Applications of Parallel Platforms and Models in Evolutionary Multi-Objective Optimization

Antonio López Jaimes and Carlos A. Coello Coello*

CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D.F. 07360, México
email: tonio.jaimes@gmail.com, ccoello@cs.cinvestav.mx

Summary. This chapter presents a review of modern parallel platforms and the way in which they can be exploited to implement parallel multi-objective evolutionary algorithms. Regarding parallel platforms, a special emphasis is given to global metacomputing which is an emerging form of parallel computing with promising applications in evolutionary (both multi- and single- objective) optimization. In addition, we present the well-known models to parallelize evolutionary algorithms (i.e., master-slave, island, diffusion and hybrid models) describing some possible strategies to incorporate these models in the context of multi-objective optimization. Since an important concern in parallel computing is performance assessment, the chapter also presents how to apply parallel performance measures in multi-objective evolutionary algorithms taking into consideration their stochastic nature. Finally, we present a selection of current parallel multi-objective evolutionary algorithms that integrate novel strategies to address multi-objective issues.

1 Introduction

Problems having two or more (normally conflicting) objectives naturally arise in a variety of disciplines. Such problems, which are called "multi-objective" have not one, but a set of solutions, which are collectively known as the "Pareto optimal set". All the solutions contained in the Pareto optimal set are equally good, and represent the best possible trade-offs among all the objectives.

The use of evolutionary algorithms (EAs) for solving multi-objective optimization problems (MOPs) has been quite popular in the last few years. The rising popularity of multi-objective evolutionary algorithms (MOEAs) is mainly due to their flexibility and ease of use with respect to traditional mathematical programming techniques [16, 20, 51, 58, 17]. However, a wide variety of real-world MOPs are highly demanding in terms of CPU time (e.g., in aeronautical

*The second author is also associated to UMI-LAFMIA 3175 CNRS.

engineering [49]). This may limit the applicability of MOEAs, since they normally require a considerably large number of objective function evaluations to achieve reasonably good results. The use of parallelism is an obvious choice to solve these problems in a reasonable amount of time [61]. 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 tend to improve the population's diversity, (4) they reduce the probability of finding suboptimal solutions, (5) and they can cooperate in parallel with other search techniques (including non-evolutionary techniques).

During the last three decades, parallel Evolutionary Algorithms used for global optimization (pEAs) have been widely studied [5, 15, 54, 60]. 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 MOP 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 MOPs 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 [39, 66, 22]) also adds to the computational overhead of a MOEA, which is one more reason to justify their parallelization.

The present chapter provides an overview of the models employed to implement parallel MOEAs and discusses some implementation issues that deserve to be considered in the light of multi-objective optimization. Additionally, we present a review of the parallel architectures currently available to implement pMOEAs. Special emphasis is given to global metacomputing which is a new form of parallel computing that allows us to share computing resources in order to build a large networked metacomputer. The chapter also presents some of the parallel performance metrics that have been adopted to assess pMOEA's efficiency as well as a short discussion regarding the conditions under which one may find superunitary speedups. We also present the concept of fixed-time speedup, which is an alternative measure of speedup that has some advantages over other forms of speedup usually adopted in the current literature.

The remainder structure of the chapter is the following. Some basic concepts related to multi-objective optimization are provided in Section 2. Section 3 describes a taxonomy of parallel architectures, emphasizing MIMD architectures (Multiple Instruction Stream, Multiple Data Stream), since they are the most common parallel platform in current use. In Section 4, we present the most commonly used pMOEA parallel models. In turn, the most popular parallel performance measures in current use are introduced in Section 5. Finally, Section 6 describes a selection of pMOEAs that incorporate novel par-

allel strategies. The reader can find the description of other pMOEAs in the overview presented by Talbi et al. [57].

2 Basic Concepts

We are interested in solving problems of the type:

$$\text{Find } \mathbf{x} \text{ wich optimizes } \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})] \tag{1}$$

subject to:

$$g_i(\mathbf{x}) \leq 0 \quad i = 1, 2, \ldots, n \tag{2}$$

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

where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables, f_i, $i = 1, ..., k$ are the objective functions and g_i, h_j, $i = 1, ..., m$, $j = 1, ..., p$ are the constraint functions of the problem.

In multi-objective optimization problems the aim is to find good compromises (trade-offs). To understand the concept of optimality, we will introduce first a few definitions.

Definition 1. Given two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$, we say that $\mathbf{x} \leq \mathbf{y}$ if $x_i \leq y_i$ for $i = 1, ..., k$, and that $\mathbf{x}$ **dominates** $\mathbf{y}$ (denoted by $\mathbf{x} \prec \mathbf{y}$) if $\mathbf{x} \leq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$.

Definition 2. We say that a vector of decision variables $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n$ is **nondominated** with respect to $\mathcal{X}$, if there does not exist another $\mathbf{x}' \in \mathcal{X}$ such that $\mathbf{f}(\mathbf{x}') \prec \mathbf{f}(\mathbf{x})$.

Definition 3. We say that a vector of decision variables $\mathbf{x}^* \in \mathcal{F} \subset \mathbb{R}^n$ ($\mathcal{F}$ is the feasible region) is **Pareto optimal** if it is nondominated with respect to $\mathcal{F}$.

This is a notion of optimality that was originally proposed by Francis Ysidro Edgeworth [26] and later generalized by Vilfredo Pareto [52]. Although some authors call this notion *Edgeworth-Pareto optimality* (see for example [55]), the term *Pareto optimality* is the most common and widespread, and is, therefore, the one used in this chapter.

Definition 4. The **Pareto Optimal Set** $\mathcal{P}^*$ is defined by:

$$\mathcal{P}^* = \{\mathbf{x} \in \mathcal{F} | \mathbf{x} \text{ is Pareto optimal}\}$$

Definition 5. The **Pareto Front** $\mathcal{PF}^*$ is defined by:

$$\mathcal{PF}^* = \{\mathbf{f}(\mathbf{x}) \in \mathbb{R}^k | \mathbf{x} \in \mathcal{P}^*\}$$

We thus wish to determine the Pareto optimal set from the set $\mathcal{F}$ of all the decision variable vectors that satisfy (2) and (3).

3 Parallel Architectures

Many schemes to classify parallel computers [28, 10, 59, 37] have been proposed so far. However, none of them has become standard in the specialized literature. The difference among these schemes lies on the characteristics of the parallel system that are taken into account, namely: the organization of the address space, the interconnection network, or the processors' granularity. For our purposes we will use the well-known Flynn's taxonomy. We will then complement the MIMD classification following a similar approach as in Johnson [37] or Bell [10].

SISD (Single Instruction Stream, Single Data Stream). This class of computers represents the conventional single-processor von Neumann computers, where only one instruction is executed at the same time over a unique data element.

SIMD (Single Instruction Stream, Multiple Data Stream). This architecture consists of a central instruction unit that broadcasts a single instruction to a group of slave processors which apply the instructions in a synchronized fashion on different pieces of data stored in its own memory. Pipeline vector processors and processor array computers are considered as specialized SIMD architectures.

MISD (Multiple Instruction Stream, Single Data Stream). A computer of this class is able to apply different instructions to the same stream of data at a time. Although some authors consider that this architecture is mainly theoretical because there is no practical computer that fits this model [31, 25, 59], other authors have considered high-level pipeline computer instances of this model. Likewise, we can include fault-tolerant computers in the MISD class since in these systems several processors are applied upon the same data using different programs in order to mask a possible fault in one of the programs.

MIMD (Multiple Instruction Stream, Multiple Data Stream). In a MIMD architecture, a group of processors independently execute different instruction streams over different data sets. This architecture is intended to support parallel applications that require processors to operate autonomously during most of the time.

3.1 Taxonomy of MIMD computers

Although SIMD computers were very popular in the past (e.g., Connection Machines [34], MasPar computers [12], and Cray computers [8], which were

all developed during the 1980s and early 1990s), nowadays manufacturers have moved toward MIMD architectures (clusters and constellations) (see e.g., [11, 47]), and SIMD computers are only used in specialized application domains such as image processing. To illustrate this it suffices to consider that in 1993, vector computers comprised about 74% of the 500 top supercomputers[2], while MIMD computers comprised 26%. By 2008, the situation had drastically changed inasmuch as vector computers comprise only 0.4% and MIMD computers comprise the remaining 99.6%. In the light of this trend in parallel high-performance architectures, MIMD systems are further discussed in the following paragraphs.

According to the organization of the address space and the connection method between memory and processors, MIMD computers are classified into two categories: shared memory MIMD systems, commonly known as *multiprocessors*, and distributed memory systems, usually referred to as *multicomputers* (see Figure 3).

Multiprocessor MIMD Systems

In this class of systems all the processors read and write to a common physical address space through an interconnection network (see Figure 1). Processors communicate each other by writing information on the global memory so that the other processors can read it. A drawback of this communication scheme is that data integrity is endangered since multiple processors can concurrently request a write operation on the same memory location. In order to avoid write conflicts the programmer has to use classic synchronization methods such as semaphores, locks and barriers. In order to improve efficiency, each processor is equipped with a cache memory to speedup access to frequently used data. However, the incorporation of cache memories motivates the need of protocols to ensure the coherence between global and cache memory. The most common form of multiprocessors are known as *Symmetric Multiprocessors* (SMP), which are computers constituted by multiple processors where the memory access time to any region of the shared memory is approximately the same for each processor. On the contrary, in a *Non-Uniform Memory Access* (NUMA) system, the shared memory is partitioned in such a way that each partition is associated to a different group of processors. As a result, the access time depends on which memory location is accessed.

Multicomputer MIMD Systems

In these systems, each processor has its own local memory. In this case, an interconnection network is used to allow communication among processors

[2]The Top500 project maintains a worldwide classification and statistics of the most powerful computers in accordance to the LINPACK benchmark (www.top500.org).

by means of message passing. In this category, we can find one of the most common parallel systems in current use, the *cluster of computers*. A cluster is a system comprised of independent computers (nodes) connected by a low-latency network. The nodes in a cluster are capable of independent operation and communicate with one another via message passing. Clusters comprise two classes: *clusters of workstations* (COWs)[3] and *constellation systems*. Both classes of clusters use both commercial off-the-shelf networks and computing nodes. Nevertheless, according to Dongarra et al. [24], the difference between these systems relies on the number of nodes connected by the network and the number of processors on each node. In a COW system, there are more nodes than processors in any of its nodes, while a constellation system has more processors than nodes in the clusters. Usually, each node in a constellation is configured as a SMP with hundreds of processors. Nowadays, a cluster with a low-latency proprietary network and more than 1000 processors is considered a *massively parallel processing system* (MPP)[4].

Global metacomputing

This is an emerging form of MIMD parallel systems that has attracted a great interest in the last decade. The basic idea of *global metacomputing* is employing computers geographically distributed around the world as if they were one large parallel machine or metacomputer. Although this concept dates back to the mid 1960's [63, 42] the limitations both in storage capacity and network performance in those days made the idea impractical.

The most ambitious form of metacomputing is named *grid computing*. Grid computing is a group of technologies and infrastructure that coordinate large-scale resource sharing among individuals or organizations around the world [29, 30]. Although, currently, the most common shared resources are computer power (e.g., clusters, constellations) and data (e.g., software, databases), grid computing is not limited to computational resources. Grid infrastructure also enables organizations to share: (1) scientific instruments such as telescopes, particle accelerators and electron microscopes, (2) analysis procedures and computational results, and (3) human expertise in the form of collaborative work.

Volunteer computing is another form of metacomputing mainly focused on facilitating the sharing of computer power and data storage among individuals around the world. The idea behind volunteer computing is to provide an infrastructure that enables individuals to share the idle processing power of their computers to build a large parallel machine which is used to solve

[3] Also known as Beowulf clusters or network of workstations (NOWs).

[4] MPP is a loosely-defined term that has been used to qualify different parallel architectures through the years. For instance, the first MPP system was a SIMD computer. However the current usage is mainly intended for distributed-memory systems.

expensive computational problems [53]. Among the most successful volunteer computing projects, we can find the distributed.net project [1] that solved the RSA RC-56 decryption challenge, and the SETI@home [2] project that analyzes radio telescope data looking for signs of extraterrestrial intelligence. As of 2008, SETI@home accounted with near 890 000 registered volunteers supplying an average of 460Tflops[5]. In order to put this figure in its proper context, it is worth considering that the fastest supercomputer currently available operates at 1026 Tflops the second fastest operates at 478 Tflops, which is comparable with SETI@home's throughput. Although these projects are categorized as "true volunteer" computing since users are unpaid and unrelated to the project administrators, there are other variations of this scheme, including private volunteer computing. That is, organizations such as companies, universities or laboratories which turn their existing networks into virtual supercomputers that they can use for their research. For a complete taxonomy of volunteer computing systems interested readers are referred to [53]. Problems that can be modelled using the master-slave paradigm are ideal to be solved with a volunteer computing system.

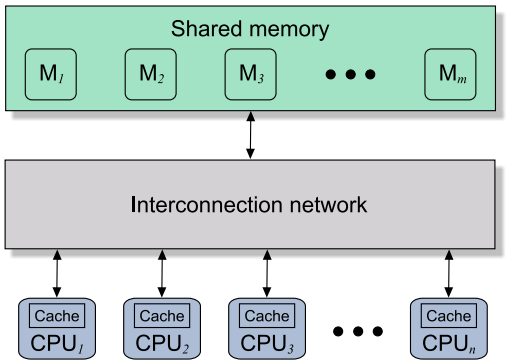


Fig. 1. Multiprocessor MIMD system (shared memory).

4 Parallelization Models of MOEAs

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, the diffusion model and the hybrid model. Nevertheless, currently there is no standard way to extend these models

[5]This information was taken from the web page of BOINC (http://boincstats.com), the middleware used by SETI@home and other volunteer computing projects.

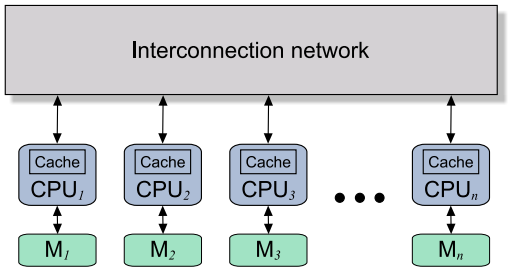


Fig. 2. Multicomputer MIMD system (distributed memory).

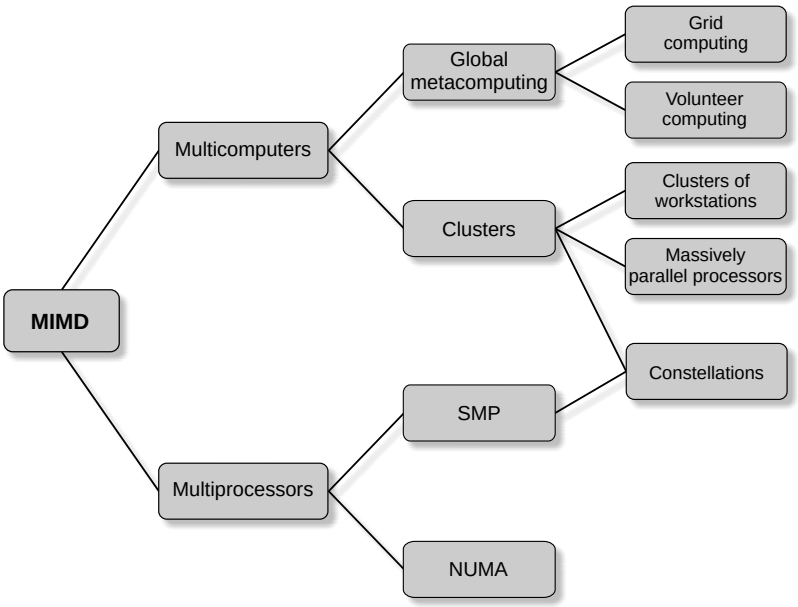


Fig. 3. MIMD taxonomy.

to the multi-objective field. Van Veldhuizen, Zydallis and Lamont [61] provide a detailed discussion of the generic versions of these models when applied to MOEAs. Next, we will briefly discuss each of them.

4.1 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 processors. As soon as the slaves complete the evaluations they return the objective function values to the master and remain idle until

the next generation.[6] 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 4. A master-slave pMOEA explores the search space in the same way as a serial MOEA does. Therefore, it finds the same solutions found by its serial counterpart. However, the execution time is reduced (ideally p times with p processors). The master-slave model is perfect to be implemented in a volunteer computing system. However, as we indicate in the following lines, the evaluation time of the objective functions should be greater compared to the communication time.

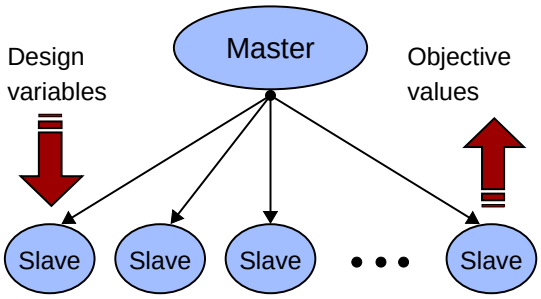


Fig. 4. Master-slave model.

In multi-objective optimization, there are three schemes to distribute the objective functions evaluations among a number of slave processors:

1. Evenly distribute the population over the slaves in such a way that each slave evaluates all the objective functions for its share of the population. Assuming that the master processor evaluates a portion of the individuals, the total computation time for each generation using P processors is given by
$$T_P = (P-1)t_{cp} + t_{ca} + \frac{nt_F}{P}, \tag{4}$$
where t_{cp} is the time required to send one individual, t_{ca} is the time required to send back the objective values to the master, t_F is the time required to evaluate one individual, and n is the size of the population. If $t_{cp} = t_{ca}$ then we obtain the computation time presented by Cantú-Paz [15]:
$$T_P = Pt_{cp} + \frac{nt_F}{P}, \tag{5}$$
From Equation (5) we can derive some interesting results. First, the optimal number of processors, P^*, that minimizes T_P is given by

[6] A scheme where the master also evaluates some individuals is possible, though.

$$P^* = \sqrt{n\gamma}, \tag{6}$$

where $\gamma = \frac{t_F}{t_{cp}}$. Using the parallel execution time of Equation (5) and the execution time of a sequential MOEA, $T_S = nt_F$, we can ensure that a master-slave MOEA is faster than the sequential counterpart using the following condition:

$$\gamma > \frac{P^2}{n(P-1)}. \tag{7}$$

Limited by Amdahl's law [7], the speedup of a master-slave MOEA has a maximum value. According to Cantú-Paz [15] the speedup curve is limited by

$$S_P^* = \frac{1}{2}P^*. \tag{8}$$

2. Each objective function is assigned to a different even partition of slaves (i.e., partition $\mathcal{P}_1$ evaluates f_1, $\mathcal{P}_2$ evaluates f_2 and so forth). Each partition of slaves then evenly distributes the population over their processors. Here, we are assuming that the size of each partition, $\mathcal{P}_i$, of slaves is $\frac{P}{k}$, where k is the number of objectives. If t_c is the time required to broadcast the entire population and $t_{f_{max}} = \max_{1 \leq i \leq k}\{t_{f_i}\}$ is the execution time of the most expensive objective function in the set of objective functions, the execution time for each generation using this distribution scheme is given by

$$T_P = t_c + (P-1)t_{ca} + \frac{knt_{f_{max}}}{P}. \tag{9}$$

3. Each objective function is decomposed into smaller algebraic terms and then these terms are distributed to different groups of slaves. This way, each group of slaves evaluates a small part of a complex objective function. Let t_{com} be the time required to combine the decomposed objective values and $t_{f_{max}} = \max_{1 \leq i \leq k,\ 1 \leq j \leq p_i}\{t_{f_{i,j}}\}$ the execution time of the most expensive partition of the objective function f_i (p_i is the number of partitions of function f_i). Then, the execution time of one generation of this distribution scheme is

$$T_P = t_{com} + t_c + (P-1)t_{ca} + \frac{knt_{f_{max}}}{P}. \tag{10}$$

4.2 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 (see Figure 5). 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 5) 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. As a consequence, this model is appropriate for shared-memory MIMD computers such as SMPs. However, custom hardware implementations on SIMD computers are also possible [27].

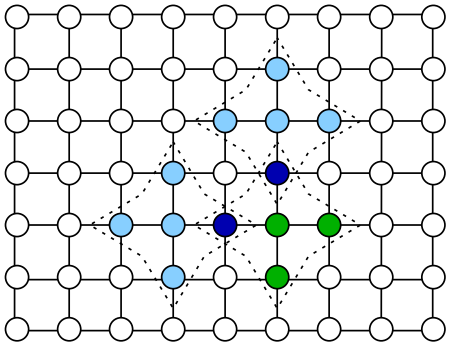


Fig. 5. Diffusion model.

4.3 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 MOEA is executed for a number of generations called an *epoch*. At the end of each epoch, individuals migrate between neighboring islands. The neighbors are defined 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 Figure 6, 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 or for grid computing systems.

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

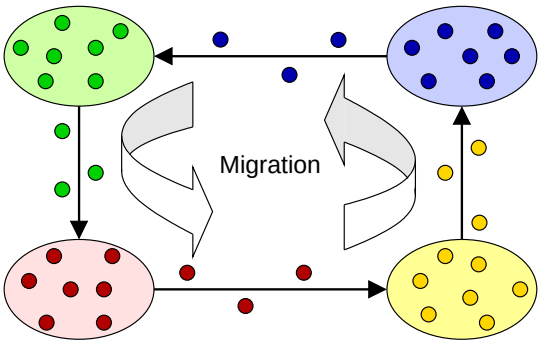


Fig. 6. Master-slave model.

individuals to migrate, and the decision regarding the individuals which will migrate and those which will be replaced by the immigrants. Tables 1 and 2 present a number of possible migration and replacement schemes proposed by Coello, Lamont and Van Veldhuizen [16].

Scheme	Description
Non-uniform schemes	
Elitist (random)	Migrate a random sample of individuals from the current nondominated front.
Elitist (niching)	Migrate individuals evenly distributed from the current nondominated front.
Elitist (front)	Migrate the entire nondominated front.
Uniform schemes	
Random	Migrate n individuals selected at random.
Elitist (random)	Migrate n individuals selected at random from the current nondominated front plus some individuals randomly selected from the ranked Pareto fronts if necessary.
Elitist (niching)	Migrate n individuals evenly distributed from the current nondominated front plus some individuals evenly distributed from the remainder ranked Pareto fronts if necessary.

Table 1. Migration schemes in the island model.

The model allows each island to have their own parameter setting. Depending on the homogeneity of the islands, we can recognize four variants of the island model:

Scheme	Description
Random	Randomly replace n individuals.
None	No replacement, thereby the population increases its size.
Elitist (random)	Maintain the current nondominated front and randomly replace any dominated individual.
Elitist (ranking)	Rank the population into nondominated fronts and replace individuals from the worst ranked fronts with the immigrants.
Elitist (100% ranking)	Combine immigrants with the current population, rank the combined population and discard individuals from the worst ranked fronts.

Table 2. Replacement schemes for the island model.

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, uses its own evolutionary operators and solution encoding technique. Even more, each island can be constituted by a different MOEA.
3. **Island pMOEA with different objective subsets**. Each island is responsible for optimizing a different partition of the entire objective subset.
4. **Island pMOEA with different regions in the search space**. Each island may be explicitly instructed to explore a particular region of decision variable or objective function space to optimize computational resources.

There are a number of approaches [41, 62] that have successfully used the heterogeneous scheme adopting different MOEAs in each island. Whereas in León, Miranda and Segura [41] only MOEAs such as NSGA-II and SPEA2 are employed, it is possible to combine different metaheuristics as in the approach proposed by Vrugt and Robinson [62]. In this approach the authors combine four different metaheuristics, namely: a multiobjective evolutionary algorithm, a particle swarm optimization algorithm, an adaptive Metropolis search technique and a differential evolution algorithm. The idea behind a heterogeneous approach is to build a robust algorithm in which some metaheuristics can compensate the weaknesses of other metaheuristics in a particular problem. The success of a heterogeneous approach greatly depends on the strategy adopted to evaluate the performance of each metaheuristic in order to favor the best metaheuristics in future generations.

In the third variant, the straightforward strategy is to assign a different objective function to a set of islands executing, thereby, a single objective evolutionary optimization algorithm [50]. On the other hand, if the optimization problem has a large number of objectives, then it is possible to partition the objective set into groups with more than one objective. In this situation,

we have to carefully decide how to group objective functions. As recent studies [14, 45] have pointed out, the conflict among objectives determines the importance of each objective in the optimization problem. In addition, there are different degrees of conflict among the objectives and it is possible that one objective that is not in conflict with a certain objective might be in great conflict with another. Therefore, we can cluster objectives according to the degree of conflict among them. That is, the objective set should be partitioned in such a way that the conflict among objectives inside each cluster is maximum and the conflict among objectives in different clusters is minimum.

In order to avoid that two or more demes exploit the same region of the search space it is convenient to instruct each deme to solve non-overlapping regions of decision variable or objective function space. In a general MOP it is very hard to devise a priori a distribution such that: (1) covers the entire search space, (2) assigns regions of equal size, and (3) aggregates a minimum complexity to constraint demes to their assigned region.

In Coello, Lamont and Van Veldhuizen [16], three strategies to distribute the search space among the demes are identified:

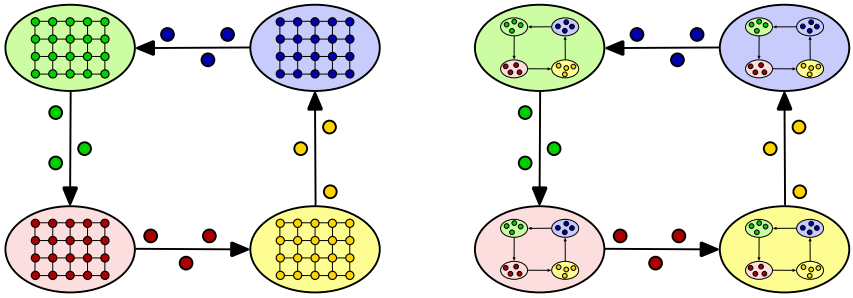
1. Constrain each deme to a particular region and force it to generate individuals until a suitable number of them is produced within its assigned region. The drawback of this strategy is that it introduces an overhead because of the extra number of objective function evaluations performed.
2. Constrain each deme to a particular region by migrating individuals to the deme covering such region. This strategy introduces a communication overhead, though.
3. Constrain each deme to a particular region of the Pareto front by introducing a bias in the search in such a way that each deme concentrates on a specific region of the Pareto front. The drawback of this approach is that the shape of the true Pareto front needs to be known a priori. Section 6 presents two pMOEAs proposed by Deb, Zope and Jain [23], and by Streichert, Ulmer and Zell [56] that follow this approach.

4.4 Hybrid Model

Another option to parallelize a MOEA is combining a coarse-grained parallel scheme at a high level (e.g., island model) with a fine-grained scheme at a low level (e.g., diffusion model). Cantú Paz discussed three types of hybrid schemes that use an island model at the high level:

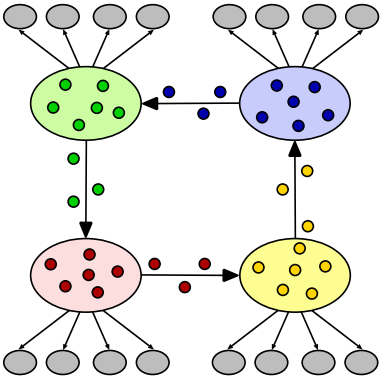
1. Each island implements a diffusion pMOEA (see Figure 7(a)),
2. Each island implements a master-slave pMOEA (see Figure 7(c)) and
3. Each island implements an island pMOEA (see Figure 7(b)).

The first hybrid is ideal for a constellation of computers where each SMP node can be used to implement the diffusion pMOEA and the entire cluster of SMPs can be configured as the island model.



(a) Island/diffusion

(b) Island/island



(c) Island/master-slave

Fig. 7. Hybrid models.

5 Performance Assessment of Parallel MOEAs

Parallel performance indicators provide useful information that can be used to identify the bottlenecks of the pMOEA or to predict its performance on a given parallel system. For instance, researchers can use these indicators to improve a parallel MOEA and a practitioner can decide if extending a given parallel system is cost-effective according to an analysis of the performance results.

5.1 Speedup

The most well-known measure of performance of a parallel system is the *speedup*, which is defined as the ratio of the total execution time on an uniprocessor to the total execution time in the parallel system. In other words, the speedup of a parallel algorithm executed in p processors is given by

$$S_p = \frac{T_1}{T_p},$$

where T_1 is the wall-clock time of the sequential algorithm and T_p is the wall-clock time of the parallel algorithm. Usually, T_1 should correspond to the optimal sequential algorithm (this speedup is denoted as *strong speedup* by Alba [4]). Nevertheless, as Helmbold and McDowell [33] point out, this requirement is impractical since even the best known sequential algorithm may be improved in the future and it may exhibit a different performance depending on the data of interest.

Alba [4] suggests different approaches to compute the speedup in evolutionary algorithms. First of all, given the stochastic nature of evolutionary algorithms, the execution times of the algorithms should involve the computation of average times over several runs. In fact, some authors [18] suggest to measure average execution times even for deterministic algorithms, since there are some random events in a parallel system such as the execution of system daemons or the reordenation of memory accesses, which may produce time variations from one execution to another. Additionally, it is recommended to compute the variance of the running times since a high variance may be a symptom of high contention for locks, for instance. Alba defines a type of speedup as the most appropriate for evaluating parallel evolutionary algorithms, namely *speedup with solution stop*. In this type of speedup, instead of using the number of evaluations as the stopping criterion, the "quality of the solution" is employed. In single-objective optimization, it is straightforward to determine if two solutions have the same quality, i.e., if they have the same fitness. In contrast, in multi-objective optimization, there is no consensus regarding how to determine, in general, the quality of the Pareto front generated by a MOEA. A practical approach is to use a unary performance indicator that assesses both the distribution and the convergence of the approximation of the Pareto front produced. Some indicators that might be adopted to determine the quality of the solution are: hypervolume [65], inverted generational distance [16] or the G-metric [44]. There are two variants to compute the *speedup with solution stop*: (1) compare the pMOEA against a canonical sequential MOEA with a global population, and (2) compare the parallel p-processor MOEA on a single processor and the same algorithm on p processors.

In addition to these types of speedup, we can use a similar approach to that adopted by the so-called fixed-time model [32], where a predefined a time limit is used and the "work" increases as processors are added. In our case, the work is defined as the approximation to the true Pareto front measured with a suitable quality indicator. Thus, the *fixed-time speedup* is defined by

$$Sx_p = \frac{I(A_1)}{I(A_p)},$$

where I is a unary quality indicator, A_p is the approximation set obtained by the pMOEA and A_1 is the approximation set obtained by the sequential MOEA using some fixed-time window in both algorithms. As in the previous

types of speedup, it is advised to use the p-processor MOEA on a single processor as the sequential algorithm. This type of speedup has some advantages over the *speedup with solution stop*. First, it is easier to incorporate the fixed-time stopping criterion to an existing code. Second, the same output datasets obtained in several runs can be used to compute the fixed-time speedup with different quality indicators. In contrast with the *speedup with solution stop*, we have to execute the algorithms again if we change the quality indicator used as our stopping criterion.

Superunitary speedup

The condition when the speedup with p processors is greater than p is referred to as *superunitary speedup* (also known as superlinear speedup). Although in the past many works on parallel evolutionary algorithms have reported superunitary speedup [43, 9, 6], in some cases this phenomenon is due to a comparison against an inefficient or ineffective sequential algorithm. Nonetheless, many authors have identified a range of conditions that give rise to superunitary speedup [32, 33, 3, 4] even if we compare against the same code on a single processor. Some of the sources of superunitary speedup are the following:

- *Increase of high-speed memories.* When the number of processors is increased, the number of high-speed memories (e.g., cache, CPU registers) is also increased. While a large number of individuals might not fit into small but high-speed memories, if the population is distributed in many processors, it is possible that the resulting subpopulations can perfectly fit in high-speed local memories instead of the low-speed RAM.
- *Reduced overhead.* The reason is that the time consumed on some operating system's kernel calls on an n processor machine is only $1/n$ of the time required on a single processor.
- *Shift algorithm's profile.* When a fixed-time model is used, superunitary speedup may result if a parallel algorithm spends more time in faster routines.

Inasmuch as these are mainly hardware sources, it is important to present details of the parallel architecture used, specially if superunitary speedup is reported.

In the specialized literature, we can find other conditions under which superunitary speedup may appear. For instance, time gains obtained from traversing the search space in parallel or improvements due to the use of smaller abstract data structures (e.g., a list for the primary population or the external archive). However, these conditions may produce superunitary speedup only when the parallel MOEA is compared to a sequential MOEA with a global population or when a subefficient sequential algorithm is employed.

5.2 Other Parallel Performance Measures

- **Efficiency** (E). This metric [40] 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:

$$E_p = \frac{S_p}{p}, \tag{11}$$

  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 communication 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 [38] to estimate the serial fraction[7] for measuring the performance of a parallel algorithm on a fixed-size problem. Mathematically, it is defined as:

$$F_p = \frac{1/S_p - 1/p}{1 - 1/p}. \tag{12}$$

  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 it 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 [38] consider that the speedup tends to be superlinear.

6 Selection of Parallel MOEAs

Although several researchers have reported the use of parallel MOEAs [48, 16], 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. [35]. 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 (see Figure 8). 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

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

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 by de Toro Negro et al. [19].

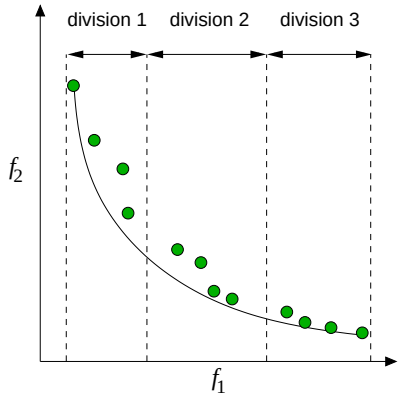
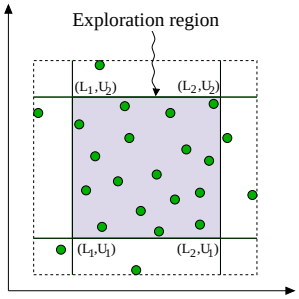


Fig. 8. Division of the population in DRMOGA.

Zhu & Leung proposed the Asynchronous Self-Adjustable Island Genetic Algorithm (aSAIGA) [64]. In aSAIGA, rather than migrating a set of individuals, the islands exchange information related to their current explored region. The "exploration region" (ER) is the hypercube containing most of the individuals of the archive maintained by the sequential MOEA (see Figure 9). 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.



The exploration region is defined by the hypercube $[L_1, U_1] \times \ldots \times [L_N, U_N]$. Let $\{f_i^1, f_i^2, \ldots, f_i^{n'}\}$ be the permutation with the ascending sorted values of the i-th ($i = 1, \ldots, N$) objective, where n' is the size of the archive and N the number of objectives. Then: $L_i = f_i^{\lceil \frac{1}{4} n \rceil}$, $U_i = f_i^{\lceil \frac{3}{4} n \rceil}$ for $1 \leq i \leq N$.

Fig. 9. Exploration region used in ASAIGA.

Another island pMOEA was introduced by Deb et al. [23]. 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" (see Figure 10) based on a concept defined by Branke, Kaußler and Schmeck [13]. 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.

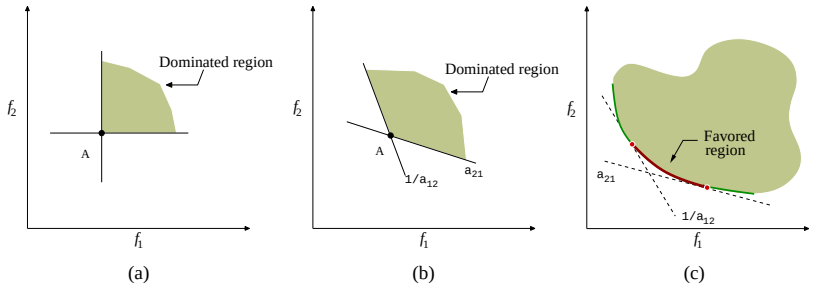


Fig. 10. Dominated region with the usual Pareto domination (a); dominated region with the concept of guided domination (b); "nondominated" region of the Pareto front (c).

Streichert et al. [56] 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 by Deb et al. [21]. 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.

López Jaimes and Coello Coello [36] proposed an approach called Multiple Resolution Multi-Objective Genetic Algorithm (MRMOGA), which consists of a pMOEA based on the island paradigm, with heterogeneous nodes. The main idea of this approach is to encode the solutions using a different resolution in each island. Then, variable decision space is divided into hierarchical levels with well-defined overlaps. Evidently, migration is only allowed in one direction (from low resolution to high resolution islands). MRMOGA uses an external population, and the migration strategy considers such population as well (see

Figure 11). The approach also uses a strategy to detect nominal convergence of the islands in order to increase their initial resolution. The rationale behind this approach is that the true Pareto front can be reached faster using this change of resolution in the islands, because the search space of the low resolution islands is proportionally smaller and, therefore, convergence is faster. The results indicated that MRMOGA outperforms a parallel version of NSGA-II, with a more significant difference as the number of processor increases.

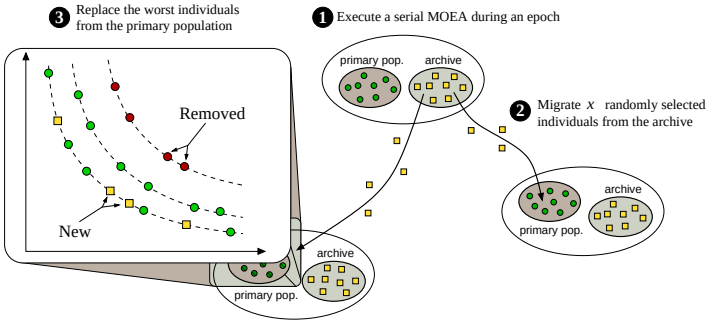


Fig. 11. Schematic view of MRMOGA.

7 Summary and Final Remarks

This chapter has presented an overview of parallel multi-objective evolutionary algorithms (pMOEAs). Firstly, we described the current parallel architectures available. Secondly, we reviewed the most common models to implement pMOEAs. Then, we presented some performance indicators that have been used to measure the efficiency of pMOEAs. Finally, a selection of some pMOEAs that incorporate innovative strategies was presented.

Global metacomputing (see Section 3) is one of the most promising current parallel architectures which is underused in evolutionary optimization. Grid computing is starting to be explored in multi-objective evolutionary optimization [46] and, to the best of our knowledge, no pMOEA has yet been implemented using volunteer computing in spite of the fact that both technologies represent an inexpensive alternative for supercomputing using existing local networks. Consequently, we expect that in the near future more researchers and practitioners will exploit these new forms of parallel computing.

In this chapter, we outlined some general strategies to distribute the objective set in the master-slave model, and some migration/replacement strategies. However, there is still room for new parallelization strategies.

Additionally, given the assessment methodology observed in the current literature, it is clear that there is no standard way to compare and easily pon-

der the performances reported of two different pMOEAs. In sequential MOEAs, we compare new algorithms against well-established MOEAs such as NSGA-II or SPEA2. Thus, we can appraise easily the value of new MOEAs reported in different works. In contrast, for pMOEAs there is no standard reference pMOEA to beat. Thereby, it is required a different methodology to assess performance and report the results produced by pMOEAs.

Acknowledgements

The first author acknowledges support from CONACyT to pursue graduate studies in computer science at CINVESTAV-IPN.

References

1. distributed.net project home page. url: http://www.distributed.net.
2. SETI@home project home page. url: http://setiathome.berkeley.edu.
3. Selim G. Akl and Lorrie Fava Lindon. Paradigms admitting superunitary behaviour in parallel computation. In *CONPAR 94 - VAPP VI: Proceedings of the Third Joint International Conference on Vector and Parallel Processing*, pages 301–312, London, UK, 1994. Springer-Verlag.
4. Enrique Alba Torres. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters, Elsevier*, 82(1):7–13, April 2002.
5. Enrique Alba Torres and José Ma. Troya Linero. A survey of parallel distributed genetic algorithms. *Complexity*, 4(4):31–51, 1999.
6. Enrique Alba Torres and José Ma. Troya Linero. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17(4):451–465, Enero 2001.
7. Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, New York, NY, USA, 1967. ACM.
8. Melvin C. August, Gerald M. Brost, Christopher C. Hsiung, and Alan J. Schiffleger. Cray X-MP: The birth of a supercomputer. *Computer*, 22(1):45–52, 1989.
9. Theodore C. Belding. The distributed genetic algorithm revisited. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 114–121, San Francisco, CA, 1995. Morgan Kaufmann.
10. Gordon Bell. Ultracomputers: A teraflop before its time. *Communications of the ACM*, 35(8):27–47, Agosto 1992.
11. Gordon Bell. Bell's law for the birth and death of computer classes. *Communications of the ACM*, 51(1):86–94, 2008.
12. Tom Blank. The MasPar MP-1 architecture. In *Compcon Spring'90. Intellectual Leverage. Digest of Papers. Thirty-Fifth IEEE Computer Society International Conference.*, pages 20–24, 1990.
13. Jürgen Branke, Thomas Kaußler, and Harmut Schmeck. Guidance in Evolutionary Multi-Objective Optimization. *Advances in Engineering Software*, 32:499–507, 2001.

14. Dimo Brockhoff and Eckart Zitzler. Are All Objectives Necessary? On Dimensionality Reduction in Evolutionary Multiobjective Optimization. In *Parallel Problem Solving from Nature IX*, pages 533–542. Springer-Verlag, 2006.
15. Erick Cantú Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Boston: Kluwer Academic Publishers, 2002.
16. Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, New York, second edition, September 2007. ISBN 978-0-387-33254-3.
17. Yann Collette and Patrick Siarry. *Multiobjective Optimization. Principles and Case Studies*. Springer, August 2003.
18. Lawrence A. Crowl. How to measure, present, and compare parallel performance. *IEEE Parallel Distrib. Technol.*, 2(1):9–25, 1994.
19. 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.
20. Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001. ISBN 0-471-87339-X.
21. 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.
22. 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.
23. 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.
24. Jack Dongarra, Thomas Sterling, Horst Simon, and Erich Strohmaier. High-performance computing: Clusters, constellations, MPPs, and future directions. *Computing in Science and Engineering*, 7(2):51–59, 2005.
25. Ralph Duncan. A survey of parallel computer architectures. *Computer*, 23(2):5–16, 1990.
26. F. Y. Edgeworth. *Mathematical Physics*. P. Keagan, London, England, 1881.
27. Sven E. Eklund. A massively parallel architecture for distributed genetic algorithms. *Parallel Computing*, 30(5-6):647–676, 2004.
28. Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, Septiembre 1972.
29. Ian Foster and Carl Kesselman, editors. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
30. Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

31. Wolfgang K. Giloi. Towards a taxonomy of computer architecture based on the machine data type view. *SIGARCH Comput. Archit. News*, 11(3):6–15, 1983.
32. John L. Gustafson. Fixed time, tiered memory, and superlinear speedup. In *In Proceedings of the Fifth Distributed Memory Computing Conference (DMCC5*, 1990.
33. David P. Helmbold and Charles E. McDowell. Modeling speedup (n) greater than n. *IEEE Trans. Parallel Distrib. Syst.*, 1(2):250–256, 1990.
34. W. Daniel Hillis. *The Connection Machine*. MIT Press, Cambridge, MA, USA, 1989.
35. 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.
36. Antonio López Jaimes and Carlos A. Coello Coello. Mrmoga: a new parallel multi-objective evolutionary algorithm based on the use of multiple resolutions: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(4):397–441, 2007.
37. Eric E. Johnson. Completing an MIMD multiprocessor taxonomy. *SIGARCH Computure Architecture News*, 16(3):44–47, 1988.
38. Alan H. Karp and Horace P. Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, 1990.
39. 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.
40. 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.
41. Coromoto León, Gara Miranda, and Carlos Segura. Parallel hyperheuristic: a self-adaptive island-based model for multi-objective optimization. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 757–758, New York, NY, USA, 2008. ACM.
42. J. C. R. Licklider and Robert W. Taylor. The computer as a communication device. *Science and Technology*, 76:21–31, 1968.
43. 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.
44. Giovanni Lizárraga Lizárraga, Arturo Hernández Aguirre, and Salvador Botello Rionda. G-metric: an m-ary quality indicator for the evaluation of nondominated sets. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 665–672, New York, NY, USA, 2008. ACM.
45. Antonio López Jaimes, Carlos A. Coello Coello, and Debrup Chakraborty. Objective Reduction Using a Feature Selection Technique. In *2008 Genetic and Evolutionary Computation Conference (GECCO'2008)*, pages 674–680, Atlanta, USA, July 2008. ACM Press. ISBN 978-1-60558-131-6.
46. Francisco Luna, Antonio J. Nebro, and Enrique Alba. Observations in using grid-enabled technologies for solving multi-objective optimization problems. *Parallel Comput.*, 32(5):377–393, 2006.

47. Hans Werner Meuer. The TOP500 project: Looking back over 15 years of supercomputing experience. *Informatik-Spektrum*, 31(3):203–222, June 2008.
48. Antonio J. Nebro, Francisco Luna, El-Ghazali Talbi, and Enrique Alba. Parallel Multiobjective Optimization. In Enrique Alba, editor, *Parallel Metaheuristics*, pages 371–394. Wiley-Interscience, New Jersey, USA, 2005. ISBN 13-978-0-471-67806-9.
49. Shigeru Obayashi and Daisuke Sasaki. Multiobjective Aerodynamic Design and Visualization of Supersonic Wings by Using Adaptive Range Multiobjective Genetic Algorithms. In Carlos A. Coello Coello and Gary B. Lamont, editors, *Applications of Multi-Objective Evolutionary Algorithms*, pages 295–315. World Scientific, Singapore, 2004.
50. Tamaki Okuda, Tomoyuki Hiroyasu, Mitsunori Miki, and Shinya Watanabe. DCMOGA: Distributed cooperation model of multi-objective genetic algorithm. In *Proceedings of the PPSN/SAB Workshop on Multiobjective Problem Solving from Nature II (MPSN-II*, 2002.
51. Andrzej Osyczka. *Evolutionary Algorithms for Single and Multicriteria Design Optimization*. Physica Verlag, Germany, 2002. ISBN 3-7908-1418-0.
52. Vilfredo Pareto. *Cours D'Economie Politique*, volume I and II. F. Rouge, Lausanne, 1896.
53. Luis F. G. Sarmenta. *Volunteer Computing*. PhD thesis, Massachusetts Institute of Technology, March 2001.
54. 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.
55. W. Stadler. Fundamentals of multicriteria optimization. In W. Stadler, editor, *Multicriteria Optimization in Engineering and the Sciences*, pages 1–25. Plenum Press, New York, NY, 1988.
56. 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.
57. El-Ghazali Talbi, Sanaz Mostaghim, Tatsuya Okabe, Hisao Ishibuchi, Günter Rudolph, and Carlos A. Coello Coello. Parallel Approaches for Multi-objective Optimization. In Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowinski, editors, *Multiobjective Optimization. Interactive and Evolutionary Approaches*, pages 349–372. Springer. Lecture Notes in Computer Science Vol. 5252, Berlin, Germany, 2008.
58. K.C. Tan, E.F. Khor, and T.H. Lee. *Multiobjective Evolutionary Algorithms and Applications*. Springer-Verlag, London, 2005. ISBN 1-85233-836-9.
59. Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, Upper Saddle River, New Jersey, EE. UU., Enero 2002.
60. 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.

61. 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.
62. Jasper A. Vrugt and Bruce A. Robinson. Improved evolutionary optimization from genetically adaptive multimethod search. *Proceedings of the National Academy of Science*, 104:708–711, January 2007.
63. Victor A. Vyssotsky, Fernando J. Corbató, and Robert M. Graham. Structure of the Multics Supervisor. In *Proceedings of the AFIPS 1965 Fall Joint Computer Conference (FJCC)*, pages 203–212, Las Vegas, Nevada, November 1965. Spartan Books, Volume 27, Part 1.
64. 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.
65. Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, November 1999.
66. 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.