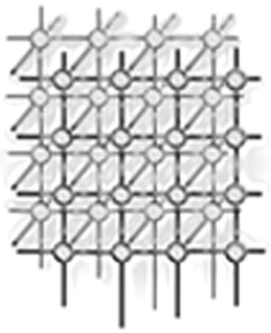
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE

Concurrency Computat.: Pract. Exper. 2006; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

MRMOGA: A New Parallel Multi-Objective Evolutionary Algorithm Based On the Use of Multiple Resolutions



Antonio López Jaimes, Carlos A. Coello Coello

CINVESTAV-IPN,

*Evolutionary Computation Group (*EVOCINV*),*

Departamento de Ingeniería Eléctrica,

Sección de Computación, Av. IPN *No. 2508, México, D.F. 07360, MEXICO*

SUMMARY

In this paper, we introduce MRMOGA (Multiple Resolution Multi-Objective Genetic Algorithm), a new parallel multi-objective evolutionary algorithm which is based on an injection island approach. This approach is characterized by adopting an encoding of solutions which uses a different resolution for each island. This approach allow us to divide the decision variable space into well-defined overlapped regions to achieve an efficient use of multiple processors. Also, this approach guarantees that the processors only generate solutions within their assigned region. In order to assess the performance of our proposed approach, we compare it with respect to a parallel version of an algorithm that is representative of the state-of-the-art in the area, using standard test functions

Contract/grant sponsor: CONACyT; contract/grant number: 45683-Y

Received 10 January 2006

Revised 9 May 2006

and performance measures reported in the specialized literature. Our results indicate that our proposed approach is a viable alternative to solve multi-objective optimization problems in parallel, particularly when dealing with large search spaces.

KEY WORDS: *Parallel processing; multi-objective evolutionary algorithms; parallel multi-objective evolutionary algorithms; multi-objective optimization; evolutionary algorithms*

1. INTRODUCTION

Evolutionary algorithms are heuristic search techniques inspired on Darwin's evolutionary theory (namely, the "survival of the fittest" principle) [11]. The idea of using techniques based on the emulation of the mechanism of natural selection to solve problems can be traced as long back as the 1930s [19]. However, it was not until the 1960s that the three main techniques based on this notion were developed: genetic algorithms [24], evolution strategies [42] and evolutionary programming [20].

Evolutionary algorithms have become very popular as optimizers in the last few years [22, 3]. As a consequence of this popularity, their use has spanned several new areas. One of these new areas is multi-objective optimization, which refers to the simultaneous optimization of two or more objective functions which are normally in conflict with each other. The first implementation of a multi-objective evolutionary algorithm (MOEA) dates back to the mid-1980s [39, 40]. Since then, a considerable amount of research has been done in this area, now known as evolutionary multi-objective optimization (EMOO for short). The growing importance

of this field is reflected by a significant increment (mainly during the last ten years) of technical papers in international conferences and peer-reviewed journals, books, special sessions in international conferences and interest groups on the Internet [9].*

Evolutionary algorithms have been found to be particularly suitable to solve multi-objective optimization problems due to several of their features [7, 9]:

- They deal simultaneously with a set of possible solutions (called "population"), which allows them to find several members of the Pareto optimal set (see definition 5) in a single run.
- They are less susceptible to the shape or continuity of the Pareto front than traditional mathematical programming techniques.
- They do not require information about the derivatives of the objective functions or an initial solution to start the search (in contrast to many traditional mathematical programming techniques).

Nowadays MOEAs have shown an acceptable performance in many real-world problems with their origins in engineering, scientific and industrial areas [9, 35, 14, 45, 10]. Nonetheless, in some of these problems, it is required to evaluate a huge number of objective functions or to use a very large population size in order to reach a desired effectiveness. Additionally, it is normally the case in real-world applications that the CPU time required to perform a single evaluation of the objective function(s) is too high as to make it impossible to find an acceptable solution in a reasonable time.

*The second author maintains an EMOO repository with over 2100 bibliographical entries at: `http://delta.cs.cinvestav.mx/~ccoello/EMOO`, with a mirror at `http://www.lania.mx/~ccoello/EMOO/`

The use of parallelism is an obvious choice to solve these problems in a reasonable amount of time [47]. Besides the time reduction that they can achieve, parallel MOEAs (pMOEAs) are attractive for many other reasons: (1) they can use more memory to cope with more difficult problems, (2) they allow the use of larger population sizes, (3) they improve the population diversity, (4) they reduce the probability of finding suboptimal solutions, (5) and they can cooperate in parallel with another search technique (including non-evolutionary techniques).

During the last three decades parallel Evolutionary Algorithms used for global optimization (pEAs) have been widely studied [2, 6, 38, 46]. However, due to the peculiarities of multi-objective optimization there are some issues that require the use of novel approaches. From these issues, the most relevant is the fact that the evaluation of each particular solution to a multi-objective optimization problem implies the evaluation of k ($k \geq 2$) objective functions. This implies a much higher computational cost and, therefore, motivates the need of parallelizing a MOEA. Additionally, real-world multi-objective optimization problems tend to have high-dimensionality (i.e., a large number of decision variables) which also normally requires a much higher computational cost in order to find a reasonably good approximation of the Pareto optimal set. Finally, the use of archiving, clustering or niching techniques (which are commonly adopted with MOEAs [28, 53, 16]) also adds to the computational overhead of a MOEA, which is one more reason to justify their parallelization.

This paper presents a scheme to split up the decision variable space in order to distribute it among different processors. The proposed algorithm, called Multiple Resolution Multi-Objective Genetic Algorithm (MRMOGA), is based on the injection island strategy proposed by Lin et al. [33]. In this strategy there is a number of sub-populations that encode the problem

using different resolutions in each island. This approach leads to the division of the decision variable space into different regions. Thus, each processor is able to concentrate its effort on exploring a different region of the search space. Additionally, using this approach, each processor can only generate solutions within its constrained region.

In order to evaluate the performance of the proposed approach we consider both its effectiveness and its efficiency. To evaluate the effectiveness we use well-known metrics normally adopted to evaluate serial MOEAs, namely: success counting, inverted generational distance, spacing, and two set coverage. On the other hand, efficiency is evaluated using the following well-known metrics from parallel computing: speedup, efficiency and serial fraction. The results of the proposed algorithm were compared against those of a parallel version of the NSGA-II [17]. Our experiments attempt to illustrate the pMOEAs' scalability as the number of processors increases. Our study involves the analysis of both the gains in convergence and the speedup.

The remainder of the paper is organized as follows. Section 2 provides some basic concepts required to understand the rest of the paper. In Section 3, we describe the major parallelization models adopted with MOEAs. In Section 4, we review the most significant previous related work. Section 5 is devoted to describing our proposed approach. The validation of our approach is provided in Section 7. Finally, Section 8 draws our conclusions and some possible paths for future research.

2. BASIC CONCEPTS

Definition 1 (Global Minimum:) *Given a function* $f : \Omega \subseteq \mathcal{R}^n \rightarrow \mathcal{R}$, $\Omega \neq \emptyset$, *for* $\vec{x} \in \Omega$ *the value* $f^* \triangleq f(\vec{x}^*) > -\infty$ *is called a global minimum if and only if*

$$\forall \vec{x} \in \Omega : \quad f(\vec{x}^*) \leq f(\vec{x}) \; . \tag{1}$$

Then, $\vec{x}^*$ *is the global minimum solution, and the set* Ω *is the feasible region* ($\Omega \in \mathcal{S}$), *where* $\mathcal{S}$ *represents the whole search space.* □

Definition 2 (General Multi-objective Optimization Problem (MOP):) *Find the vector* $\vec{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T$ *which will satisfy the* m *inequality constraints:*

$$g_i(\vec{x}) \leq 0 \qquad i = 1, 2, \ldots, m \tag{2}$$

the p *equality constraints*

$$h_i(\vec{x}) = 0 \qquad i = 1, 2, \ldots, p \tag{3}$$

and will optimize the vector function

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T \tag{4}$$

where $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ *is the vector of decision variables.* □

Definition 3 (Pareto Optimality:) *A point* $\vec{x}^* \in \Omega$ *is* **Pareto optimal** *if for every* $\vec{x} \in \Omega$ *and* $I = \{1, 2, \ldots, k\}$

$$\forall_{i \in I}(f_i(\vec{x}^*) \leq f_i(\vec{x})) \tag{5}$$

and, there is at least one $i \in I$ *such that*

$$f_i(\vec{x}^*) < f_i(\vec{x}) \tag{6}$$

□

In words, this definition says that $\vec{x}^*$ is Pareto optimal if there exists no feasible vector $\vec{x}$ which would decrease some criterion without causing a simultaneous increase in at least one other criterion. The phrase "Pareto optimal" is considered to mean with respect to the entire decision variable space unless otherwise specified.

Definition 4 (Pareto Dominance:) *A vector* $\vec{u} = (u_1, \ldots, u_k)$ *is said to dominate* $\vec{v} = (v_1, \ldots, v_k)$ *(denoted by* $\vec{u} \preceq \vec{v}$*) if and only if* $\vec{u}$ *is partially less than* $\vec{v}$*, i.e.,* $\forall i \in \{1, \ldots, k\},\ u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i$. □

Definition 5 (Pareto Optimal Set:) *For a given MOP* $\vec{f}(x)$*, the Pareto optimal set* ($\mathcal{P}^*$) *is defined as:*

$$\mathcal{P}^* := \{x \in \Omega \mid \neg\exists\, x' \in \Omega \ \ \vec{f}(x') \preceq \vec{f}(x)\}. \tag{7}$$

□

Pareto optimal solutions are also termed *non-inferior*, *admissible*, or *efficient* solutions [25]; their corresponding vectors are termed *non-dominated*.

Definition 6 (Pareto Front:) *For a given MOP $\vec{f}(x)$ and Pareto optimal set $\mathcal{P}^*$, the Pareto front ($\mathcal{PF}^*$) is defined as:*

$$\mathcal{P}F^* := \{\vec{f} = (f_1(x), \ldots, f_k(x)) \mid x \in \mathcal{P}^*\}. \tag{8}$$

□

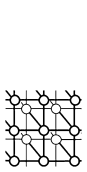
In the general case, it is impossible to find an analytical expression of the line or surface that contains these points. The normal procedure to generate the Pareto front is to compute the feasible points Ω and their corresponding $f(\Omega)$. When there is a sufficient number of these, it is then possible to determine the non-dominated points and to produce the Pareto front.

3. PARALLELIZATION OF MULTI-OBJECTIVE EVOLUTIONARY ALGORITHMS

The parallelization schemes that have been proposed for MOEAs are derived from the well-known models designed for single-objective optimization: the master-slave model, the island model and the diffusion model. Nevertheless, currently there is no standard way to extend these models to the multi-objective field. In [47], the authors provide a detailed discussion of the generic versions of these models when applied to MOEAs. Next, we will briefly discuss each of them.

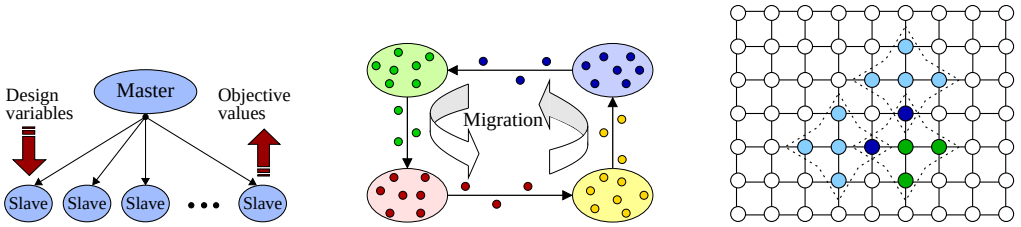
Master-Slave model. The master-slave model is one of the simplest ways to parallelize a MOEA and hence the most popular among practitioners. Here, a master processor executes the MOEA, and the objective function evaluations are distributed among a number of slave processes. As soon as the slaves complete the evaluations they return the objective values to the master and remain idle until the next generation. In addition to the evolutionary operators (selection, recombination and mutation), the master processor executes other tasks such as Pareto ranking, and archiving. This model is depicted in Figure 1(a). A master-slave pMOEA explores the search space in the same way as a serial MOEA does, therefore it finds the same solutions as those found by its serial counterpart. However, the execution time is reduced (ideally p times with p processors). It is important to note that the time required to evaluate the objective functions needs to be greater than the communication time in order to get a significant gain in speed with this sort of model [6].

Diffusion model. Like the master-slave model, the diffusion model considers a unique population, but in this case the population is spatially distributed onto a neighborhood structure. Usually this structure is a two dimensional rectangular grid, and there is one individual per grid point (Figure 1(b)). Ideally, there is one processor per individual, and therefore this model is sometimes called *fine-grained*. This kind of pMOEA is also known as a *cellular* pMOEA because the model is similar to a cellular automaton with stochastic transition rules. The selection and mating is confined to a small neighborhood around each individual. The neighborhoods are overlapped (as depicted by the dotted lines in Figure 1(b)) so that the good traits are spread or "diffused" throughout the whole population. The communication costs tend to be high, since the individuals who take part in the selection are distributed



among several processors. For this reason, this model is appropriate for shared-memory MIMD computers, or massively parallel SIMD computers.

Island model. This model was inspired by the natural phenomenon in which a number of spatially isolated populations are linked together by dispersal and migration. In an island pMOEA, the population is divided into several small sub-populations, called *islands* or *demes*, which evolve independently of each other. In each of these islands a serial pMOEA is executed for a number of generations called an *epoch*. At the end of each epoch, individuals migrate between neighboring islands. The neighbors are given by the *migration topology*, which determines the migration paths along which individuals can move to other islands. A typical representation of the island model is shown in the Figure 1(c), in which a ring topology is adopted, although other migration topologies are possible (2-D and 3-D meshes, tori, hypercubes or trees). Island pMOEAs are also known as *distributed* pMOEAs, as they are usually implemented on distributed memory MIMD computers. In particular, due to the low inter-processor communication frequency, this model is well-suited for clusters of computers. This model is very popular among researchers, but it requires many parameters and design decisions. The main issues to consider with this sort of model include the migration topology, the migration frequency, the number of individuals to migrate, and the decision regarding the individuals which will migrate and those which will be replaced by the immigrants. The model allows each island to have their own parameter setting. Depending on the homogeneity of the islands, we can recognize two variants of the island model:



(a) Master-slave (b) Island (c) Diffusion

Figure 1. Diagrams of the three major models used to parallelize MOEAs.

1. **Island pMOEA with homogeneous nodes**. The MOEAs performed in every island have all the same parameter values (e.g., population size, mutation, crossover and migration rate).
2. **Island pMOEA with heterogeneous nodes**. Each island applies a MOEA which has a different parameter setting, or uses its own evolutionary operators and solution encoding technique. In addition, each island may be explicitly instructed to explore a particular region of decision variable or objective function space.

4. PREVIOUS RELATED WORK

Although several researchers have reported the use of parallel MOEAs, not many of them have actually proposed novel schemes to parallelize a MOEA. Next, we will review the most representative work along these lines that we were able to find in the specialized literature.

The Divided Range Multi-Objective Genetic Algorithm (DRMOGA) was proposed by Hiroyasu et al. [23]. Here, the global population is sorted according to one of the objective

functions (which is changed after a number generations). Then, the population is divided into equally-sized sub-populations. Each of these sub-populations is allocated to a different processor in which a serial MOEA is applied. At the end of a specific number of generations, the sub-populations are gathered and the process is repeated, but this time using some other objective function as the sorting criterion. The main goal of this approach is to focus the search effort of the population on different regions of the objective space. However, in this approach we cannot guarantee that the sub-populations will remain in their assigned region. A similar approach is followed in [12].

Zhu & Leung proposed the Asynchronous Self-Adjustable Island Genetic Algorithm (aSAIGA) [50]. In aSAIGA, rather than migrating a set of individuals, the islands exchange information related to their current explored region. Based on the information coming from other islands, a "self-adjusting" operation modifies the fitness of the individuals in the island to prevent two islands from exploring the same region. In a similar way to DRMOGA, this approach cannot guarantee that the sub-populations move tightly together throughout the search space, hence the information about the explored region may be meaningless.

Another island pMOEA was introduced by Deb et al. [18]. In this case, although all processors search on the entire decision variable space, the approach assigns each processor a different search region of the Pareto-optimal front. In order to steer the search towards the assigned region, the authors adopt a "guided domination" based on a concept defined in [5]. This concept uses a weighted function of the objectives in order to achieve a larger dominated region for each vector. Therefore, each processor using this new concept only finds a region of the real Pareto front. The weakness of this approach is that we must have *a priori* knowledge of the

shape of the Pareto front in order to define accurately the search directions. Furthermore, this technique can only deal with convex Pareto fronts.

Streichert et al. [44] recently proposed an approach that partitions the overall population using a clustering algorithm aiming to specialize the exploration of each island on different areas of the Pareto front. Periodically (after a specified number of generations) the islands are gathered, clustered and redistributed onto the available processors. The individuals are kept within their region by considering this as their feasible zone and using the constrained dominance principle defined in [15]. That is, any individual generated outside its constrained region is marked as "invalid". The main drawback of the approach is that the repeated gathering of all sub-populations produces a high communication overhead, which is increased with the number of processors.

5. MULTIPLE RESOLUTION SCHEME

In most of the approaches that use a "divide and conquer" strategy, the division has been made in the objective function space [18, 23, 49, 50]. To the best of our knowledge, only Streichert et al. [44] carried out such division in the decision variable space. Nonetheless, none of these approaches guarantees the generated individuals belong to their assigned region. On the other hand, the secondary population (external archive), if any, does not interact in the search process in any of the approaches previously described.

Taking care of these observations, we designed a model called Multiple Resolution Multi-Objective Genetic Algorithm (MRMOGA). Our approach is based on the pMOEA island model with heterogeneous nodes and is characterized by the following:

- Each island encodes the solutions with a different resolution (i.e., different number of precision digits after the decimal point). This implies a partition of the decision variable space in such a way that every new individual belongs to its assigned region.
- It uses a migration strategy that considers both the primary population and the secondary population (or external archive).
- It uses a strategy to detect the nominal convergence of the islands in order to increase the initial resolution adopted.

We describe the MRMOGA in Algorithm 1. In each island, we execute a serial MOEA which is initialized as usual. However, the MOEA of each island uses a different resolution. At the time of migration each processor takes some individuals from its archive ($PF_{known}\binom{i}{t}$, where i denotes the processor number and t the generation number) to send them to the primary population of its neighbors. At the end of the search process, the root process combines the non-dominated front of each processor $PF_{known}\binom{i}{t}$. Figure 2 shows a schematic view of the algorithm. In the next sections all the components of the approach are described in detail.

5.1. The Principle of Multiple Resolutions

The idea behind the proposed approach is based on the following assumption: the Pareto optimal solutions are found in fewer iterations using low resolution representations than using higher resolution representations. This is a rather obvious idea, since the search space is smaller as the resolution decreases (when adopting a discrete search space), and hence we need to explore fewer solutions to produce our approximation of the Pareto front (PF_{known}).

Input:
$\mathcal{P}_i$: Population of the i-th processor.
E: Maximum number of epochs.
G: Generations per epoch.

Randomly generate the population $\mathcal{P}_i$ on each processor.
for $e \leftarrow 1$ until E **do**
  for $g \leftarrow 1$ until G **do**
    SERIALMOEA($\mathcal{P}_i$)
  end for
  if $e \neq E$ **then** ▷ Do not migrate in last epoch
    ▷ According to migration topology
    for all outgoing neighbor j of processor i **do**
      MIGRATE($PF_{known}^{(i)}_{(g)}$, $\mathcal{P}_j$)
    end for
    for all incoming neighbor j of processor i **do**
      REPLACE($\mathcal{P}_i$, $PF_{known}^{(j)}_{(g)}$)
    end for
  end if
end for
Combine all $PF_{known}^{(i)}$ in processor 0.
Filter overall population through the grid and show FP_{known}.

Algorithm 1: MRMOGA pseudocode.

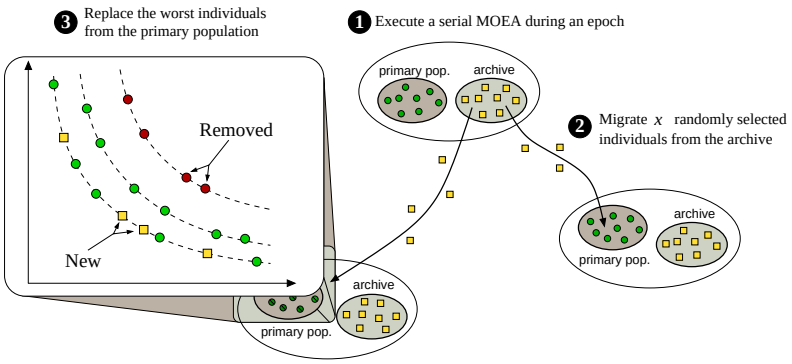


Figure 2. Schematic view of our MRMOGA.

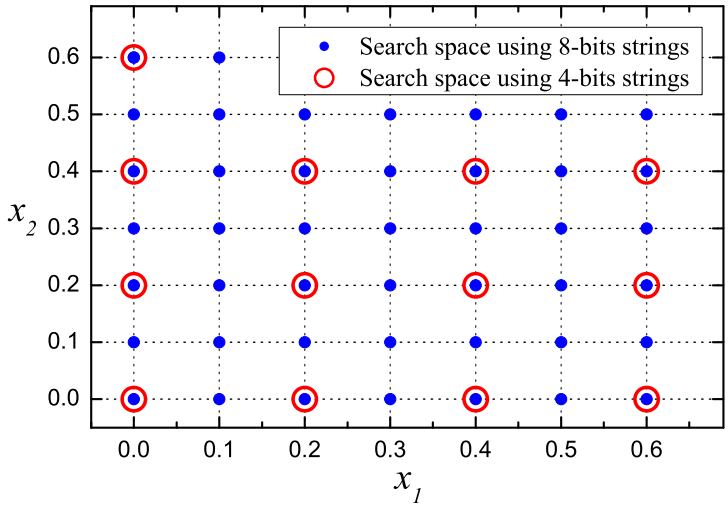


Figure 3. Search space for resolutions r_1 and r_2 achieved with 4 and 8 bits, respectively.

Taking advantage of this idea, we can design an island model in which each node encodes the solutions with a different resolution. The low resolution islands have the purpose of approaching quickly towards the true Pareto front (PF_{true}), thus finding individuals with high fitness with a relatively low number of evaluations. Afterwards, these individuals are migrated into higher resolution nodes for exploiting the regions nearby these highly fit individuals.

Our approach can be considered as a partition of the decision variable space. In such a partition there are well-defined overlapped solutions as is illustrated in Figure 3. As the figure shows, all solutions of the search space generated with 4 bits (circles) are contained within the 8-bits search space (dots). That is, the search space of a low-resolution island is contained in the search space of a higher resolution island.

The interpretation of "resolution" is problem dependant. For instance, in numerical optimization problems, the resolution is given by the number of digits after the decimal point. However for discrete optimization problems, like the traveling salesperson problem, the interpretation is not straightforward. The implementation of MRMOGA for this study was intended for numerical optimization, and we thus adopted a binary string representation. However, it is possible to use another type of representation/encoding depending on the nature of the problem. For instance, Lin et al. [33] proposed an encoding scheme for a problem described by discrete variables (i.e., the graph-partitioning problem). Note however, that the use of other types of representation (e.g., real numbers), although possible, may raise issues not dealt with in this paper.

5.2. Topology and Migration/Replacement Scheme

The search space of an island is contained within the search space of any island with higher resolution. However, the converse is not true (see Figure 3). The underlying assumption here is that low resolution islands approach faster to PF_{true} than high resolution islands. Thus, the natural choice is that the migration flow can only go from low resolution islands to high resolution islands. Although it is possible to exchange individuals from high to low resolution islands, this could cause some problems due to the loss of precision digits, and also adds communication overhead. For instance, a feasible solution may become infeasible. Also, it is worth noting that if migration from high resolution islands to low resolution islands is performed, it is unlikely that incoming individuals are closer to PF_{true} than the low resolution individuals.

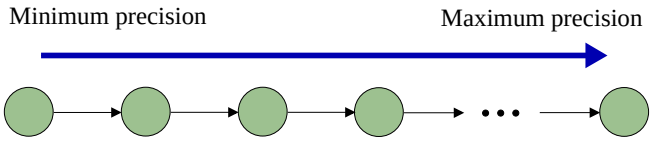


Figure 4. Strict topology.

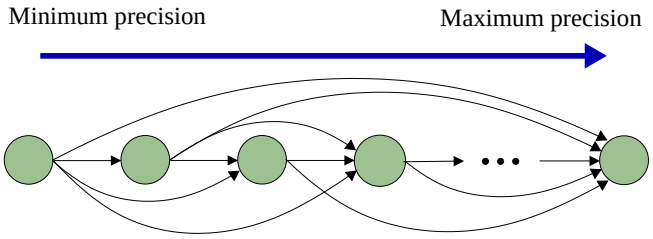


Figure 5. Complete topology.

The migration from low to high resolutions imposes a hierarchical migration topology. That is, a parent island has as children all the islands with higher resolution. The root of the topology is the island with the smallest binary strings. Taking care of this, we implemented the two following topologies, which are similar to those defined in [33]:

- **Strict topology**. Each island migrates only solutions to its direct child (Figure 4). The degree of this topology (i.e., the number of incoming neighbors of each node) is $\delta(i) = 1$ for every node $i = 0, \ldots, n$.
- **Complete topology**. Each island migrates solutions to all of its children in the hierarchy (Figure 5). This topology has degree $\delta(i) = i$ for every node $i = 0, \ldots, n$.

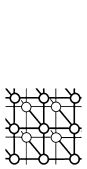
Both the primary population and the external archive are involved in the migration/replacement scheme. Algorithm 2 shows the pseudocode of the migration and replacement schemes.

- **Migration scheme**. This scheme is similar to the *elitist random* scheme suggested in [47]. Our scheme migrates x randomly selected members from the archive (i.e., the known Pareto front). If the archive size is less than x, the required individuals to complete x are randomly selected from the primary population.
- **Replacement scheme**. Initially, the members of the primary population are ranked in non-dominated levels by the method used in the NSGA-II [15]. Later, x individuals are replaced from the last ranked Pareto front (the "worst" individuals). If the size of the last front is less than x, a sufficient number of individuals are taken from the next ranked fronts until the x individuals are completed. Finally, the procedure of the Algorithm 3 is applied to the incoming individuals to adapt its chromosomes to the new length.

5.3. Need for Conversion in Migration

Since in the migration process of our approach we exchange chromosomes (binary strings) with different lengths, ℓ and ℓ' ($\ell < \ell'$), we need to adapt the smaller length string into a larger length string. Such adaptation must fulfill the following requirements:

- It should be possible to apply the evolutionary operators to the re-encoded string with the new length ℓ'.



procedure MIGRATE($PF_{known}\binom{i}{t}$, m)
  if $|PF_{known}\binom{i}{t}| \geq m$ **then**
    $migrants \leftarrow m$ randomly selected members of $PF_{known}\binom{i}{t}$
  else
    $migrants \leftarrow PF_{known}\binom{i}{t} + (m - |PF_{known}\binom{i}{t}|)$ randomly selected members of $\mathcal{P}_i$
  end if
  Send $migrants$ to neighbor
end procedure

procedure REPLACE($PF_{known}\binom{i}{t}$, m)
  Receive m individuals (immigrants) from the neighboring processors
  Sort $\mathcal{P}_i$ into non-dominated levels
  Replace m individuals of the last ranked front of $\mathcal{P}_i$ with immigrants
  for all decision vector $\mathbf{x}$ of the immigrants **do**
    CONVERT($\mathbf{x}, x_i^{min}, x_i^{max}, \ell'$)
  end for
end procedure

Algorithm 2: Pseudocode of the migration and replacement procedures.

- The re-encoded string must represent the same phenotype (decoded variable vector) as the original string with length ℓ.

The expression adopted to obtain a real value, $x_i \in [x_i^{min},\ x_i^{max}]$, from a binary string, s_i, is the following:

$$x_i = x_i^{min} + \frac{d_i}{2^{\ell_i} - 1}(x_i^{max} - x_i^{min}), \tag{9}$$

where ℓ_i is the length of the string s_i that encodes the i-th variable and d_i is the decoded integer value of the string s_i.

In the proposed approach only the phenotypes of the individuals are migrated, i.e., the vector of the decoded variables, $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. To carry out the conversion, we need to

find the strings s'_i (with the new length ℓ'_i) that produce the decision values x_i. For this, we need to solve equation (9) for d_i, which gives us:

$$d_i^* = \frac{x_i - x_i^{min}}{x_i^{max} - x_i^{min}}(2^{\ell'_i} - 1), \tag{10}$$

where d_i^* is the nearest real value to d_i, which is obtained with a precision of ℓ'_i bits. Now, we round up this value to its nearest integer. Finally, we encode the rounded up value of d_i^* to obtain the binary string with length ℓ'. In Algorithm 3, we depict the pseudocode for mapping a chromosome of length ℓ into another one of length ℓ'. Note that the vector $\mathbf{x}^*$ generated from string s' (by equation (9)) is not exactly the same as the original vector $\mathbf{x}$. However, the mapping guarantees to keep a precision of at least p'_i ($p_i < p'_i$) digits after the decimal point.

Input:
$\mathbf{x}$: Decoded variable vector.
x_i^{min}: Lower bound for the i-th variable.
x_i^{max}: Upper bound for the i-th variable.
ℓ': Vector with the lengths of the new strings s'_i.

Output:
s': Resulting binary string of length ℓ'.

function CONVERT($\mathbf{x}, x_i^{min}, x_i^{max}, \ell'$)
  $s' \leftarrow \varepsilon$
  for each variable i encoded in the chromosome s' **do**
    $d_i^* \leftarrow$ ROUND$\Big((x_i - x_i^{min}) * (2^{\ell'_i} - 1)/(x_i^{max} - x_i^{min})\Big)$
    $s'_i \leftarrow$ DECIMAL2BINARY$(d_i^*,\ \ell'_i)$
    $s' \leftarrow s' + s'_i$ ▷ String concatenation
  end for
end function

Algorithm 3: Procedure to map a binary string s of length ℓ into a string s' of length ℓ' ($\ell < \ell'$).

5.4. The serial MOEA applied on each sub-population

In each processor, we perform the serial MOEA described in Algorithm 4. In the following sections, we describe the elements that integrate the serial MOEA.

procedure SERIALMOEA($maxResolution$, $\mathcal{P}_i$)
  Evaluate objective values
  Assign rank based on Pareto dominance
  Assign linearly scaled fitness
  Filter vectors of $\mathcal{P}_i$ through the adaptive grid
  if $PF_{known}\binom{i}{t}$ has converged and $resolution_i < maxResolution$ **then**
    Increment the resolution of $\mathcal{P}_i$
  end if
  Generate offspring (selection, crossover and mutation)
end procedure

Algorithm 4: Pseudocode of the serial MOEA of each processor.

5.4.1. Fitness Assignment

The fitness assignment is based on the strategy used by MOGA [21]. The rank of an individual is given by:

$$\text{rank}(x_i, t) = 1 + p_i^{(t)}$$

where x_i is an individual at generation t, which is dominated by $p_i^{(t)}$ individuals in the current generation. The fitness is assigned by the following steps:

1. Sort the population according to rank.
2. Assign fitness to individuals by interpolating from the best (rank 1) to the worst (rank $n \leq M$).

3. Average the fitness of individuals with the same rank, so that all of them are sampled at the same rate.

5.4.2. Genetic operators

The genetic operators adopted are among the simplest and most commonly used in the Evolutionary Computation community:

Selection. The selection technique used was the Stochastic Universal Selection proposed by Baker [4]. This technique reduces the differences between the expected values and the actual number of copies adopted for an individual, with respect to the values obtained with other proportional selection techniques such as the roulette wheel [34]. The pseudocode of this technique is shown in the Algorithm 5.

```
Input:
  P: The current population.

  procedure SELECTION(P)
      r ← RANDOM()        ▷ Generates a random number between 0.0 and 1.0
      sum ← 0
      for all individual i in population P do
          sum ← sum + e(i)        ▷ e(i) denotes the expected number of copies of individual i
              in the next generation
          while sum > r do
              SELECT(i)
              r ← r + 1
          end while
      end for
  end procedure
```

Algorithm 5: Stochastic Universal Selection.

Crossover. We adopted two-point crossover. In this technique two crossover points are randomly selected and the substrings between those points are exchanged to produce

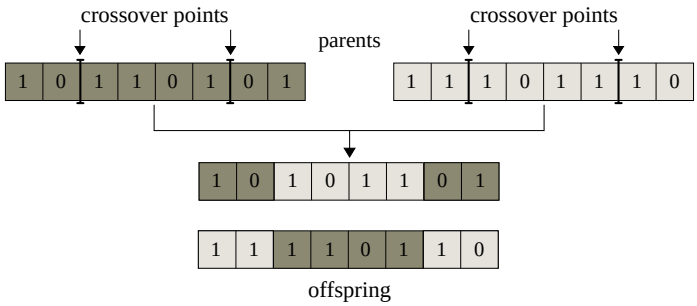


Figure 6. Example of the two-point crossover.

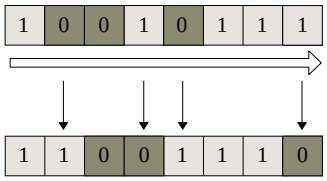


Figure 7. Uniform mutation where each string's value has a 50% probability of being changed.

two children. Two-point crossover tends to produce a better convergence than one-point crossover [34], and therefore our decision to adopt it. Figure 6 shows an example of this technique.

Mutation. We adopted uniform mutation, which is the most common mutation approach used with binary encoding [22]. In this technique each value of the chromosomic string is interchanged by its inverse according to the mutation probability. This technique is depicted in Figure 7.

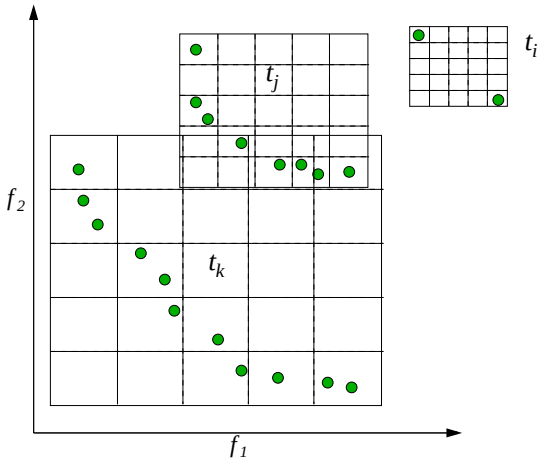
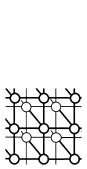


Figure 8. Hypergrid used to maintain the archive diversity.

5.4.3. Technique to maintain diversity

In order to retain the non-dominated solutions and to distribute them evenly along the Pareto front, MRMOGA uses an adaptive grid similar to the one proposed in [29]. Before the archive has reached its maximum capacity, the non-dominated solutions are added to the archive if they are not dominated by some member of the archive. When the archive is full, then the adaptive grid is used.

The adaptive grid divides the objective function space into regions called hypercubes. Every non-dominated solution occupies only one hypercube, as is shown in Figure 8. The basic idea is that a non-dominated solution is accepted only if it belongs to a hypercube that has fewer individuals than the most crowded hypercube. Although the number of regions is constant over time, the location and extension of the grid may change (i.e., adapt) over time (see Figure 8).



5.4.4. Constraint-handling strategy

Although the aim of this work was not to propose a specialized constraint-handling technique, in order to explore extensively the MRMOGA's abilities, we incorporated a simple strategy to handle constraints. The main reason for providing such a mechanism is the fact that among the six test problems included in our experimental study, there are two well-known test problems with constraints (see Sections 6.3.3 and 6.3.4).

In our constraint-handling strategy the dominance between two solutions is determined by comparing the number of constraints violated by each solution. A solution with a smaller number of violated constraints dominates the other. In the case that both individuals violate the same number of constraints, Pareto dominance is applied as usual. This strategy is used in the ranking process, and at the moment of deciding whether or not a solution is accepted into the external archive.

5.4.5. Increasing an island's resolution

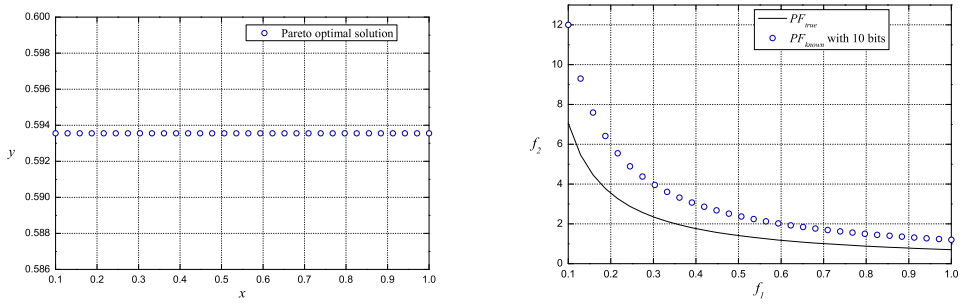
MRMOGA has the capability of generating solutions only within the decision subspace that each island was devoted for. Unfortunately, this approach has some drawbacks since the regions are assigned statically: For instance, the Pareto optimal solutions may be concentrated within a small area of the search space so that some regions could contain few or no Pareto optimal solutions. This leads to an imbalance in the contribution of each processor to find the true Pareto front. In terms of efficiency, the power of some processors is wasted, while in terms of effectiveness, the probability of finding optimal solutions is decreased. In fact, Shonkwiler [43]

has shown that the probability of finding a solution increases by running a genetic algorithm on several processors instead of just one.

In the case of MRMOGA we recognize the following difficulties:

- The search space of the low resolution islands is proportionally smaller. Therefore, the true Pareto front is found in this case, in fewer iterations than in higher resolution islands. Thus, once the low resolution islands have converged, they spend the remainder of time oscillating around the Pareto front without any real contribution regarding the generation of other regions of the Pareto front. This issue was originally identified by Parmee and Vekeria [37] when they used an injection island strategy to solve a single-objective engineering optimization problem.
- For some MOPs, the PF_{known} obtained with certain resolutions (particularly when using low resolutions) do not belong to PF_{true}, thus, the islands with those resolutions do not contribute at all to finding solutions that are part of PF_{true}. This implies losing these islands from the beginning of the search process. In Figure 9, we illustrate this situation. As can be seen, the known Pareto front obtained with a 10-bit resolution (i.e., one decimal precision) does not belong to the true Pareto front of the problem.

In order to cope with these difficulties, we use a similar strategy to that used in [37], namely, to increase the resolution of each island once it has reached nominal convergence. In this way, at every moment the processors' utilization is kept high, and the islands that have already converged will keep contributing to find other regions of the true Pareto front.



(a) (b)

Figure 9. Known Pareto optimal set (a) using 10-bits chromosomes (one decimal precision), and its corresponding known Pareto front (b).

5.4.6. Criterion to Detect Nominal Convergence

The detection of the nominal converge in the islands is achieved by monitoring the movements in the external archive. In the process to determine whether or not an individual is inserted into the known Pareto front ($PF_{known}(t)$), one of the following situations occur:

1. Regardless of whether the individual belongs or not to $PF_{known}(t)$, the solutions dominated by the individual are removed from the archive. In this case, we can interpret that $PF_{known}(t)$ is getting closer to PF_{true}.
2. The individual is inserted into $PF_{known}(t)$ without dominating any individual of the archive. The archive is still growing, which means new regions of PF_{true} are being discovered.

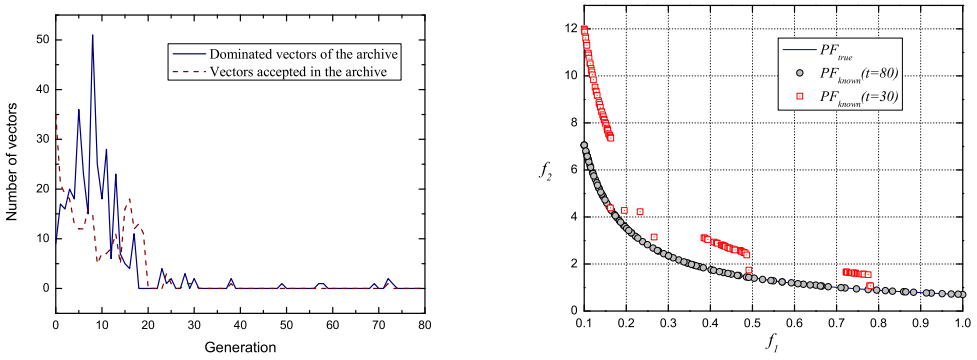
3. The individual does not dominate any member of $PF_{known}(t)$, but replaces the individual located in the most crowded hypercube. The search has arrived at a phase where is mainly concentrated on achieving a better distribution of $PF_{known}(t)$.

To illustrate the three above situations, let us track the progress of a simulation run of the algorithm on a single processor. In Figure 10(a), we illustrate the number of dominated vectors of $PF_{known}(t)$ and the number of vectors added to $PF_{known}(t)$ at each generation. On the other hand, Figure 10(b) shows $PF_{known}(t)$ at two distinct stages of the search process. In the first 30 generations there is a large number of dominated vectors of $PF_{known}(t \leq 30)$ (Fig. 10(a)), hence we can infer that the population still advances towards PF_{true}. This fact is reflected in the known Pareto front found at generation 30 in which there are few individuals nearby PF_{true}. In contrast, after generation 30, the number of dominated vectors of the known Pareto front begins to decrease down to almost zero. At this moment, $PF_{known}(t = 80)$ is located exactly on PF_{true} (Fig. 10(b)).

The population of an island is near to reaching nominal convergence if at each generation most of the movements belong to case 3. To detect convergence, at each generation, we keep a record of the number of dominated individuals in $PF_{known}(i)$ (case 1) at generation i ($dominated_i$).

Depending on $dominated_i$, we decide whether or not an island has reached convergence such that it is ready to increase the resolution of the population. At each generation, we check if in the k last generations the following condition was satisfied:

$$\sum_{i=1}^{k} \frac{dominated_i}{k} \leq \epsilon, \tag{11}$$



(a) (b)

Figure 10. The plot (a) shows the number of solutions inserted into the archive and the number of dominated members of $PF_{known}(t)$. The plot (b) shows $PF_{known}(t)$ at generations $t = 30$ and $t = 80$.

where ϵ is the average of dominated individuals of PF_{known} in the last k generations. The constant ϵ was experimentally set to 0.05. If the condition (11) is met, then the island resolution is increased to the next resolution. The resolution change is accomplished in the following way: one half of the population is randomly reinitialized taking into account the new resolution. The other half is a random sample taken from the archive. The conversion of the individuals from the second half to the new resolution is performed using the same procedure adopted for migration.

6. METRICS AND TEST PROBLEMS

The metrics we used are divided in two parts. Metrics to evaluate effectiveness and metrics to evaluate efficiency.

6.1. Metrics to evaluate effectiveness

In evolutionary multi-objective optimization, it is generally accepted that the quality of the Pareto front obtained by an algorithm must be evaluated taking into consideration three goals: (*i*) to minimize the distance between the approximation obtained and the true Pareto front (normally obtained by enumeration); (*ii*) to achieve a good distribution of the non-dominated solutions obtained; (*iii*) to maximize the spread of the Pareto front obtained.

In order evaluate these three aspects, we adopted four metrics in our comparative study. To measure the distribution we used the *spacing metric*, for evaluating the closeness to the Pareto-optimal front we used the *inverted generational distance*, the *success counting*, and the *two set coverage* metrics. Finally, to evaluate the spread we also used the *inverted generational distance*. Additionally, visual comparisons of the generated non-dominated sets are made in order to verify if the actual behavior of the algorithm correspond to what the metrics indicate. Each of the metrics adopted is defined next.

- **Success Counting** (SC). This metric, introduced in [8], counts the number of vectors in PF_{known} that belong to PF_{true}. This metric is formally described as:

$$SC = \sum_{i=1}^{n} s_i, \tag{12}$$

where $n = |PF_{known}|$ and

$$s_i = \begin{cases} 1 & \text{if vector } \mathbf{u}^{(i)} \in PF_{known} \text{ is in } PF_{true}, \\ 0 & \text{otherwise.} \end{cases}$$

It should be clear that $SC = n$ represents the ideal behavior, since every vector reported in PF_{known} by the MOEA is member[†] of PF_{true}. $SC = 0$ indicates that no reported vector belongs to PF_{true}.

- **Inverted Generational Distance** (IGD). The main goal of this metric is to measure the overall progress of PF_{known} towards PF_{true}. This metric is defined by:

$$IGD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n}, \tag{13}$$

where $n = |PF_{true}|$ and d_i is the Euclidian distance between each vector of PF_{true} and the nearest member of PF_{known}. In addition, this metric measures the spread of PF_{known} onto PF_{true}. That is, a non-dominated set whose vectors are located on a reduced area of the Pareto-optimal set, will be penalized in the value of this metric even though its vectors belong (or are near) to PF_{true}. Lower values are preferred for this metric. This way, $IGD = 0$ indicates $PF_{known} \subseteq PF_{true}$ and its elements are well spread throughout PF_{true}. Any other value in IGD means PF_{known} deviates from PF_{true}.

- **Two Set Coverage** ($\mathcal{C}$). This metric was suggested in [51] to compare the overall quality of two non-dominated sets. If A and B are two non-dominated sets, the coverage metric

[†]In this paper is assumed that a vector, $\mathbf{u}$, belongs to PF_{true} if and only if there is no vector in PF_{true} which dominates $\mathbf{u}$.

$\mathcal{C}(A,B)$ calculates the fraction of vectors in B which are weakly dominated[‡] by A:

$$\mathcal{C}(A,B) = \frac{|\{b \in B \mid \exists\, a \in A : a \preceq b\}|}{|B|} \tag{14}$$

The value $\mathcal{C}(A,B) = 1$ reflects that all members of B are dominated by or equal to some member of A (A is said to cover B). The opposite, $\mathcal{C}(A,B) = 0$, represents the situation where none of the solutions in B is covered by A. It is required to calculate both $\mathcal{C}(A,B)$ and $\mathcal{C}(B,A)$, because in general, $\mathcal{C}(A,B)$ differs from $1 - \mathcal{C}(B,A)$.

- **Spacing** (SP). The spacing metric was proposed by [41] as a way of measuring the range (distance) variance of neighboring vectors in PF_{known}. This metric is defined by:

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{d} - d_i)^2}, \tag{15}$$

where $d_i = min_j(|f_1^i(\mathbf{x}) - f_1^j(\mathbf{x})| + |f_2^i(\mathbf{x}) - f_2^j(\mathbf{x})|)$, $i,j = 1,\ldots,n$, $\bar{d}$ is the average of all d_i and n is the number of vectors in PF_{known}. A value of $SP = 0$ indicates that all vectors in PF_{known} are equidistantly spaced. It is worth mentioning that a PF_{known} set with a good value in SP is not necessarily well spread throughout the whole front. Additionally, note that only the shortest distance from each vector to its neighbors is used. Therefore, if we have a PF_{true} composed of two or more Pareto regions with more than one vector (MOP in section 6.3.1), then the distance between these regions will not be taken into account to compute the metric.

[‡]A vector $\mathbf{u}$ is said to weakly dominate $\mathbf{v}$ (denoted by $\mathbf{u} \preceq \mathbf{v}$) iff $\forall i \in \{1,...,k\} : u_i \leq v_i$.

6.2. Metrics to evaluate efficiency

In order to evaluate the efficiency of our MRMOGA we took three well-known metrics from the parallel computing literature: speedup, efficiency and serial fraction. Each of them is defined next.

- **Speedup** (S). In this study, we used the *orthodox speedup* suggested by Alba [1]. This speedup allows a fair comparison since both the serial and the parallel MOEA are stopped when they have reached the same target solution "quality". This speedup is calculated by the expression:

$$S_p = \frac{\overline{T}_1}{\overline{T}_p}, \tag{16}$$

  where $\overline{T}_1$ is the average execution time of the pMOEA with p sub-populations running on a single processor and $\overline{T}_p$ denotes the average execution time of the same pMOEA run on p processors.

- **Efficiency** (E). This metric [30] measures the fraction of time that a processor is effectively utilized. It is defined by the ratio between the speedup and the number of processors used. This metric is defined as:

$$E_p = \frac{S_p}{p}, \tag{17}$$

  where S_p is the speedup achieved with p processors. In an ideal system, the efficiency is equal to 1. In reality, due to the communications costs, and the idle time caused by the synchronization, the efficiency oscillates between 0 and 1.

- **Serial Fraction** (F). This metric was defined by Karp and Flatt [26] to estimate the serial fraction[§] for measuring the performance of a parallel algorithm on a fixed-size problem. Mathematically is defined as:

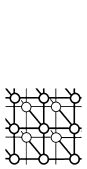
$$F_p = \frac{1/S_p - 1/p}{1 - 1/p}. \tag{18}$$

where S_p the speedup achieved with p processors. Smaller values of F are considered better. If F increases with the number of processors, then it is a symptom of growing communications costs, and therefore an indicator of poor scalability. Thus, if the speedup of an algorithm is small, we can still say that is efficient if F_p remains constant when increasing p. In this case, the efficiency drops due to the limited parallelism of the algorithm. On the other hand, if the value of F decreases with p, Karp and Flatt [26] consider that the speedup tends to be superlinear.

6.3. MOPs used to test MRMOGA

The assessment of MRMOGA was carried out using six multi-objective problems that have some recognized features which may cause difficulties for MOEAs, namely: converging to the real Pareto front, and maintaining population diversity [13]. This group of problems was chosen because they have a large search space that make them suitable to solve them by a pMOEA. One problem was selected by having a disconnected real Pareto front that challenges the pMOEA's

[§]The ratio of the time taken by the inherently serial component of an algorithm to its execution time on one processor.



ability to maintain diversity within the population. Finally, we selected two multi-objective test problems with constraints.

6.3.1. Test problem KUR

This two-objective problem was defined by Kursawe [31]:

Minimize $F = (f_1(\mathbf{x}), f_2(\mathbf{x}))$, where

$$f_1(\mathbf{x}) = \sum_{i=1}^{2} \left(-10 \exp \left(-0.2 \sqrt{x_i^2 + x_{i+1}^2} \right) \right), \tag{19}$$

$$f_2(\mathbf{x}) = \sum_{i=1}^{3} \left(|x_i|^{0.8} + 5 \sin(x_i^3) \right), \tag{20}$$

subject to

$$-5 \leq x_i \leq 5, \qquad i = 1, 2, 3$$

This problem is difficult because it presents a discontinuous non-convex Pareto front. Note that this problem is discontinuous both in the objective function space and in decision variable space.

6.3.2. Test suit ZDT

The next three problems belong to the test suite proposed by Zitzler et al. [52]. These problems are designed with emphasis on some characteristics that traditionally cause difficulties to any MOEA. Problem ZDT1 presents a convex Pareto front, problem ZDT2 has a non-convex Pareto front, and problem ZDT3 presents a discontinuous Pareto front.

- This problem has a convex Pareto front:

Minimize $F = (f_1(\mathbf{x}), f_2(\mathbf{x}))$, where

$$f_1(\mathbf{x}) = x_1, \tag{21}$$

$$f_2(\mathbf{x}) = g(\mathbf{x})\left(1 - \sqrt{f_1/g(\mathbf{x})}\right), \tag{22}$$

$$g(\mathbf{x}) = 1 + 9 \times \sum_{i=2}^{m} \frac{x_i}{m-1}, \tag{23}$$

where $m = 30$ and $x_i \in [0, 1]$ for $i = 1, \ldots, m$. The true Pareto front is formed with $g(\mathbf{x}) = 1$.

- This problem has a non-convex Pareto front:

  Minimize $F = (f_1(\mathbf{x}), f_2(\mathbf{x}))$, where

$$f_1(\mathbf{x}) = x_1, \tag{24}$$

$$f_2(\mathbf{x}) = g(\mathbf{x})\left(1 - (f_1/g(\mathbf{x}))^2\right), \tag{25}$$

$$g(\mathbf{x}) = 1 + 9 \times \sum_{i=2}^{m} \frac{x_i}{m-1}, \tag{26}$$

  where $m = 30$ and $x_i \in [0,1]$ for $i = 1, \ldots, m$. The true Pareto front is formed with $g(\mathbf{x}) = 1$.

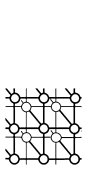
- This problem has a Pareto front composed of five discontinuous convex regions:

  Minimize $F = (f_1(\mathbf{x}), f_2(\mathbf{x}))$, where

$$f_1(\mathbf{x}) = x_1, \tag{27}$$

$$f_2(\mathbf{x}) = g(\mathbf{x})\left(1 - \sqrt{f_1/g(\mathbf{x})} - \frac{f_1}{g(\mathbf{x})}\sin(10\pi f_1)\right), \tag{28}$$

$$g(\mathbf{x}) = 1 + 9 \times \sum_{i=2}^{m} \frac{x_i}{m-1}, \tag{29}$$



where $m = 30$ y $x_i \in [0, 1]$ for $i = 1, \ldots, m$. The true Pareto front is formed with $g(\mathbf{x}) = 1$. The sine function in f_2 produces a discontinuity. Nonetheless, there are no discontinuities in the decision variable space.

6.3.3. Test problem OSY

Osyczka and Kundu [36] suggested the next problem which has six decision variables and six side constraints:

Minimize $F = (f_1(\mathbf{x}), f_2(\mathbf{x}))$, where

$$\begin{aligned} f_1(\mathbf{x}) &= -(25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2 + (x_4 - 4)^2 + (x_5 - 1)^2), \\ f_2(\mathbf{x}) &= x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2. \end{aligned}$$

Subject to

$$\begin{aligned} 0 &\leq x_1, x_2, x_6 \leq 10, \\ 1 &\leq x_3, x_5 \leq 5, \\ 0 &\leq x_4 \leq 6, \end{aligned}$$

and

$$
\begin{aligned}
0 &\leq x_1 + x_2 - 2, \\
0 &\leq 6 - x_1 - x_2, \\
0 &\leq 2 - x_2 + x_1, \\
0 &\leq 2 - x_1 + 3x_2, \\
0 &\leq 4 - (x_3 - 3)^2 - x_4, \\
0 &\leq (x_5 - 3)^2 + x_6 - 4
\end{aligned}
$$

Both P_{true} and PF_{true} are disconnected in problem OSY. Its Pareto front is composed by five segments situated in some frontier imposed by the constraints. This problem is difficult because it requires a MOEA to generate and maintain individuals on the boundaries between the feasible and the infeasible region.

6.3.4. Test problem KIT

The problem KIT was introduced in [27]:

Maximize $F = (f_1(x, y), f_2(x, y))$, where

$$f_1(x, y) = -x^2 + y, \tag{30}$$

$$f_2(x, y) = \frac{1}{2}x + y + 1. \tag{31}$$

Subject to

$$x, y \geq 0,$$

$$0 \geq \frac{1}{6}x + y - \frac{13}{2}, \quad (32)$$

$$0 \geq \frac{1}{2}x + y - \frac{15}{2}, \quad (33)$$

$$0 \geq 5x + y - 30. \quad (34)$$

$$(35)$$

This problem is characterized by being concave and disconnected both in P_{true} and PF_{true}. The main difficulty of KIT is that members of P_{true} are situated on the boundary between the feasible and the infeasible regions.

6.4. The reference pMOEA

As a reference point to evaluate the performance of our proposal, we consider an island model implementation of the NSGA-II (called PNSGA-II). According to a study done by von Lucken [48], in which several modern MOEAs were involved, the parallel counterpart of the NSGA-II was the best algorithm in most of the test problems adopted, and therefore our motivation to adopt it in as a basis for comparison in our own work.

The only extensions done to the NSGA-II are the operations to exchange individuals and the mechanism to build the PF_{known} set from the $PF_{known}\binom{p}{t}$ set of each processor p.

The migration topology of this algorithm is a ring. In the algorithm, x individuals taken from the first front of the non-dominated sorted population are migrated, and x randomly selected individuals are replaced.

To present the final PF_{known}, we used a similar procedure to that used by the NSGA-II to choose the new population from the combination of the parent and offspring populations. In

the case of the PNSGA-II, the root processor combines the $PF_{known}^{(p)}_t$ sets of each processor p. The resulting population is sorted into different non-dominated levels. To construct the PF_{known} set from the best front, we consecutively choose each front until the population is completed. If the size of the last front exceeds the size of the population, then this front is sorted according to the $\prec_n$ operator defined in [15] and then the best solutions are chosen to complete the PF_{known} set.

7. RESULTS AND DISCUSSION

Besides showing the viability of the proposed approach, we are interested in analyzing the gain in convergence when increasing the number of processors. For this purpose, we set the sub-population size of each processor to the minimum allowable such that the algorithms still achieve an acceptable effectiveness. In this way, we may add as many processors as desired, while maintaining a good effectiveness of the algorithm. In the experiments performed, both algorithms were run 30 times (using different random seeds for the initial populations in each case), varying the number of processors (we used 1, 4, 8, 12 and 16 processors). Also, to allow a fair comparison of results (and be able to assess the effectiveness of each approach), for each run, the total number of evaluations was fixed to the same value. For each number of processors, the performance of both algorithms is compared using all the metrics described in the previous section. For each algorithm, the best parameter setting was determined experimentally. For the sake of a fair comparison, we used the same parameter settings in all problems. In Table I, we show the parameter values used in all the test problems adopted.

	MRMOGA	PNSGA-II
Parameter	**value**	**value**
Epochs	60	24
Generations per epoch	10	15
Island size	30	50
Number of migrants	20	25
Evaluations per island	18000	18000
Known Pareto front size	100	100
Decimals of precision	5	5
Crossover rate	0.9	0.9
Mutation rate	$1/\ell_i$	$1/k$

Table I. Paremeters used in all the experiments performed.

Characteristic	**Description**
CPU	Intel Xeon; 2.45MHz
Number of nodes	16 (2 processors per node)
Memory	2 GB per node
Operating System	Red Hat Linux 3.2.2-5
Communications network	FastEthernet
Communications library	MPICH 1.12

Table II. Characteristics of the cluster used in the experiments.

For the PNSGA-II, k is the required chromosome length to represent solutions with 5 decimals of precision. For our MRMOGA, ℓ_i is the chromosome length corresponding to the resolution of each island. For the highest resolution island we have that $\ell_{hi} = k$.

The PNSGA-II was implemented using the source code provided by Prof. Kalyanmoy Deb[¶]. The MRMOGA implementation was written in C++. For the inter-process communication, we used the MPICH library. The simulation runs were carried out in a cluster of 16 dual nodes which has the characteristics described in Table II.

[¶]The NSGA-II source code is available at http://www.iitk.ac.in/kangal/soft.htm.

7.1. Results for problem KUR

As can be seen in Figure 11(a), using less than 12 processors, the PNSGA-II achieves the best results in SC. But this is not sufficient to conclude that the PNSGA-II has better convergence than our MRMOGA, since for every number of processors, our MRMOGA yields, by far, the best values for IGD (Fig. 11(b)). Furthermore, it is worth mentioning that the MRMOGA convergence improves as more processors are used. On the contrary, based on SC, the convergence of the PNSGA-II reached its maximum at around 4 processors and then began to decrease. The best SC value was obtained by MRMOGA-16 followed by PNSGA-II-4. The results of the $\mathcal{C}$ metric (see Table III) agree with those of the SC metric in the sense that the fraction of weakly dominated solutions of the PNSGA-II augments after 4 processors, whereas the fraction of covered solutions of the MRMOGA has a tendency to drop as more processors are used (excluding 1 to 4). However, for any number of processors, the PNSGA-II-n ($n = 1, .., 16$) weakly dominates a greater fraction of the MRMOGA-n outcomes than when doing the comparison in the opposite direction. In this case, there is no clear evidence that one algorithm outperforms the other, although our MRMOGA seems to improve its convergence steadily. Considering a cost/performance tradeoff, the PNSGA-II is the winner since it achieves a good convergence using few processors. On the other hand, if we need leading effectiveness and we can afford the use of more processors, our MRMOGA will certainly be a better choice.

With respect to distribution, it is important to note that our MRMOGA covers all the regions composing the Pareto-optimal front of KUR (see Figure 12). Conversely, independently of the number of processors, the PNSGA-II is not able to totally cover the bottom right region of PF_{true} nor the point $(-20, 0)$ (see Figure 13). The metric IGD reflects this situation. The

$\mathcal{C}(M,P)$	PNSGA-II-1	PNSGA-II-4	PNSGA-II-8	PNSGA-II-12	PNSGA-II-16
MRMOGA-s-1	0.394667	0.270000	0.277667	0.309667	0.306667
MRMOGA-s-4	0.416000	0.280000	0.286000	0.318667	0.313000
MRMOGA-s-8	0.454667	0.357000	0.357000	0.385333	0.384000
MRMOGA-s-12	0.463333	0.350000	0.369000	0.379000	0.385667
MRMOGA-s-16	0.474667	0.374333	0.389000	0.404000	0.405333
$\mathcal{C}(P,M)$	MRMOGA-s-1	MRMOGA-s-4	MRMOGA-s-8	MRMOGA-s-12	MRMOGA-s-16
PNSGA-II-1	0.505333	0.453333	0.373000	0.356667	0.321667
PNSGA-II-4	0.587000	0.542667	0.482667	0.470667	0.444333
PNSGA-II-8	0.610667	0.557667	0.499000	0.492333	0.448333
PNSGA-II-12	0.625333	0.573000	0.506000	0.501000	0.456000
PNSGA-II-16	0.644667	0.594333	0.527333	0.513667	0.474667

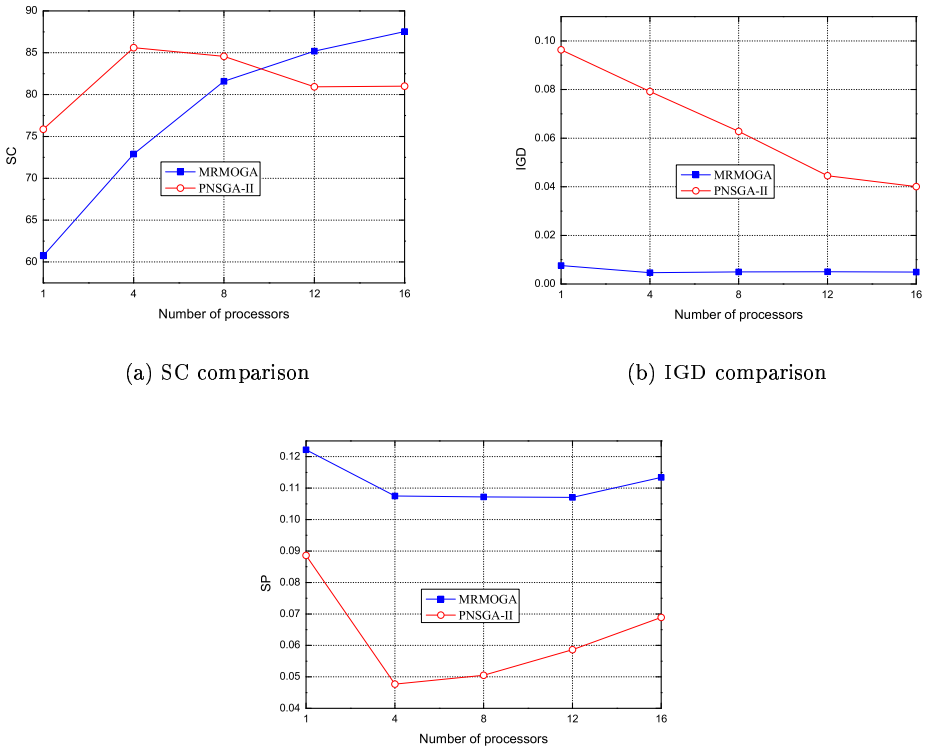
Table III. Direct comparison between our MRMOGA and the PNSGA-II using the $\mathcal{C}$ metric for problem KUR (in the coverage tables M stands for MRMOGA and P for PNSGA).

vectors of the regions of PF_{true} not covered by the PNSGA-II are far away from the nearest vectors of its PF_{known}, which produces substantially worse values than those achieved by our MRMOGA, which is able to cover all regions of PF_{true}.

In terms of SP, the PNSGA-II provided a better distribution of solutions (Fig. 11(c)). However, it is worth noting that the distance from vector $(-20, 0)$ to all other vectors is greater than the average distance of the vectors in PF_{true}. Since the PNSGA-II, on average, is not capable of finding this vector, its value on SP does not get affected.

7.2. Results for Test Suite ZDT

The intrinsic size of the search space in these three problems is $2^{510} \approx 3.352 \times 10^{153}$, considering the use of binary encoding and the decimals of precision adopted to discretize the decision variables. The overall performance (regarding all metrics) of each of the algorithms is similar both in ZDT1 and ZDT2. From the SC graphs illustrated in Figures 14(a) and 17(a) it is clear that our MRMOGA outperforms the PNSGA-II when more than one processor is used. The



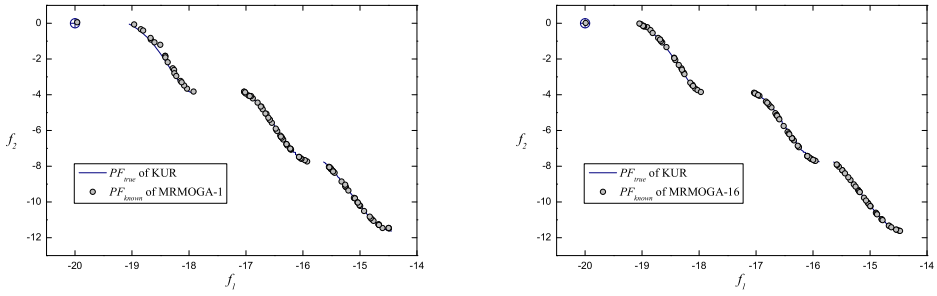
(a) SC comparison

(b) IGD comparison

(c) SP comparison

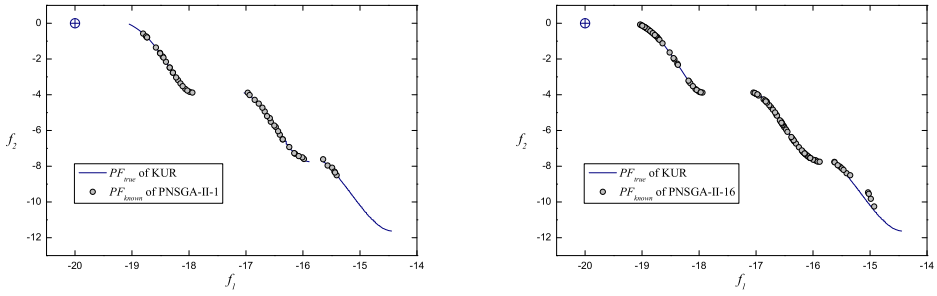
Figure 11. Effectiveness of MRMOGA and PNSGA-II according to the number of processors for the problem KUR.

IGD values (Figures 14(b), 17(b) and 20(b)) of both algorithms are very similar for the three problems, which indicates that the algorithms converged very closely to the true Pareto front. With one processor, our MRMOGA has a poor performance in both problems as reflected by the $\mathcal{C}$ metric presented in Tables IV and V. On problems ZDT1 and ZDT2, the non-dominated fronts reported by MRMOGA-S-1 are 100% covered by PNSGA-II-1, while the former covers



(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 12. PF_{known} obtained by our MRMOGA using 1 and 16 processors for the problem KUR.



(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 13. PF_{known} obtained by the PNSGA-II using 1 and 16 processors for the problem KUR.

0% of the PNSGA-II-1 outcomes. However, when more processors are used, the improvement of our MRMOGA is remarkable with respect to all the metrics adopted, especially regarding convergence since from 1 to 4 processors SC is improved about 100 times in ZDT1 and around 1000 times in ZDT2. It is important to note that when our MRMOGA uses one processor, there is no division of the search space and, therefore, its poor performance is somehow expected in this case. After one processor, our MRMOGA substantially outperforms the PNSGA-II both in ZDT1 and ZDT2. Comparing the best results achieved by each algorithm for ZDT1 (Table IV), PNSGA-II-16 weakly dominates less than 6% of the MRMOGA-s-16 outcomes. In contrast, the non-dominated fronts produced by the latter cover about 65% of the PNSGA-II-16 outcomes. A similar behavior is observed using 4 to 12 processors. In ZDT2, the difference in convergence is more noticeable. For example, PNSGA-II-16 (its best performance) weakly dominates at most 21% of the MRMOGA-s-4 outcomes (its worst performance using more than one processor). In contrast, using only 4 processors, MRMOGA-s-4 covers about 74% of the generated fronts of PNSGA-II-16. This difference in convergence is practically impossible to detect from visually examining the PF_{known} sets. However, the magnification of the region shown in Figures 15(b) and 16(b) allows us to identify this difference.

In ZDT3, a similar situation to that observed in KUR occurs. With respect to SC (Fig. 20(a)), our MRMOGA has better convergence than the PNSGA-II after 8 processors are used. Considering SC and the $\mathcal{C}$ comparison presented in Table VI, it can be noted that only when our MRMOGA uses 1 or 4 processors, the PNSGA-II performs better. When both algorithms use 8 or more processors, our MRMOGA weakly dominates a larger fraction of the generated fronts of the PNSGA-II. Again, it is interesting to see the improvement in the convergence of our MRMOGA

$\mathcal{C}(M,P)$	PNSGA-II-1	PNSGA-II-4	PNSGA-II-8	PNSGA-II-12	PNSGA-II-16
MRMOGA-s-1	0.000668	0.000667	0.000000	0.000000	0.000000
MRMOGA-s-4	0.923848	0.703667	0.540000	0.517667	0.506000
MRMOGA-s-8	0.939880	0.735333	0.601000	0.577333	0.562000
MRMOGA-s-12	0.949900	0.750000	0.636333	0.591000	0.577667
MRMOGA-s-16	0.965932	0.794667	0.705333	0.659333	0.650667
$\mathcal{C}(P,M)$	MRMOGA-s-1	MRMOGA-s-4	MRMOGA-s-8	MRMOGA-s-12	MRMOGA-s-16
PNSGA-II-1	1.000000	0.221333	0.019667	0.006667	0.005333
PNSGA-II-4	1.000000	0.363000	0.077000	0.050333	0.033333
PNSGA-II-8	1.000000	0.399000	0.101000	0.069333	0.044000
PNSGA-II-12	1.000000	0.406667	0.099000	0.073667	0.051000
PNSGA-II-16	1.000000	0.416667	0.111000	0.080000	0.056000

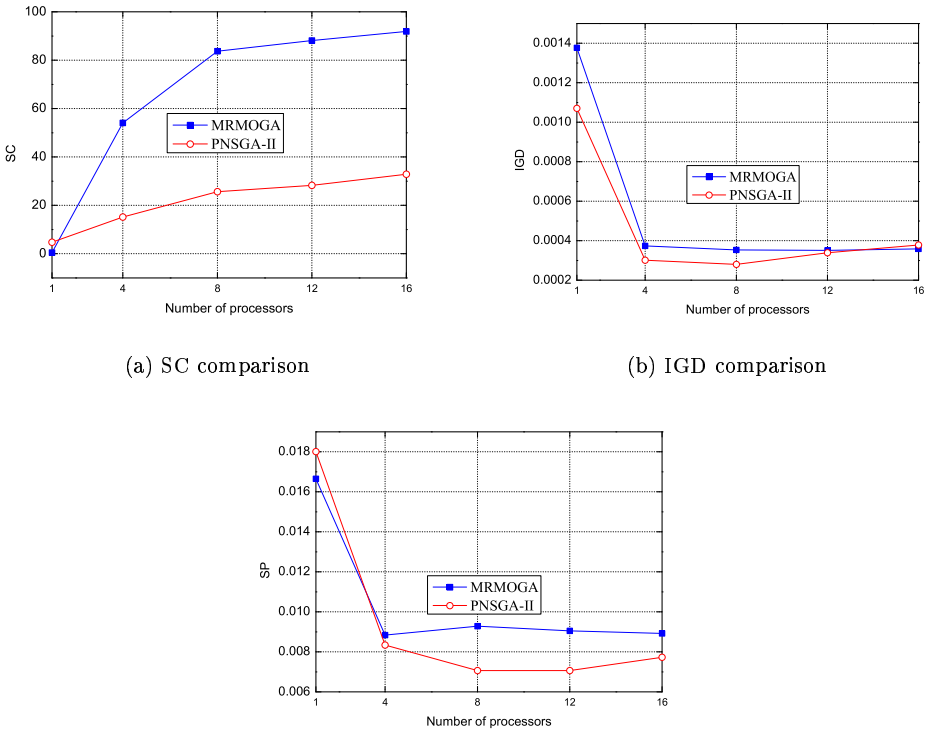
Table IV. Direct comparison between our MRMOGA and the PNSGA-II using the $\mathcal{C}$ metric for problem ZDT1.

$\mathcal{C}(M,P)$	PNSGA-II-1	PNSGA-II-4	PNSGA-II-8	PNSGA-II-12	PNSGA-II-16
MRMOGA-s-1	0.005394	0.000667	0.000000	0.000000	0.000000
MRMOGA-s-4	0.997303	0.885000	0.806667	0.788000	0.753333
MRMOGA-s-8	0.992583	0.890333	0.789667	0.767667	0.731667
MRMOGA-s-12	0.987862	0.867333	0.766667	0.744667	0.733667
MRMOGA-s-16	0.991234	0.891333	0.790000	0.760000	0.736000
$\mathcal{C}(P,M)$	MRMOGA-s-1	MRMOGA-s-4	MRMOGA-s-8	MRMOGA-s-12	MRMOGA-s-16
PNSGA-II-1	1.000000	0.069667	0.004333	0.001000	0.000333
PNSGA-II-4	1.000000	0.159000	0.024000	0.011000	0.005667
PNSGA-II-8	1.000000	0.196667	0.041667	0.020667	0.011000
PNSGA-II-12	1.000000	0.197000	0.043333	0.021333	0.011333
PNSGA-II-16	1.000000	0.206000	0.048333	0.028333	0.010667

Table V. Direct comparison between our MRMOGA and the PNSGA-II using the $\mathcal{C}$ metric for problem ZDT2.

when using more than one processor (Figures 20(a) and 20(b)). As in the problem KUR, here we must take in mind a cost/performance tradeoff. But this time, our MRMOGA achieves an acceptable performance after using 8 processors instead of 12.

Concerning the metric SP, the PNSGA-II performs better than our MRMOGA in the three problems. However, as can be seen in the plots of Figure 21 the distribution of our MRMOGA is very competitive. Also, it is noted that both algorithm cover all regions of the true Pareto front.



(a) SC comparison

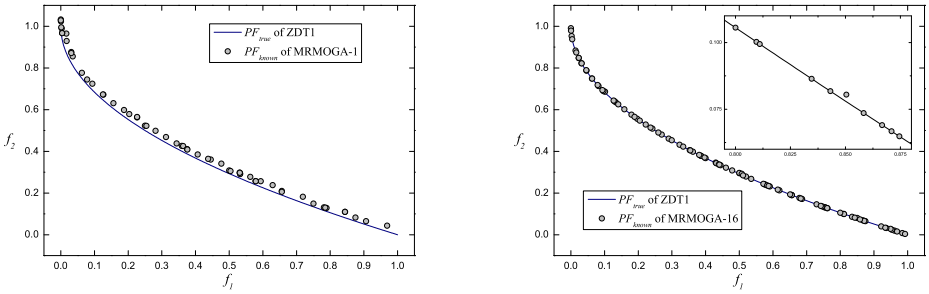
(b) IGD comparison

(c) SP comparison

Figure 14. Effectiveness of our MRMOGA and the PNSGA-II according to the number of processors for the problem ZDT1.

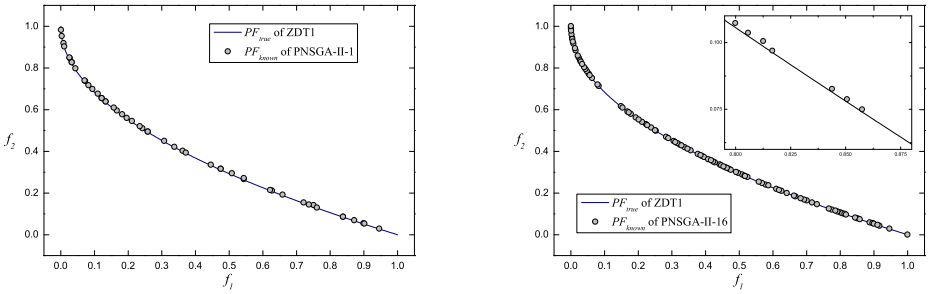
7.3. Results for Test problem OSY

In the plots of Figure 24, we show that our MRMOGA is able to maintain vectors near to all regions composing the Pareto-optimal front of OSY. On the contrary, the PNSGA-II, for some runs, does not converge to certain regions of the front, particularly to the segment DE (Fig. 25). Additionally, based on SC, our MRMOGA outperforms the PNSGA-II for every number



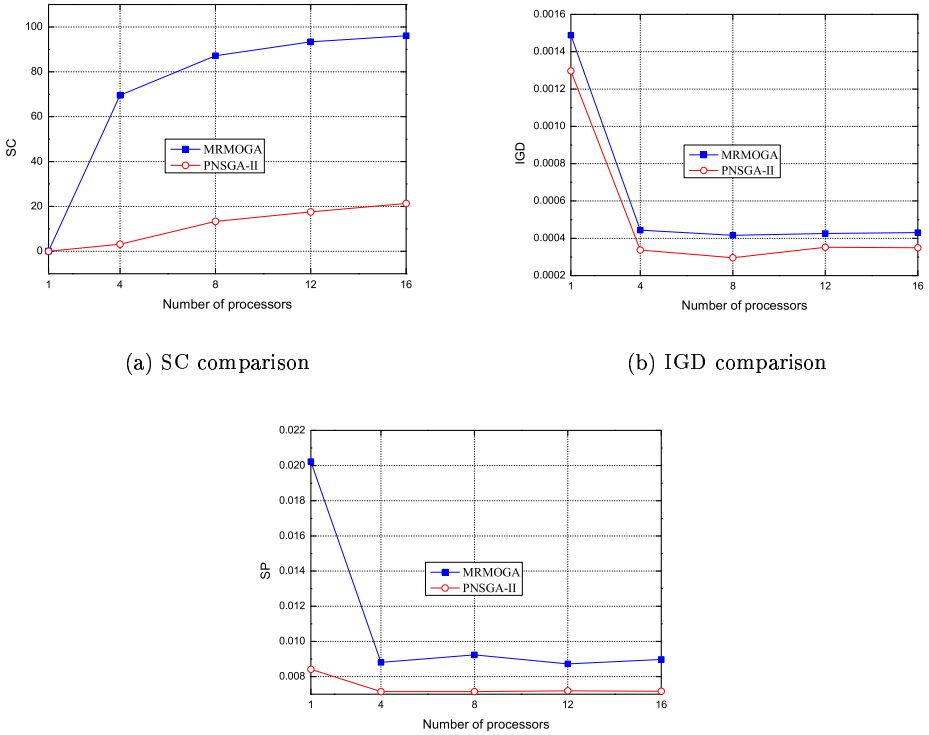
(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 15. PF_{known} obtained by our MRMOGA using 1 and 16 processors for the problem ZDT1.



(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 16. PF_{known} obtained by the PNSGA-II using 1 and 16 processors for the problem ZDT1.



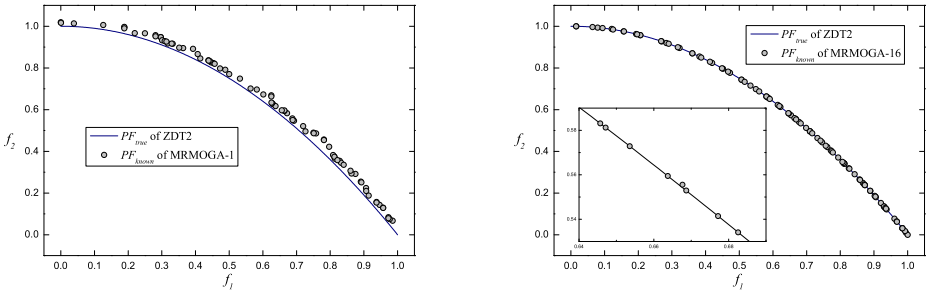
(a) SC comparison

(b) IGD comparison

(c) SP comparison

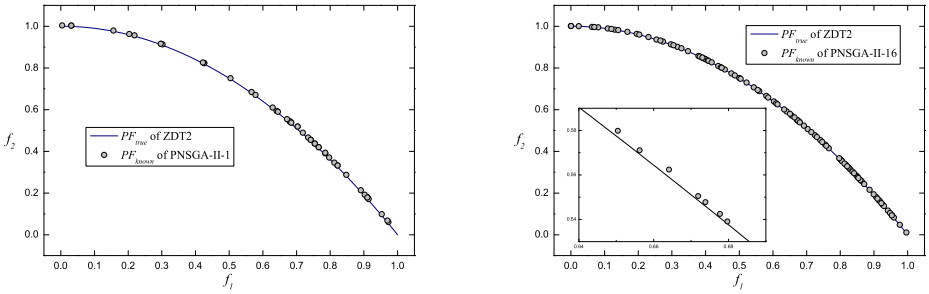
Figure 17. Effectiveness of our MRMOGA and the PNSGA-II according to the number of processors for the problem ZDT2.

of processors adopted, since the former obtains a larger number of vectors belonging to PF_{true} (Fig. 23(a)). Nevertheless, it is interesting to note that the PNSGA-II outperforms our MRMOGA in the $\mathcal{C}$ comparison (see Table VII). For instance, the fronts of MRMOGA-S-n weakly dominate around 50% of the PNSGA-II-16 outcomes (its best assessment), whereas PNSGA-II-n, covers at most 84% of the non-dominated fronts of MRMOGA-S-16 (its best assessment). A possible



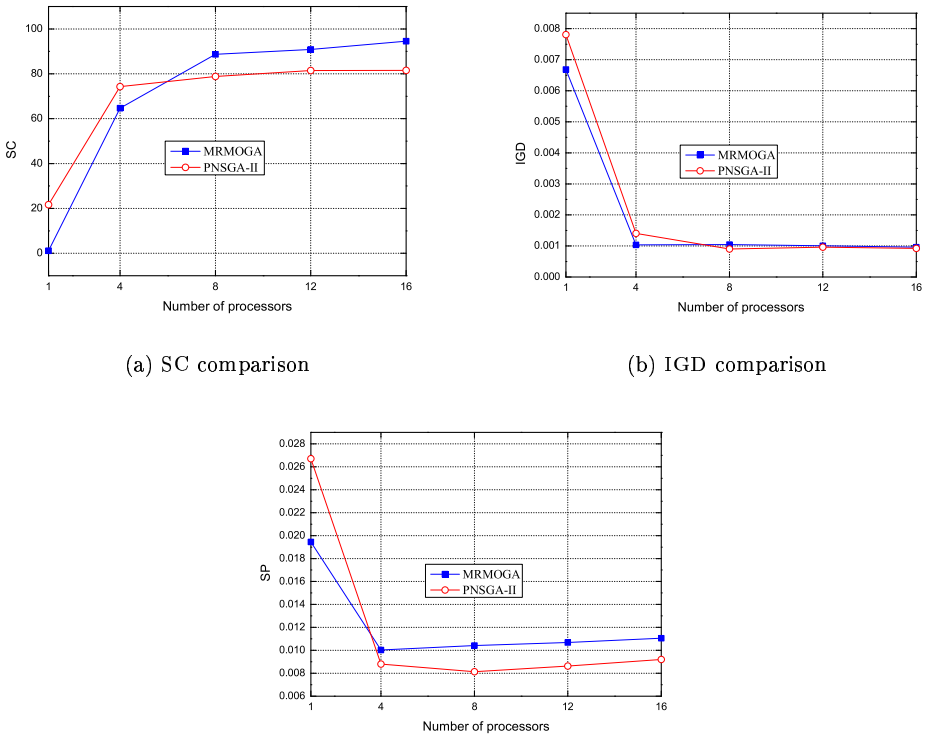
(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 18. PF_{known} obtained by our MRMOGA using 1 and 16 processors for the problem ZDT2.



(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 19. PF_{known} obtained by the PNSGA-II using 1 and 16 processors for the problem ZDT2.



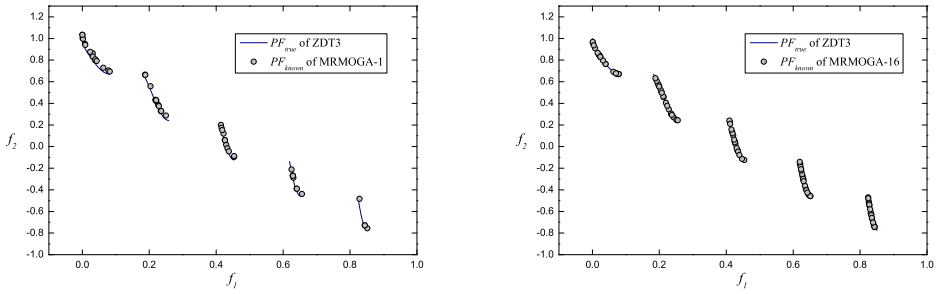
(a) SC comparison

(b) IGD comparison

(c) SP comparison

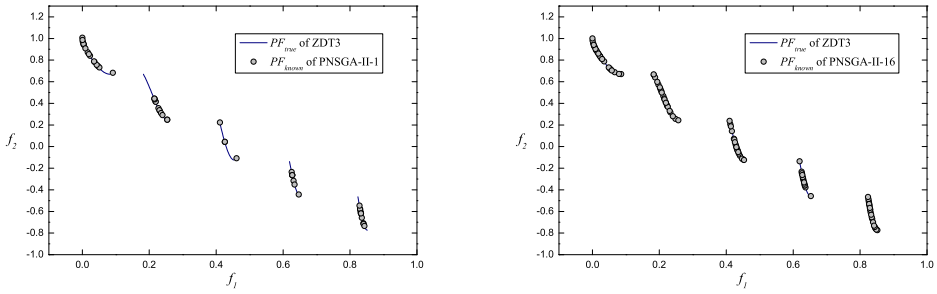
Figure 20. Effectiveness of our MRMOGA and the PNSGA-II according to the number of processors for the problem ZDT3.

reason for this discrepancy is that the vectors of PF_{known} reported by our MRMOGA and that do not belong to PF_{true} (and maybe also some that do) are weakly dominated by the PF_{known} set generated by the PNSGA-II. This conjecture is reinforced by the fact that the values of IGD obtained by the PNSGA-II are marginally better than those achieved by our MRMOGA (Figure 23(b)).



(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 21. PF_{known} obtained by our MRMOGA using 1 and 16 processors for the problem ZDT3.



(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 22. PF_{known} obtained by the PNSGA-II using 1 and 16 processors for the problem ZDT3.

$\mathcal{C}(M,P)$	PNSGA-II-1	PNSGA-II-4	PNSGA-II-8	PNSGA-II-12	PNSGA-II-16
MRMOGA-s-1	0.029074	0.005667	0.000333	0.000000	0.000000
MRMOGA-s-4	0.669371	0.339333	0.275667	0.229000	0.237333
MRMOGA-s-8	0.826910	0.500333	0.401333	0.346000	0.341667
MRMOGA-s-12	0.818796	0.525333	0.442000	0.409667	0.391333
MRMOGA-s-16	0.818120	0.530333	0.435667	0.390000	0.384667
$\mathcal{C}(P,M)$	MRMOGA-s-1	MRMOGA-s-4	MRMOGA -s-8	MRMOGA-s-12	MRMOGA-s-16
PNSGA-II-1	0.999466	0.367000	0.080000	0.044000	0.018667
PNSGA-II-4	1.000000	0.543667	0.229000	0.164333	0.073333
PNSGA-II-8	1.000000	0.561667	0.259667	0.188333	0.095667
PNSGA-II-12	1.000000	0.578667	0.280333	0.208333	0.098667
PNSGA-II-16	1.000000	0.580000	0.274667	0.208000	0.103333

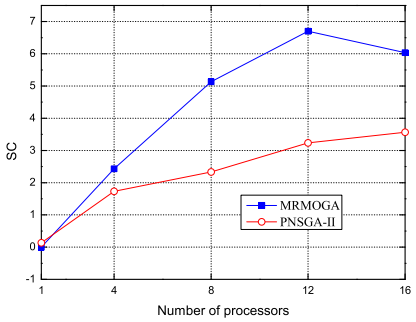
Table VI. Direct comparison between our MRMOGA and the PNSGA-II using the $\mathcal{C}$ metric for problem ZDT3.

$\mathcal{C}(M,P)$	PNSGA-II-1	PNSGA-II-4	PNSGA-II-8	PNSGA-II-12	PNSGA-II-16
MRMOGA-s-1	0.767845	0.398333	0.168333	0.146000	0.120000
MRMOGA-s-4	0.869246	0.525333	0.312333	0.271000	0.251667
MRMOGA-s-8	0.934623	0.676333	0.549333	0.436000	0.439000
MRMOGA-s-12	0.931955	0.673667	0.548667	0.443000	0.441667
MRMOGA-s-16	0.947298	0.713000	0.612333	0.479000	0.490667
$\mathcal{C}(P,M)$	MRMOGA-s-1	MRMOGA-s-4	MRMOGA-s-8	MRMOGA-s-12	MRMOGA-s-16
PNSGA-II-1	0.977097	0.702000	0.510333	0.393333	0.335000
PNSGA-II-4	0.991896	0.876333	0.821333	0.752333	0.733333
PNSGA-II-8	0.995419	0.916667	0.853333	0.810000	0.785667
PNSGA-II-12	0.994715	0.902000	0.872667	0.826667	0.817667
PNSGA-II-16	0.996124	0.931667	0.879000	0.837333	0.837000

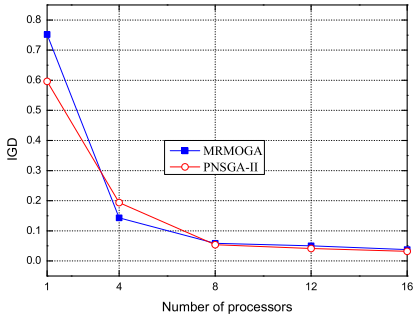
Table VII. Direct comparison between our MRMOGA and the PNSGA-II using the $\mathcal{C}$ metric for problem OSY.

It is worth mentioning that when using only one processor (that is, without exploring in multiple resolutions), our MRMOGA was not able to find any vector that belongs to PF_{true}. Conversely, with more than one processor, our MRMOGA reported some solutions that belong to PF_{true}. This is an indication that searching through different resolutions allows us to yield solutions that would not be possible (or easy) to find without this scheme.

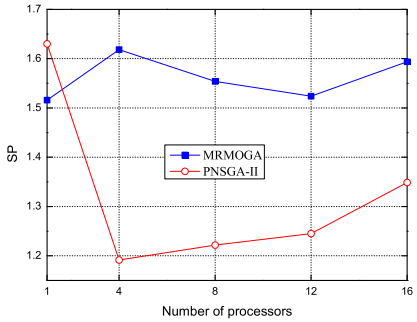
With respect to the distribution, the PNSGA-II obtains the best results in SP (Figure 23(c)), although our MRMOGA remains competitive as can be seen in the graphs of the Figure 24.



(a) SC comparison



(b) IGD comparison

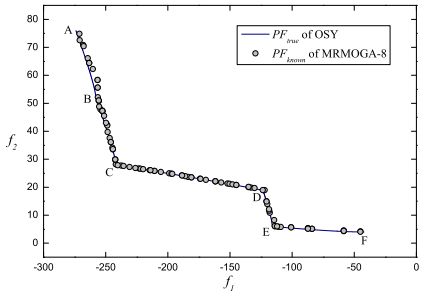


(c) SP comparison

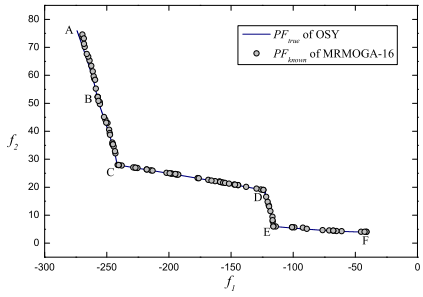
Figure 23. Effectiveness of our MRMOGA and the PNSGA-II according to the number of processors for the problem OSY.

7.4. Results for Test problem KIT

In this problem, according to the *SC* and *IGD* metrics, our MRMOGA showed better convergence than the PNSGA-II beginning from one processor as can be seen in Figure 26. Using one processor, MRMOGA approaches quickly to the Pareto front as revealed by the

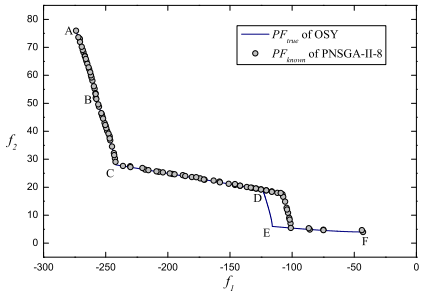


(a) PF_{known} obtained with 8 processor.

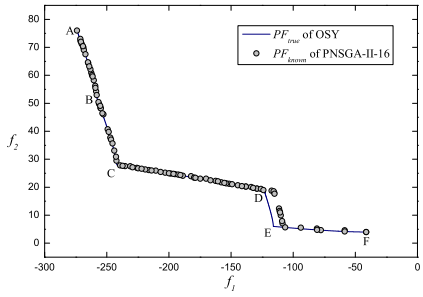


(b) PF_{known} obtained with 16 processors.

Figure 24. PF_{known} obtained by our MRMOGA using 1 and 16 processors for the problem OSY.



(a) PF_{known} obtained with 8 processor.



(b) PF_{known} obtained with 16 processors.

Figure 25. PF_{known} obtained by the PNSGA-II using 8 and 16 processors for the problem OSY.

$\mathcal{C}(M,P)$	PNSGA-II-1	PNSGA-II-4	PNSGA-II-8	PNSGA-II-12	PNSGA-II-16
MRMOGA-s-1	0.880667	0.315667	0.222000	0.245667	0.211333
MRMOGA-s-4	0.964000	0.667333	0.557000	0.564333	0.537667
MRMOGA-s-8	0.967333	0.711667	0.620667	0.633333	0.599333
MRMOGA-s-12	0.977333	0.745333	0.668667	0.674667	0.649333
MRMOGA-s-16	0.980667	0.779000	0.703000	0.711000	0.690000
$\mathcal{C}(P,M)$	MRMOGA-s-1	MRMOGA-s-4	MRMOGA-s-8	MRMOGA-s-12	MRMOGA-s-16
PNSGA-II-1	0.744000	0.105667	0.037667	0.023000	0.015000
PNSGA-II-4	0.928333	0.592333	0.446667	0.327667	0.290667
PNSGA-II-8	0.946000	0.678000	0.523000	0.418000	0.382667
PNSGA-II-12	0.942667	0.675667	0.532667	0.406667	0.377333
PNSGA-II-16	0.955667	0.711667	0.553333	0.432000	0.395000

Table VIII. Direct comparison between our MRMOGA and the PNSGA-II using the $\mathcal{C}$ metric for problem KIT.

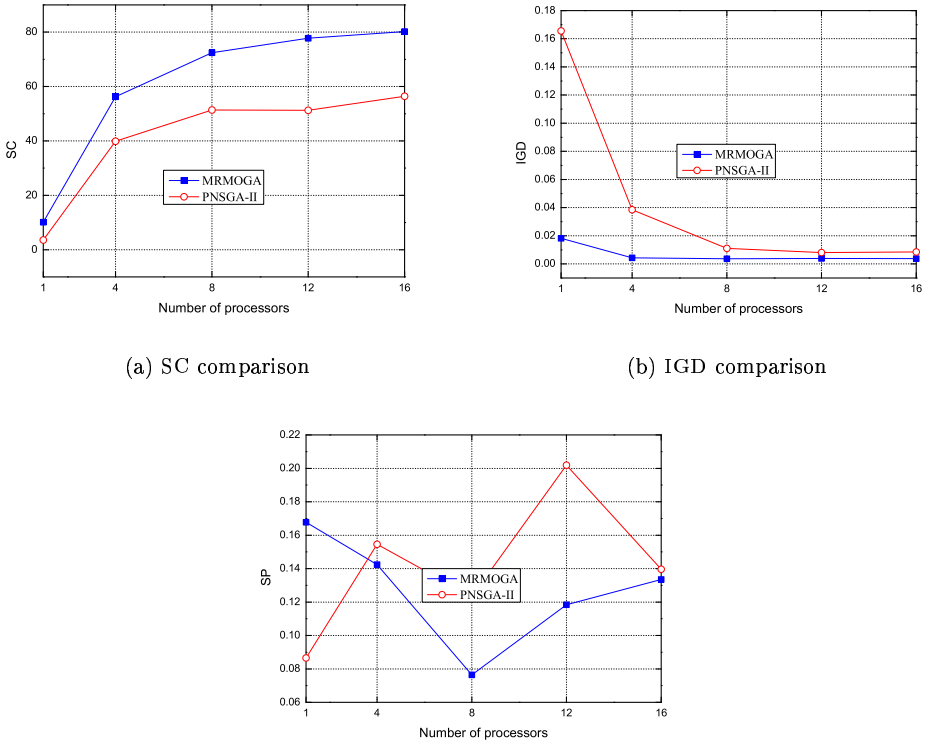
comparison of the Figures 27(a) and 28(a) corresponding to the PF_{known} sets generated by each algorithm. This means that our MRMOGA is not affected by the fact that elements in PF_{true} of KIT are situated in the boundary of the feasible region. In the direct comparison with respect to the $\mathcal{C}$ metric (Table VIII), our MRMOGA also clearly surpasses the PNSGA-II. This problem is the only one, among the problems used in this study[‖], where our MRMOGA performs better using one processor.

Based on the visual comparison between the non-dominated fronts obtained by each algorithm and on the results in SP of Figure 26(c), we can state that our MRMOGA achieves a better distribution.

7.5. Methodology to Evaluate the Efficiency

In the experiments presented in this section, the metrics will be calculated based on the orthodox speedup. To define the stopping criterion required for calculating this speedup we

[‖]Alongside the six problems of this paper, three more problems were considered in the experiments but not included here.



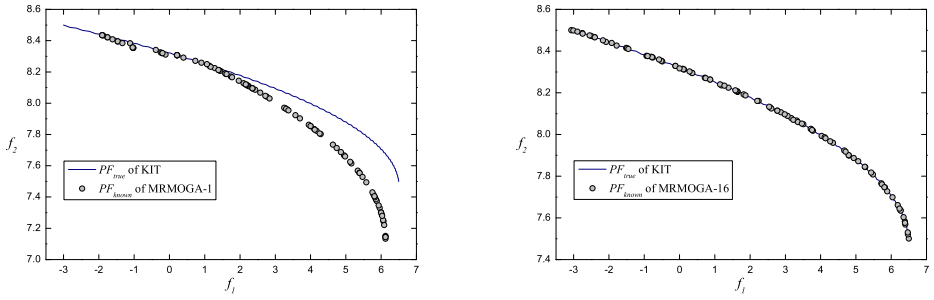
(a) SC comparison

(b) IGD comparison

(c) SP comparison

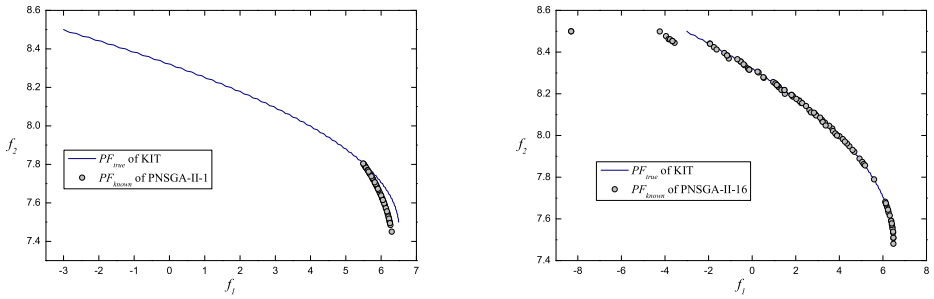
Figure 26. Effectiveness of our MRMOGA and the PNSGA-II according to the number of processors for the problem KIT.

must determine how to measure the "quality" of a PF_{known} set. For this purpose, we use the hypervolume metric (HV) suggested in [51], which computes the volume of the region dominated by PF_{known}. In these experiments, the pMOEA is stopped when the value of HV is equal or greater than a value previously set.



(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 27. PF_{known} obtained by our MRMOGA using 1 and 16 processors for the problem KIT.



(a) PF_{known} obtained with 1 processor. (b) PF_{known} obtained with 16 processors.

Figure 28. PF_{known} obtained by the PNSGA-II using 1 and 16 processors for the problem KIT.

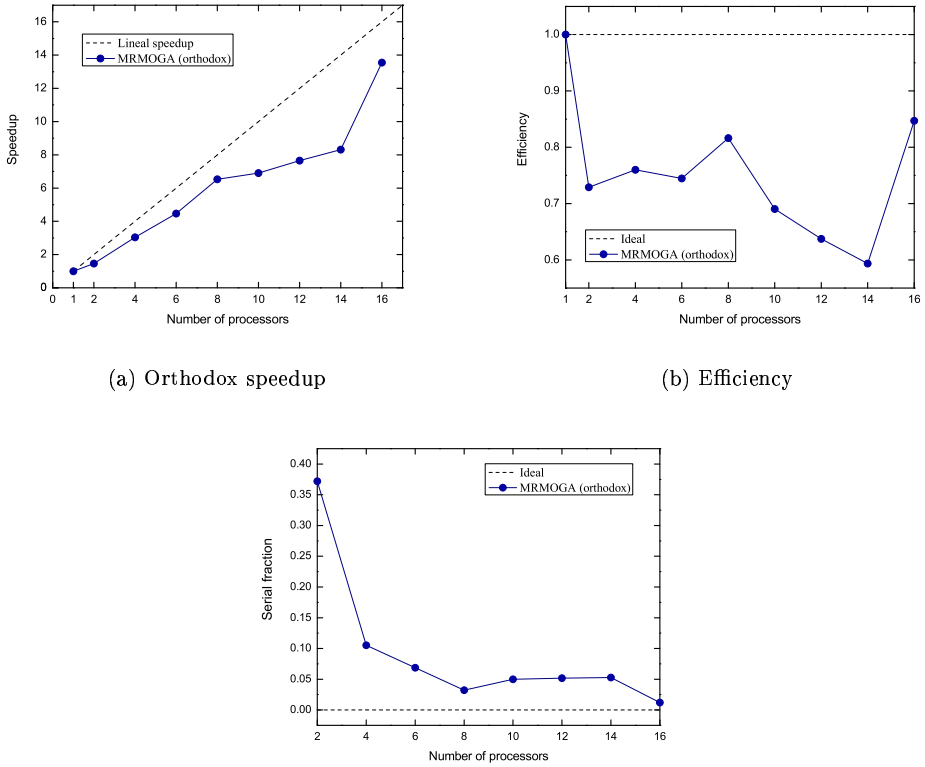
If we want to obtain precisely the HV value of the $PF_{known}(t)$, it is necessary to gather all $PF_{known}\binom{p}{t}$ sets from each processor p after each generation t. Nonetheless, this procedure introduces an overhead due to the communications involved. This overhead is not present in the serial algorithm which results in an unfair comparison. To overcome this problem, in our experiments we considered that the $PF_{known}\binom{p}{t}$ set of the higher resolution island reflects closely to PF_{known}. Therefore, the HV metric is calculated only for the highest resolution island. In this way, the cost to compute HV is compensated in the ratio of the speedup. Unfortunately, for the PNSGA-II we cannot proceed in a similar way, since all the islands equally contribute to form the known Pareto front. As a consequence, we decided not to include in the paper an efficiency comparison between the PNSGA-II and our MRMOGA.

In the experiments described in the next sections, only the strict topology is considered. Likewise, the metrics were computed only for the ZDT1 problem. A study considering more problems is left for a future work. The orthodox speedup of our MRMOGA is calculated in the following way: the serial algorithm is MRMOGA-s-16 (16 islands) running on a single processor. On the other hand, the parallel algorithm is MRMOGA-s-16 on an increasing number of processors (from 2 to 16). The population size of each island is set to 25 ($16 \times 25 = 400$ individuals) for all runs. Note that MPI assigns cyclicly the processes (islands) to the available processors. Therefore, it is possible that some processors have unequal workload depending on the number of processors. This situation is exhibited with 6 and 10 to 14 processors. For instance, with 10 processors there are 6 processors with 2 islands and 4 processors with only one island.

7.6. Evaluation Using the Orthodox Speedup

As can be seen in Figure 29(a), with any number of processors MRMOGA achieves a sublinear speedup, but it is near-linear for some number of processors. From 2 to 8 processors the speedup tends to be superlinear as reveals the F metric, since this decreases with the number of processors (Figure 29(c)). However, between 10 and 14 processors the speedup begins to decay, and for 16 processors the speedup is again near to the linear speedup. This behavior is due to the load imbalance when 10 to 14 processors are used. In a balanced situation the low resolution islands are faster than the high resolution islands, but the imbalance causes the low resolution islands to become the slowest ones (since there are two islands in the same processor). Therefore, the high resolution islands stay idle waiting for the immigrants coming from the low resolution islands, which causes the speedup drops.

If we consider a balanced environment (e.g., excluding 10 to 14 processors) the efficiency seems to get better whenever a processor is added as shows the graph in Figure 29(b). In a similar manner, F decreases with an increasing number of processors. This behavior might be caused by the addition of cache memory. Whichever this factor is, we can say that our MRMOGA has very good scalability as reveals the decrement of F. Additionally, the overhead due to communications and synchronization increases very slowly as more processors are used as shows E.



(a) Orthodox speedup

(b) Efficiency

(c) Serial fraction

Figure 29. Results of the efficiency of our MRMOGA according to the orthodox speedup, efficiency, and serial fraction for problem ZDT1.

8. CONCLUSIONS AND FUTURE WORK

The main goal of this paper was to propose a novel parallel multi-objective evolutionary algorithm whose design takes care of both effectiveness and efficiency. The proposed approach is based on the island paradigm with heterogenous nodes and is characterized by dividing

the decision variable space through an encoding of the solutions with a different resolution in each sub-population. This scheme can be seen as a division of the decision variable space. The performance assessment considered both *effectiveness* and *efficiency*. In order to evaluate the performance of our proposal, its results were compared against those obtained by a parallel version of the NSGA-II. The evaluation was carried out using 6 well-known multi-objective test problems which embody different characteristics that cause difficulties for MOEAs. The most important group of problems has a large search space that makes them well-suited to be solved by a pMOEA. Other problems were selected for having a disconnected true Pareto front that shows the capacity of a pMOEA to maintain a diverse population. Finally, the assessment considered two constrained test problems.

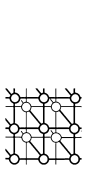
Our proposal showed a good trade-off between effectiveness and the communications overhead that impacts in the efficiency. Based on the obtained results, we can state that to divide the decision variable space with different resolutions considerably improves the effectiveness and efficiency of a parallel MOEA. In addition, the proposed algorithm has a great exploratory ability, because in all the test functions, it was able to find solutions with a better spread along the real Pareto front than the algorithm with respect to which it was compared. Also, our MRMOGA showed good ability to solve constrained MOPs, even for those problems recognized for their difficulty (such as OSY).

From the comparative study we can conclude that the proposed scheme to divide the decision variable space noticeably improves the convergence of a pMOEA. In all but one of the problems considered, the PNSGA-II performs better than our MRMOGA when using only one processor**.

**By using only one processor each serial optimizer is isolated and, in consequence, our MRMOGA does not divide the search space and the PNSGA-II is reduced to the original NSGA-II.

However, using more than one processor, our MRMOGA surpasses the PNSGA-II. Our approach was also shown to maintain a well-spread PF_{known} set, since in all the problems, it covered the entire Pareto-optimal front. Throughout the empirical study performed, we observed that the scheme of our MRMOGA has more value in problems with large search spaces such as the ZDT problems. In these problems, MRMOGA achieved better results in terms of effectiveness as well as in terms of efficiency. The exploration of the search space by using multiple resolutions not only improves convergence, but also makes it possible to yield solutions that would be difficult to find otherwise. This was observed in the problem OSY, in which some solutions were found only when the decision variable space was divided. The main weakness of our approach is the poor distribution at the "fine-grain level". That is, although its PF_{known} covers all the extent of the true Pareto front, the vectors are not equidistantly spaced. Also, it is interesting to notice that in some problems the convergence (considering the SC metric) of an algorithm reached its maximum with a certain number of processors and then dropped. This means that in those cases it does not make any sense to use more processors because the convergence will not increase. This implies that the algorithm is not able to take advantage of an increasing number of processors or, in other words, the algorithm has poor scalability. This behavior suggests the consideration of both the gains in convergence and the speedup to decide whether or not a pMOEA has good scalability.

Since the spread of non-dominated solutions is the weakest point of our approach, as part of our future work we want to integrate a new mechanism to achieve a better distribution. An option to cope with this problem is the use the ϵ-dominance mechanism [32], which is a relaxed form of Pareto dominance that allows us to control convergence of a MOEA. This technique



could be integrated to the optimizer of each island or applied at the end of the search process when the root processor combines all the non-dominated fronts coming from all processors. Due to the unexpected capacity shown by our MRMOGA for solving constrained problems, it would be interesting to carry out a comparative study consisting of more constrained problems to observe in more detail the performance of the approach on these problems. Our future work also considers applying MRMOGA to real-world problems with very costly objective function evaluations. This will allow us to have a more pragmatic assessment of the usefulness of our MRMOGA.

Acknowledgments

The authors thank the anonymous reviewers for their comments which greatly helped us to improve the contents of this paper.

REFERENCES

1. Enrique Alba Torres. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters, Elsevier*, 82(1):7–13, April 2002.
2. Enrique Alba Torres and Jos Ma. Troya Linero. A survey of parallel distributed genetic algorithms. *Complexity*, 4(4):31–51, 1999.
3. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, New York, USA, 1997.
4. James Edward Baker. Reducing Bias and Inefficiency in the Selection Algorithm. In John J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–22, Hillsdale, New Jersey, July 1987. Lawrence Erlbaum.

5. Jürgen Branke, Thomas Kaußler, and Harmut Schmeck. Guidance in Evolutionary Multi-Objective Optimization. *Advances in Engineering Software*, 32:499–507, 2001.

6. Erick Cantú Paz. *Efficient and Accurate Parallel Genetic Algorithms.* Boston: Kluwer Academic Publishers, 2002.

7. Carlos A. Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, August 1999.

8. Carlos A. Coello Coello and Margarita Reyes Sierra. A Coevolutionary Multi-Objective Evolutionary Algorithm. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 1, pages 482–489, Canberra, Australia, December 2003. IEEE Press.

9. Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems.* New York: Kluwer Academic Publishers, May 2002.

10. Yann Collette and Patrick Siarry. *Multiobjective Optimization. Principles and Case Studies.* Springer, August 2003.

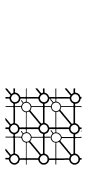
11. Charles Robert Darwin. *On the Origin of Species by Means of Natural Selection Or the Preservation of Favoured Races in the Struggle for Life.* Cambridge University Press, Cambridge, UK, sixth edition, 1964. Originally published in 1859.

12. Francisco de Toro Negro, J. Ortega, E. Ros, S. Mota, B. Paechter, and J. M. Martn. PSFGA: Parallel Processing and Evolutionary Computation for Multiobjective Optimisation. *Parallel Computing, Elsevier*, 30(5–6):721–739, May 2004.

13. Kalyanmoy Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation*, 7(3):205–230, Fall 1999.

14. Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms.* John Wiley & Sons, Chichester, UK, 2001. ISBN 0-471-87339-X.

15. Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.



16. Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

17. Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable Multi-Objective Optimization Test Problems. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 825–830, Piscataway, New Jersey, May 2002. IEEE Service Center.

18. Kalyanmoy Deb, Pawan Zope, and Abhishek Jain. Distributed Computing of Pareto-Optimal Solutions with Evolutionary Algorithms. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 534–549, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

19. David B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence.* The Institute of Electrical and Electronic Engineers, New York, 1995.

20. Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution.* John Wiley, New York, 1966.

21. Carlos M. Fonseca and Peter J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

22. David Edward Goldberg. *Genetic Algorithms in search, optimization, and machine learning.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

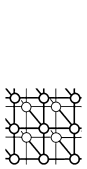
23. Tomoyuki Hiroyasu, Mitsunori Miki, and Shinya Watanabe. The New Model of Parallel Genetic Algorithm in Multi-Objective Optimization Problems–Divided Range Multi-Objective Genetic Algorithm–. In *2000 Congress on Evolutionary Computation*, volume 1, pages 333–340, Piscataway, New Jersey, July 2000. IEEE Service Center.

24. John H. Holland. Concerning efficient adaptive systems. In M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, editors, *Self-Organizing Systems—1962*, pages 215–230. Spartan Books, Washington, D.C., 1962.

25. Jeffrey Horn. Multicriterion Decision Making. In Thomas Bäck, David Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, volume 1, pages F1.9:1 – F1.9:15. IOP Publishing Ltd.

and Oxford University Press, 1997.

26. Alan H. Karp and Horace P. Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, 1990.
27. Hajime Kita, Yasuyuki Yabumoto, Naoki Mori, and Yoshikazu Nishikawa. Multi-Objective Optimization by Means of the Thermodynamical Genetic Algorithm. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature—PPSN IV*, Lecture Notes in Computer Science, pages 504–512, Berlin, Germany, September 1996. Springer-Verlag.
28. Joshua Knowles and David Corne. Properties of an Adaptive Archiving Algorithm for Storing Nondominated Vectors. *IEEE Transactions on Evolutionary Computation*, 7(2):100–116, April 2003.
29. Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.
30. Vipin Kumar and George Karypis Ananth Grama, Anshul Gupta. *Introduction to Parallel Computing: design and analysis of parallel algorithms.* Benjamin Cummings Publishing Company, Redwood City, California, USA, 1994.
31. Frank Kursawe. A Variant of Evolution Strategies for Vector Optimization. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature. 1st Workshop, PPSN I*, volume 496 of *Lecture Notes in Computer Science Vol. 496*, pages 193–197, Berlin, Germany, October 1991. Springer-Verlag.
32. Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining Convergence and Diversity in Evolutionary Multi-objective Optimization. *Evolutionary Computation*, 10(3):263–282, Fall 2002.
33. Shyh-Chang Lin, William F. Punch III, and Erik D. Goodman. Coarse-grain genetic algorithms, categorization and new approaches. In *Sixth IEEE Symposium on Parallel and Distributed Processing*, pages 28–37, Dallas, Texas, USA, October 1994. IEEE Computer Society Press.
34. Melanie Mitchell. *An Introduction to Genetic Algorithms.* The MIT Press, Cambridge, Massachusetts, 1996.
35. Andrzej Osyczka. *Evolutionary Algorithms for Single and Multicriteria Design Optimization.* Physica Verlag, Germany, 2002. ISBN 3-7908-1418-0.
36. Andrzej Osyczka and Sourav Kundu. A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm. *Structural Optimization*, 10:94–99, 1995.



37. Ian C. Parmee and Harish D. Vekeria. Co-operative Evolutionary Strategies for Single Component Design. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 529–536, San Francisco, California, USA, 1997. Morgan Kaufmann Publishers.

38. H. Sawai and S. Adachi. Parallel distributed processing of a parameter-free GA by using hierarchical migration methods. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, volume 1, pages 579–586, San Francisco, California, USA, July 1999. Morgan Kaufmann.

39. J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms.* PhD thesis, Vanderbilt University, 1984.

40. J. David Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100, Hillsdale, New Jersey, 1985. Lawrence Erlbaum.

41. Jason R. Schott. Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1995.

42. Hans-Paul Schwefel. Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. Dipl.-Ing. thesis, 1965. (in German).

43. Ron Shonkwiler. Parallel genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 199–205, Urbana-Champaign, Illinois, USA, June 1993. Morgan Kaufmann.

44. Felix Streichert, Holger Ulmer, and Andreas Zell. Parallelization of Multi-objective Evolutionary Algorithms Using Clustering Algorithms. In Carlos A. Coello Coello and Eckart Zitzler, editors, *Evolutionary Multi-Criterion Optimization. Fourth International Conference, EMO 2005*, pages 92–107, Guanajuato, Mexico, 2005. Springer-Verlag. Lecture Notes in Computer Science. Volume 3410.

45. K.C. Tan, E.F. Khor, and T.H. Lee. *Multiobjective Evolutionary Algorithms and Applications.* Springer-Verlag, London, 2005. ISBN 1-85233-836-9.

46. Marco Tomassini. Parallel and distributed evolutionary algorithms: A review. In K. Miettinen, M. Mäkelä, P. Neittaanmäki, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 113–133. John Wiley and Sons, 1999.

47. David A. Van Veldhuizen, Jesse B. Zydallis, and Gary B. Lamont. Considerations in engineering parallel multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(3):144–173, April 2003.

48. Christian von Lücken. Algoritmos evolutivos para optimización multi-objetivo: un estudio comparativo en un ambiente paralelo asíncrono. Master's thesis, Universidad Nacional de Asunción, San Lorenzo, Paraguay, December 2003. (in Spanish).

49. Shinya Watanabe, Tomoyuki Hiroyasu, and Mitsunori Miki. LCGA: Local Cultivation Genetic Algorithm for Multi-Objective Optimization Problems. In W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, page 702, San Francisco, California, July 2002. Morgan Kaufmann Publishers.

50. Zhong-Yao Zhu and Kwong-Sak Leung. Asynchronous Self-Adjustable Island Genetic Algorithm for Multi-Objective Optimization Problems. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 837–842, Piscataway, New Jersey, May 2002. IEEE Service Center.

51. Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, November 1999.

52. Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Technical Report 70, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zürich, Switzerland, December 1999.

53. Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.