of parallel computing power so that they can be solved in a reasonable amount of time.

ACKNOWLEDGEMENTS

Francisco Luna acknowledges support by the Spanish Ministry of Science and Innovation and FEDER under contract TIN2011-28336. Antonio J. Nebro is supported by Grants TIN2011-25840 (Ministerio de Ciencia e Innovación) and P11-TIC-7529 and P12-TIC-1519 (Plan Andaluz de Investigación, Desarrollo e Innovación). Juan J. Durillo acknowledges the Austrian Research Promotion Agency under contract nr. 834307 (AutoCore). Carlos A. Coello Coello acknowledges support from CONACyT project no. 103570.

REFERENCES

[1] Geem, Z. W. (ed.) (2011) *Optimization in Civil & Environmental Engineering*. Old City Publishing, Inc.,

USA.

[2] Coello, C. A. and Lamont, G. B. (2004) *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, Singapore.

[3] Stewart, T., Bandte, O., Braun, H., Chakraborti, N., Ehrgott, M., Göbelt, M., Jin, Y., Nakayama, H., Poles, S., and Stefano, D. (2008) Real-world applications of multiobjective optimization. In Branke, J., Deb, K., Miettinen, K., and Sowiski, R. (eds.), *Multiobjective Optimization*, Lecture Notes in Computer Science, **5252**, pp. 285–327. Springer Berlin Heidelberg, Germany.

[4] Blum, C. and Roli, A. (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, **35**, 268–308.

[5] Coello Coello, C. A., Lamont, G. B., and Van Veldhuizen, D. A. (2007) *Evolutionary Algorithms for Solving Multi-Objective Problems*, second edition. Springer, New York.

[6] Turner, M., Clough, R. W., Martin, H. C., and Topp, L. J. (Setember 1956) Stiffness and deflection analysis of complex structuures. *Journal of the Aeronautical Sciences (Institute of the Aeronautical Sciences)*, **23**, 805–823.

[7] Nebro, A., Luna, F., Talbi, E.-G., and Alba, E. (2005) Parallel Multiobjective Optimization. *Parallel Metaheuristics*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

[8] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, **6**, 182–197.

[9] Beume, N., Naujoks, B., and Emmerich, M. (2007) Sms-emoa: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, **181**, 1653–1669.

[10] Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., and Alba, E. (2009) Mocell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, **24**, 723 – 725.

[11] Nebro, A. J., Durillo, J. J., Garca-Nieto, J. M., Coello, C. A. C., Luna, F., and Alba, E. (2009) SMPSO: A new PSO-based metaheuristic for multi-objective optimization. *2009 IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making*, Nashville, TN, March 30 - April 2, pp. 66 – 73. IEEE.

[12] Luna, F., Zavala, G. R., Nebro, A. J., Durillo, J. J., and Coello, C. A. (2013) Solving a real-world structural optimization problem with a distributed sms-emoa algorithm. *Second Int. Workshop on Soft Computing Techniques in Cluster and Grid Computing Systems (SCCG 2013), 3PGCIC*, Compiegne, France, October, pp. 600–605. IEEE.

[13] Zavala, G., Nebro, A. J., Luna, F., and Coello Coello, C. A. (2014) A survey of multi-objective metaheuristics applied to structural optimization. *Structural and Multidisciplinary Optimization*, **49**, 537–558.

[14] Alba, E. and Tomassini, M. (2002) Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, **6**, 443–462.

[15] Cantú-Paz, E. (2000) *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publisher, Netherlands.

[16] Luque, G. and Alba, E. (2011) *Parallel Genetic Algorithms*, Studies in Computational Intelligence, **367**. Springer-Verlag Berlin Heidelberg, Germany.

[17] Van Veldhuizen, D., Zydallis, J., and Lamont, G. (2003) Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms. *IEEE Trans. on Evolutionary Computation*, **87**, 144–173.

[18] López, J. A. and Coello Coello, C. A. (2009) Applications of parallel platforms and models in evolutionary multi-objective optimization. In Lewis, A., Mostaghim, S., and Randall, M. (eds.), *Biologically-Inspired Optimisation Methods*, Studies in Computational Intelligence, **210**, pp. 23–49. Springer Berlin Heidelberg, Germany.

[19] Talbi, E.-G., Mostaghim, S., Okabe, T., Ishibuchi, H., Rudolph, G., and Coello, C. A. C. (2008) Parallel approaches for multiobjective optimization. In Branke, J., Deb, K., Miettinen, K., and Sowiski, R. (eds.), *Multiobjective Optimization*, Berlin, Heidelberg, Germany, Lecture Notes in Computer Science, **5252**, pp. 349–372. Springer Berlin Heidelberg.

[20] Luna, F., Nebro, A. J., and Alba, E. (2006) Parallel evolutionary multiobjective optimization. In Nedjah, N., Alba, E., and de Macedo, L. (eds.), *Parallel Evolutionary Computations*, chapter 2, Studies in Computational Intelligence, **22**, pp. 33 – 56. Springer Berlin Heidelberg, Germany.

[21] Luna, F. and Alba, E. (2014) Parallel multiobjective evolutionary algorithms: an updated survey. *Handbook of Computational Intelligence*. Springer.

[22] Thain, D., Tannenbaum, T., and Livny, M. (2002) Condor and the Grid. In Berman, F., Fox, G., and Hey, T. (eds.), *Grid Computing: Making the Global Infrastructure a Reality*, December. John Wiley & Sons Inc., England.

[23] Zitzler, E. and Thiele, L. (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evolutionary Computation*, **3**, 257–271.

[24] Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., and Alba, E. (2007) Design issues in a multiobjective cellular genetic algorithm. In Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., and Murata, T. (eds.), *Evolutionary Multi-Criterion Optimization. 4th International Conference, EMO 2007*, Germany, March, Lecture Notes in Computer Science, **4403**, pp. 126–140. Springer.

[25] Clerc, M. and Kennedy, J. (2002) The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, **6**, 58–73.

[26] Bäck, T., Fogel, D., and Michalewicz, Z. (eds.) (1997) *Handbook of evolutionary computation*. Oxford University Press and Institute of Physics, New York.

[27] Durillo, J. J., Nebro, A. J., Luna, F., and Alba, E. (2008) A Study of Master-Slave Approaches to Parallelize NSGA-II. *IEEE Int. Symp. on Parallel and Distributed Processing, 2008 - IPDPS 2008*, Miami, FL, USA, April, pp. 1–8. IEEE.

[28] Durillo, J. J. and Nebro, A. J. (2011) jmetal: A java framework for multi-objective optimization. *Adv. Eng. Softw.*, **42**, 760–771.

[29] Knowles, J. (2005) A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers. *5th Int. Conf. on Intelligent Systems Design and Applications (ISDA'05)*, Wroclaw, Poland, September, pp. 552 – 557. IEEE.

[30] Sheskin, D. (2007) *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, Boca Raton, FL, USA.