INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 20 (2013) 1–48
DOI: 10.1111/j.1475-3995.2012.00862.x

Parallel metaheuristics: recent advances and new trends

Enrique Alba[a], Gabriel Luque[a] and Sergio Nesmachnow[b]

[a] *ETSI Informática, Universidad de Málaga, Spain*
[b] *Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay*
E-mail: eat@lcc.uma.es [Alba]; gabriel@lcc.uma.es [Luque]; sergion@fing.edu.uy [Nesmachnow]

Received 23 September 2011; received in revised form 15 May 2012; accepted 12 June 2012

Abstract

The field of parallel metaheuristics is continuously evolving as a result of new technologies and needs that researchers have been encountering. In the last decade, new models of algorithms, new hardware for parallel execution/communication, and new challenges in solving complex problems have been making advances in a fast manner. We aim to discuss here on the state of the art, in a summarized manner, to provide a solution to deal with some of the growing topics. These topics include the utilization of classic parallel models in recent platforms (such as grid/cloud architectures and GPU/APU). However, porting existing algorithms to new hardware is not enough as a scientific goal, therefore researchers are looking for new parallel optimization and learning models that are targeted to these new architectures. Also, parallel metaheuristics, such as dynamic optimization and multiobjective problem resolution, have been applied to solve new problem domains in past years. In this article, we review these recent research areas in connection to parallel metaheuristics, as well as we identify future trends and possible open research lines for groups and PhD students.

Keywords: parallel computing; metaheuristics; analysis of algorithms; communications

1. Introduction

In practice, optimization (and searching and learning) problems are often NP-hard, complex, and time-consuming. Two major approaches are traditionally used to tackle these problems: exact methods and metaheuristics. Exact methods allow exact solutions to be found but are often impractical as they are extremely time-consuming for real-world problems (large dimension, hardly constrained, multimodal, time-varying problems, etc.). On the other hand, metaheuristics provide suboptimal (sometimes optimal) solutions in a reasonable time. Thus, metaheuristics usually allow the resolution delays imposed in the industrial field to be met and the study of general problem classes instead of particular problem instances. In general, many best-performing techniques (in precision and effort) to solve complex and real-world problems use metaheuristics. Their fields of application range from combinatorial optimization, bioinformatics, telecommunications to economics, software engineering, etc., which need fast solutions with high quality (Alba et al., 2009).

Metaheuristics fall in two categories: trajectory-based metaheuristics and population-based metaheuristics (Blum and Roli, 2003). The main difference between these two methods depends on the number of tentative solutions used in each step of the (iterative) algorithm. A trajectory-based technique starts with a single initial solution and at each step of the search the current solution is replaced by another (often the best ones) solution found in its neighborhood. On the one hand, trajectory-based metaheuristics allow a locally optimal solution to be found quickly, therefore they are called exploitation-oriented methods, promoting intensification in the search space. On the other hand, population-based algorithms make use of a population of solutions. In this case, the initial population is randomly generated (or created with a greedy algorithm), and then enhanced through an iterative process. At each generation of the process, the whole population (or a part of it) is replaced by newly generated individuals (often the best ones). These techniques are called exploration-oriented methods because their main ability depends on the diversification in the search space.

Most basic metaheuristics are sequential. Although their utilization allows significant reduction of the temporal complexity of the search process, latter this is commonly used for real-world problems arising in both academic and industrial domains. Therefore, parallelism comes as a natural way not only to reduce the search time but also to improve the quality of the solutions provided. For a discussion on how parallelism can be mixed with metaheuristics, the reader has several sources of information in the literature (Alba, 2005; Luque and Alba, 2011; Talbi, 2006), respectively, devoted to summarize the field, discuss different kinds of algorithms, and focus in parallel genetic algorithms. Each of them illustrates complementary approaches to handle and deploy parallelization in connection to metaheuristics (Ferreira and Pardalos, 1996; Nedjah et al., 2006).

In this article, we include an extensive and organized survey of the recent advances in the parallel metaheuristic domain. We discuss new algorithm models and how these techniques can take advantage of the features in the new parallel platforms. Our contribution is to review scientific papers in the past decade, specially after the work done by Alba (2005) that contains the state of the art of the work done until 2005. Therefore, we here review recent works, especially for the period 2005–2011, explain their merits, include the new advances in theoretical foundations of such algorithms, and define the best practices expected from researchers when dealing with parallel metaheuristics. This survey also identifies and presents the most promising open research lines and trends in this field in the coming years for research groups and PhD students.

After presenting a brief introduction to parallel metaheuristics in Section 2, we organize the existing scientific papers according to different criteria in the following structure of the work. In Section 3, a taxonomy of the different parallel search models is presented. In Section 4, we classify the literature according to the domains of work where the problems that they solve can be found. Later, in Section 5, we describe the advances with respect to the novel parallel computing platforms, such as grid, cloud, GPU (graphics processing unit), APU (accelerated processing unit), multicore, and FPGA (field programmable gate arrays), indicating how parallel metaheuristics can profit from their new features. Then, we include a presentation of parallel methodological issues: a survey about best practices in experimental design, plus some (very needed) theory of parallel metaheuristics in

Section 6. Finally, we describe and discuss the new trends and open research lines related to parallel techniques in Section 7. A summary of conclusions is included in the last section.

2. Introduction to parallel metaheuristics

In this section, we provide a brief survey of classic parallel models found in the literature. This is an important issue, because new approaches for parallel metaheuristics often use these classical techniques as a base for their works. We distinguish between trajectory-based and population-based metaheuristics as the parallel models applied to each one are slightly different.

2.1 Trajectory-based metaheuristics

Metaheuristics for solving optimization problems could be viewed as "walks through neighborhoods" tracing search trajectories through the solution domain of the problem (Crainic and Toulouse, 2002). The well-known metaheuristic families based on the manipulation of a single solution include simulated annealing (SA), tabu search (TS), iterated local search (ILS), variable local search (VNS), and greedy randomized adaptive search procedures (GRASP).

Algorithm 1. Trajectory-based general scheme.

```
Generate($s(0)$); // Initial solution
$t := 0$; // Numerical step
while not Termination_Criterion($s(t)$) do
    $s'(t)$ := SelectMove($s(t)$); // Exploration of the neighborhood
    if ĀcceptMove($s'(t)$) then
      $s(t)$ := ApplyMove($s'(t)$);
    $t := t+1$;
endwhile
```

Walks are performed by iterative procedures that allow moving from one solution to another in the solution space (see Algorithm 1). This kind of metaheuristic performs the moves in the neighborhood of the current solution, i.e., it has a perturbative nature. The walks start from a solution randomly generated or obtained from another optimization algorithm. At each iteration, the current solution is replaced by another selected from the set of its neighboring candidates. The search process is stopped when a given condition is satisfied (e.g., reached a maximum number of moves, found a solution with a target quality, or stuck for a given time).

A powerful way to achieve high computational efficiency with trajectory-based methods is the use of parallelism. Different parallel models have been proposed for trajectory-based metaheuristics, and three of them are commonly used in the literature (see Fig. 1): (i) the parallel exploration and evaluation of the neighborhood (or parallel moves model), (ii) the parallel multistart model, and (iii) the parallel evaluation of a single solution (or move acceleration model).

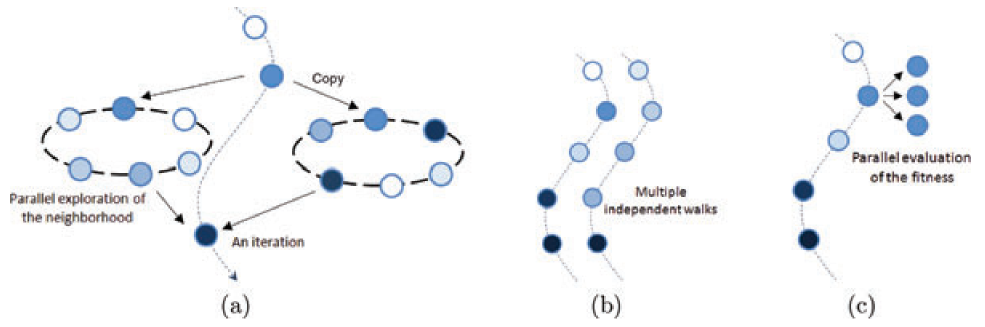
Copy
Parallel exploration of the neighborhood
An iteration
Multiple independent walks
Parallel evaluation of the fitness
(a)
(b)
(c)

Fig. 1. The three classical parallel models for trajectory-based metaheuristics: (a) parallel exploration of the neighborhood (or "parallel moves model"), (b) parallel multistart model, and (c) parallel evaluation of the fitness (or "move acceleration model").

- **Parallel moves model:** It is a low-level master–slave model that does not alter the behavior of the technique. A sequential search would compute the same result but slowly. At the beginning of each iteration, the master duplicates the current solution between distributed nodes. Each one separately manages their candidate/solution and the results are returned to the master.
- **Parallel multistart model:** It involves in simultaneously launching several trajectory-based methods for computing better and robust solutions. They may be heterogeneous or homogeneous, independent or cooperative, start from the same or different solution(s), and configured with the same or different parameters.
- **Move acceleration model:** The quality of each move is evaluated in a parallel centralized way. This model is particularly interesting when the evaluation function can be parallelized as its CPU time-consuming and/or I/O intensive. In that case, the function can be viewed as an aggregation of a certain number of partial functions that can be run in parallel.

2.2 Population-based metaheuristics

Population-based metaheuristics are stochastic search techniques that have been successfully applied in many real and complex applications (epistatic, multimodal, multiobjective, and highly constrained problems). A population-based algorithm is an iterative technique that applies stochastic operators on a pool of individuals (the population) (see Algorithm 2). Every individual in the population is the encoded version of a tentative solution. An evaluation function associates a fitness value to every individual indicating its suitability to the problem. Iteratively, the probabilistic application of "variation operators" on selected individuals guides the population to tentative solutions of higher quality. The well-known metaheuristic families based on the manipulation of a population of solutions include evolutionary algorithms (EAs), ant colony optimization (ACO), particle swarm optimization (PSO), scatter search (SS), differential evolution (DE), evolutionary strategies (ES), and estimation distribution algorithms (EDA).

Algorithm 2. Population-based metaheuristic pseudo-code.

```
Generate($P(0)$); // Initial population
$t := 0$; // Numerical step
**while** not Termination_Criterion($P(t)$) **do**
    Evaluate($P(t)$); // Evaluation of the population
    $P''(t)$ := Apply_Variation_Operators($P'(t)$); // Generation of new solutions
    $P(t+1)$ := Replace($P(t)$, $P''(t)$); // Building the next population
    $t := t + 1$;
**endwhile**
```

For nontrivial problems, executing the reproductive cycle of a simple population-based method on long individuals and/or large populations usually requires high computational resources. In general, evaluating a fitness function for every individual is frequently the most costly operation of this algorithm. Consequently, a variety of algorithmic issues are being studied to design efficient techniques. These issues usually consist of defining new operators, hybrid algorithms, parallel models, and so on (Gallardo et al., 2007; Nguyen et al., 2009; Whitley et al., 2010). We now provide a brief summary of the well-known parallel models used in this field.

Parallelism arises naturally when dealing with populations, since each of the individuals belonging to it is an independent unit [at least according to the Pittsburg style (De Jong et al., 1993), although there are other approaches such as Michigan one that do not consider the individual as an independent unit (Wilson, 1995)]. Indeed, the performance of population-based algorithms is often improved when running in parallel. Two parallelizing strategies are specially focused on population-based algorithms (see Fig. 2): (1) parallelization of computations, in which the operations commonly applied to each of the individuals are performed in parallel; and (2) parallelization of population, in which the population is split into different parts that can be simply exchanged or evolved separately, and then joined later.

In the beginning of the parallelization history of these algorithms, the well-known "master–slave" (also known as "global parallelization" or "farming") method was used. In this approach, a central processor performs the selection operations while the associated slave processors (workers) run the

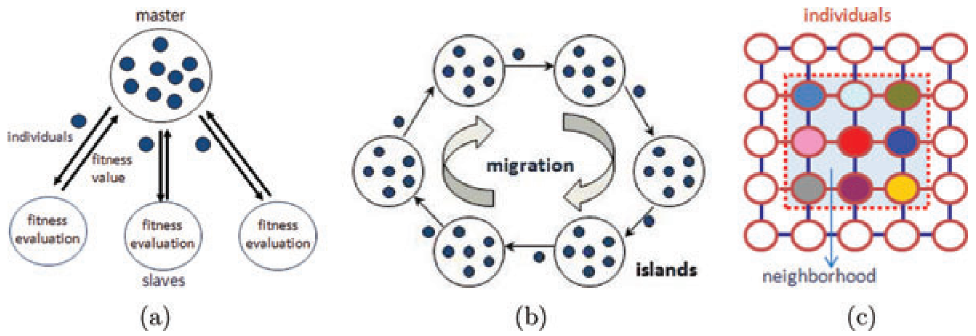

Fig. 2. Most important classical parallel models for population-based metaheuristics: (a) master–slave, (b) distributed, and (c) cellular models.

variation operator and the evaluation of the fitness function. This algorithm has the same behavior as the sequential one (as shown in Fig. 2a), although its computational efficiency is improved, especially for time-consuming objective functions. On the other hand, many researchers use a pool of processors to speed up the execution of a sequential algorithm, only because "independent runs" can be made more rapidly by using several processors than by using a single one. In this case, no interaction at all exists between the independent runs. This is a very interesting idea for any research laboratory because it improves the productivity of researches.

However, actually most parallel population-based techniques found in the literature utilize some kind of spatial disposition for the individuals, and then parallelize the resulting chunks in a pool of processors. Among the most widely known types of structured metaheuristics, the "distributed" (or coarse-grain) and "cellular" (or fine-grain) algorithms are very popular optimization procedures (Alba and Tomassini, 2002). In the case of distributed ones (Fig. 2b), the population is partitioned in a small set of subpopulations (islands) in which isolated serial algorithms are executed. Sparse individual exchanges are performed among these islands with the goal of introducing some diversity into the subpopulations, thus preventing search of getting stuck in local optima. In order to design a distributed metaheuristic, we must take several decisions. Among them, the main decision is to determine the parameters related to migration policy: topology (logical links between the islands), migration rate (number of individuals that undergo migration in every exchange), migration period (number of performed steps in every subpopulation between two successive exchanges), and the selection/replacement of the migrants. In the case of a cellular method (Fig. 2c), the concept of "neighborhood" is introduced, so that an individual may only interact with its nearby neighbors in the breeding loop. The overlapped small neighborhood in the algorithm helps in exploring the search space because a slow diffusion of solutions through the population provides a kind of exploration, while exploitation takes place inside each neighborhood (Alba and Dorronsoro, 2008). Also, hybrid models have been proposed in which a two-level approach of parallelization is undertaken. In general, the higher level for parallelization is a coarse-grain implementation (i.e., a set of islands) and each island performs other parallel model such as cellular, master–slave method, or even another distributed method (see an example in Fig. 3).

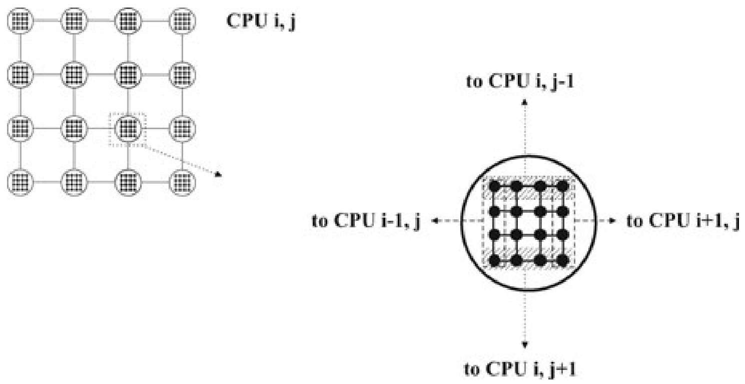

Fig. 3. Example of using a distributed model for cellular methods (Luque et al., 2009).

3. Recent parallel models for metaheuristics

The first idea that we must reinforce while researching with parallel metaheuristics is the difference between the model and its implementation. The model defines the behavior of the method. The same model could be implemented in different ways and on different parallel platforms. In the past, some parallel models were proposed for specific parallel platforms, such as cellular algorithms developed for MMP (massively parallel machine) architectures. However, this model can be successfully implemented on other platforms such as sequential ones (Alba and Dorronsoro, 2005) or clusters (Luque et al., 2009). It is also true that, recently, several models have been proposed using the special features of new parallel technologies, such as the systolic model for GPU platforms (Alba and Vidal, 2011) and that targeting the model to the architecture is one way to solve problems in this domain. In this section, we focus on the existing parallel models for metaheuristics, while in Section 5 we separately deal with implementation issues.

Although the distributed model (or parallel multistart model in the case of trajectory-based techniques) has been a very well-known model for decades (Tanese, 1989), it continues being one of the frequently used models in the literature due to its easy implementation on clusters, multicores, and other parallel platforms, and because of its accuracy for a significantly large number of problems (see next section). The recent research in this model is focused on the seeding of the islands (Muelas et al., 2008) and on the migration policy: studying different topologies (Hijaze and Corne, 2009), using additional information in the migration phase (Araujo et al., 2009), building self-adaptive migration models (Araujo and Merelo, 2009; Lässig and Sudholt, 2011), and introducing advanced techniques during the communication phase in the case of single-solution search methods (Luque et al., 2010, 2011). Also, very simple parallel variants of distributed models such as a noncooperative versions for trajectory-based algorithms have been proposed recently (Shylo et al., 2011) showing very accurate results.

Another search model that has received a large interest from the scientific community in the last 6 years is the cellular model (Alba and Dorronsoro, 2008). Some of the papers in this field run purely sequential, and try to design an efficient dynamics that makes a good trade-off between exploration and exploitation (Alba and Dorronsoro, 2005; Nguyen et al., 2009). These works propose a self-adaptative method to change the balance between exploration and exploitation, but while the first one (Alba and Dorronsoro, 2005) incorporates a mechanism that changes the shape of the grid, the second one (Nguyen et al., 2009) defines a memetic algorithm (ACMA) where the application of the local improvement (a trajectory-based metaheuristic) is regulated by the distribution or diversity of a population. Following this last schema, Segura et al. (2011a) proposes a parallel hybridization among parallel population algorithms and parallel single-solution methods.

The main parallel issues investigated in population-based algorithm family are related to the design of efficient models of cellular methods that can be executed in actual parallel platforms. In this case, we can find the distributed model proposed in Luque et al. (2009) or the grid-based hybrid cellular GA defined in. In both papers, the authors propose a hybrid parallel model that allows to port the cellular model to parallel hardware. As it is shown in Fig. 3, the first work (Luque et al., 2009) proposed a cellular algorithm (apcGA) in which the complete population is divided into several asynchronous islands, allowing to make an efficient use of cluster platforms, although the global behavior is similar to a cellular GA. The algorithm (PEGA) defined Dorronsoro et al. (2007)

in uses a master–slave model to distribute the most consuming operations (fitness evaluation and application of the operators) among the different processors of the parallel platform that is a grid system.

Also, some new parallel algorithms have been proposed following the parallel moves strategy (or master/slave) for both population- and trajectory-based metaheuristics, but these algorithms usually are classical ones adapted to be used in modern parallel platforms, mainly GPUs (Harding and Banzhaf, 2009; Van Luong et al., 2010; Tsutsui and Fujimoto, 2009), and they will be detailed in next sections.

Finally, we end this section by discussing several models proposed to deal with the challenges of new domains such as multiobjective optimization (MO) (Coello et al., 2007; Deb, 2001) and dynamic environments (Branke, 2001). In short, researchers use well-known models such as cellular or distributed ones, but by adding some extensions to take into account the special features of these domains.

For the first domain, the multiobjective one, we can find some cellular models such as Zhang and Li (2007) and Zhang et al. (2009) that extend the classical model using decomposition, or Nebro et al. (2006, 2009) that include an additional archive of nondominated solutions to the canonical cellular model. These extensions allow the cellular algorithm to analyze and maintain nondominated solutions in different parts of the search space. Also, some distributed and hybrid models have been used to solve multiobjective problems applying the canonical model (Hiroyasu et al., 2005, Jaimes and Coello, 2007 and Luna et al., 2006) or implementing subalgorithms (Nebro and Durillo, 2010), where each one optimizes one or more objectives. Finally, there are also some works using a master–slave model to parallelize state-of-the-art multiobjective algorithms such as NSGA-II (Durillo et al., 2008) or MOEA/D (Nebro and Durillo, 2010).

For dynamic optimization (the optimized function changes as the algorithm seeks at finding the global optimum), the most popular parallel model is the distributed or multipopulation model, because it helps to maintain a high diversity, which is a key feature in this domain. Some recent examples are Blackwell and Branke (2004), Du and Li (2008), Khouadjia et al. (2011), and Mendes and Mohais (2005). Also, the cellular model has been applied to this optimization field (Alba et al., 2007e), showing promising results that outperform the results computed with other classical methods.

4. Modern applications solved by parallel metaheuristics

Parallel metaheuristics have been shown to be useful in practice to solve a large number of applications. Many real-life problems may need days or weeks of computing time to be solved on serial machines. This is the usual scenario when considering difficult problems such as complex combinatorial problems with large search spaces, multiobjective problems with many hard-to-evaluate objective functions, or dynamic optimization problems. Although the intrinsic time complexity of a problem cannot be lowered by using a finite amount of computing resources, parallel computing techniques often allow reducing the resolution times to reasonable levels. This is a very important advantage in an industrial or commercial setting, where the time to compute a solution is instrumental for decision-making and competitiveness. Furthermore, the new models offered by structured populations often allow performing a better exploration of alternative solutions of the search space, taking advantage of the diversity enhancements, speciation-like features, and cooperative facilities. These features of parallel metaheuristics allow researchers to cope with a large class of difficult problems, where sequential metaheuristics tend to perform poorly or are difficult to apply.

Parallel metaheuristics have been used successfully in operations research, engineering, manufacturing, telecommunication, and many other application domains. It is impossible to completely review such vast work, in fact this would need a separate survey to deal with the wide spectrum of optimization tasks in which parallel metaheuristics are being used with success. However, for the sake of completeness, we do provide a global description of the main successful and real-world applications in very relevant fields of the optimization with this kind of parallel methods.

In the academic branch of optimization problems, many papers have focused on providing experimental results based on well-known benchmark problems, such as routing (TSP—Bai et al., 2009; Delisle et al., 2005b, 2009; Gao and Dong, 2010; Hung and Chen, 2010; Li et al., 2010; Xiong et al., 2010), assignment (SAT—Luo and Liu, 2006, MAXSAT—Munawar et al., 2009; Sadeg and Drias, 2007, QAP—James et al., 2005; Tsutsui, 2007, 2008; Tsutsui and Fujimoto, 2009), covering and partitioning (He et al., 2007), topological mathematical problems (Catalá et al., 2007; He et al., 2007; Jovanovic et al., 2010; Kokosiński and Kwarciany, 2007; Mocholí et al., 2005; Taskova et al., 2010), and other classical problems (Zhu and Curry, 2009). These works have been mainly devoted to demonstrate the effectiveness of the parallel models for metaheuristics to outperform the traditional sequential implementations, regarding both computational efficiency metrics and quality of results. Traditionally, most parallel metaheuristics have been proposed for solving combinatorial optimization problems (characterized by discrete decision variables and a finite search space), but in the last years many works have also tackled problems using real encoding, specially when solving multiobjective problems.

Regarding real-world problems, parallel metaheuristics have been used in many scientific, industrial, and commercial application domains. Among the most important ones in recent years, we can mention the following:

- *Automation and robotics*, where accurate and efficient learning algorithms to control and use the information technologies are required to reduce the need for human presence. Here, parallel metaheuristics provide a decisive help to tackle learning problems that handle large volume of data (Bouamama, 2010), and those control problems that involve complex training procedures (Hereford, 2006, 2010; Huang et al., 2009).
- *Bioinformatics*, an emergent scientific field where parallel models of metaheuristics are helpful tools to cope with computationally expensive optimization problems in molecular biology that often also need to manage very large amount of data, such as sequence alignment (Gomes et al., 2008; Zola et al., 2006), DNA sequencing (Hongwei and Yanhua, 2009; Wirawan et al., 2008), gene finding (Rausch et al., 2008), genome assembly (Alba and Luque, 2006b; Nebro et al., 2008b), drug design (Boisson et al., 2008), protein structure alignment/prediction (Chu and Zomaya, 2006; Guo et al., 2009; Islam and Ngom, 2006; Tantar et al., 2007), phylogenetic inference (Blagojevic et al., 2007; Cancino et al., 2010; Grouchy et al., 2009), and other related problems (Guarracino et al., 2006; Martins et al., 2006; Nebro et al., 2008a).
- *Engineering design*, where systems have many components, a large design space, and they usually involve functions with huge computation demands. These characteristics make parallel

metaheuristics one of the most promising alternatives to get accurate solutions in reasonable execution times for complex tasks such as aerodynamic optimization and airfoil design (Asouti and Giannakoglou, 2009; Lim et al., 2007), design optimization of turbomachinery blade rows (Sasaki et al., 2006), electronic circuit and VLSI design (Alba et al., 2007b; Lau et al., 2006; Sait et al., 2007, 2008), antenna design (Kalinli et al., 2010; Weis and Lewis, 2009), signal processing (Li et al., 2007b), etc.

- *Hydraulic engineering*, where parallel metaheuristics have been used to efficiently deal with real-world scenarios arising in water supply network design optimization (López-Ibánez, 2009), groundwater source identification (Babbar and Minsker, 2006; Mirghania et al., 2009; Sinha and Minsker, 2007), and multiobjective groundwater problems (Tang et al., 2007b).
- *Information processing, classification, and data mining*, where parallel metaheuristics significantly help with the main challenge in this field, which is related to dealing with huge volumes of data, such as in feature selection and classification (Hamdani et al., 2006; Lopez et al., 2006), classification rules discovery (Chen et al., 2006; Chintalapati et al., 2010; Roozmand and Zamanifar, 2008), data mining (Langdon, 2010), clustering (Gunes and Sima, 2010), natural language processing (Alba et al., 2006), and other problems.
- *Manufacturing and industrial applications*, an area in which productivity plays a major role for the increase of competitiveness in nowadays globalized economies. Emerging technologies are continuously providing new advanced manufacturing processes, and also new challenges. In this context, several problems have recently been tackled with parallel implementations of metaheuristics, including warehouse location and placement problems (Almeida-Luz, 2009; Byun et al., 2009; Homberger, 2008; Homberger and Gehring, 2008), steel industry (Zhao et al., 2011), packing problems (León et al., 2009; Peng et al., 2006; Segura et al., 2011b), and assembly line balancing problems (Ozbakir et al., 2011).
- *Routing, logistics and vehicle planning*, a field where sophisticated methods are needed to manage the flow of resources, especially for large and complex problem instances. Parallel metaheuristics have been recently applied to logistic problems (Fang and Wu, 2010; Li and Bai, 2010; Liefooghe, 2010), facility location (Subramanian et al., 2010; Wang et al., 2008), site location (Zhao et al., 2010), vehicle routing (Doerner et al., 2005, 2007; Ellabib et al., 2007; Khouadjia et al., 2011; Lucka and Piecka, 2009; Yu et al., 2011), emergency vehicle fleet management (Ibri et al., 2010), bus network optimization (Yang et al., 2007), and path planning (Allaire et al., 2009), among others problems in this area.
- *Scheduling*, which is a key class of planning problems to provide a correct service on deciding how to commit resources to a group of tasks. Scheduling is an important tool in many application areas (e.g., industrial, manufacturing), where it can have a major impact on the productivity of a process. Parallel metaheuristics have been applied to traditional scheduling problems, such as timetabling (Mansour and Haidar, 2010), job shop and flow shop scheduling (Aydin and Sevkli, 2008; Bozejko et al., 2010; Bozejko and Wodecki, 2008; Melab et al., 2006a), as well as other problem variants (Alba et al., 2007a; Da Silva and Ochi, 2009; Liefooghe, 2010). Recently, scheduling has become a relevant problem in large modern parallel computing infrastructures such as grid, volunteer computing, and cloud computing environments. In these services, parallel metaheuristics are efficient methods to compute large scheduling in reduced execution times (Alba et al., 2007d; Nesmachnow et al., 2011, 2012a, 2012b; Xhafa and Abraham, 2008).

- *Software engineering and software development*, a field in which parallel metaheuristics have just started to show their usefulness for exploring large search spaces, such as in the optimization of dynamic data types and memory managers in embedded systems (Risco-Martin et al., 2008, 2009), and in software testing (Alba and Chicano, 2008).
- *Telecommunications*, a field that has grown at a fast pace in recent years, posing difficult challenges to the research community due to the large size of the infrastructures, the need for real-time results, etc. Parallel metaheuristics have shown a great impact on addressing these challenges by providing accurate and efficient solutions to the related optimization problems in network design (Alba and Chicano, 2005; Nesmachnow et al., 2007; Pedemonte and Cancela, 2010; Ribeiro and Rosseti, 2007), network routing (Durillo et al., 2008; Liu et al., 2008a; Segura et al., 2009; Zhu et al., 2010), and network planning, especially in modern networks technologies such as cellular (Alba and Chicano, 2005; Talbi et al., 2007), mobile ad-hoc networks (Liu et al., 2008a), vehicular networks, sensor networks, and peer-to-peer (Luna et al., 2008; Nesmachnow et al., 2009).

Parallel implementations of EAs, ACO, and PSO have been the preferred choices for efficiently solving optimization problems related to real-world applications, while parallel trajectory-based metaheuristics (SA, LS) have been also used as viable second options.

Besides the application domains previously highlighted, in the last few years parallel implementations of metaheuristics have been also successfully applied in many other areas such as energy and power network optimization (Peng et al., 2010; Zhao et al., 2005), health and medicine (Karnan and Gopal, 2010), strategic and military applications (Gao et al., 2010), economy and finance (Liu et al., 2008b), workforce planning (Alba et al., 2007d), and image processing (Cardenas et al., 2010; Harding and Banzhaf, 2008; Peng et al., 2006) and many other optimization problems (Alba et al., 2009; Crainic et al., 2009). This shows the growing research in parallel metaheuristics, and therefore we can conclude that the near future will witness many more real-life situations and problems tackled using parallel metaheuristic algorithms.

5. Technologies for parallel metaheuristics

When studying a parallel algorithm, it is important to take into account on which computing platform it has been implemented, as the hardware architecture notably impacts the time required to perform the computations, communications, synchronizations, and the data sharing. Until the last decade, the classic proposals of parallel metaheuristics focused on traditional supercomputers and clusters of workstations. Currently, the novel emergent parallel computing architectures such as multicore processors, graphic processing units (GPUs), or grid environments, provide new opportunities to develop parallel computing techniques to improve problem solving and to lower the required computation times. This section introduces popular parallel computing platforms, providing the main concepts about the implementation of parallel metaheuristics in each hardware, and describing the most significant recent works in the field, a key issue for developers and users. The section also summarizes some important issues about software tools for implementing parallel metaheuristics.

5.1 Parallel hardware platforms

Among the many classifications proposed for parallel computing platforms, the most used is the traditional taxonomy by Flynn (1972). This classification distinguishes four categories regarding the way that the instructions and the data streams are handled: single instruction-single data (SISD), single instruction-multiple data (SIMD), multiple instruction-single data (MISD), and multiple instruction-multiple data (MIMD).

Traditional SIMD architectures (shared memory multiprocessors) were mostly used in the 1980s and the beginning of the 1990s, while MIMD (e.g., networks of workstations) were the leading platforms in parallel computing during the 1990s. In the last decade, new parallel computer architectures have emerged. These novel architectures can be classified in three main groups: (i) programmable circuits (such as FPGA, field programmable gate arrays), (ii) multiprocessor computers (including multicore computers, multikernel computers, and GPUs), and (iii) distributed computing platforms (clusters of computers, grid, and cloud computing environments).

Parallel implementations of metaheuristics have been proposed for all classical architectures of parallel computers (Alba, 2005). In recent years, the design and implementation of parallel metaheuristics have followed important trends in the field of parallel hardware, which have been focused on the previously mentioned modern computing platforms.

5.1.1 Programmable circuits

Before 2005, FPGAs were sporadically used as a viable choice to develop hardware-based parallel implementations of metaheuristics (Alba, 2005). The main advantage of this infrastructure consists in the large number of programmable logical elements, able to compute simple logical and mathematical operations. FPGA also provides a programmable interconnection hierarchy for the logical elements, which allows implementing sophisticated algorithms. A significant lower number of FPGA implementations of parallel metaheuristics have been proposed in the last 5 years, mainly because they provide less flexibility than the new hardware platforms for software-based implementations of metaheuristics such as GPUs. FPGAs have been used when specific real-time requirements to find the solution for an optimization problem are formulated (Allaire et al., 2009; Walton et al., 2010), or just to achieve an improved efficiency (Farmahini-Farahani et al., 2010; Huang et al., 2009; Mohammed et al., 2011). The cellular model for parallel metaheuristics has been the most popular choice to implement in FPGAs, because the logical elements and the reprogrammable interconnection make it easy to implement its population organization and interactions (Fernando et al., 2008; Jewajinda and Chongstitvatana, 2008). Other works have proposed low-level implementations of parallel metaheuristics for specific processors such as the cell broadband engine (Blagojevic et al., 2007; Perez et al., 2009; Wirawan et al., 2008).

5.1.2 Multiprocessor architectures

Multiprocessor parallel platforms are a useful choice to develop parallel implementations of metaheuristics. The availability of multiple computing resources integrated in a single device allows to deal with optimization and learning problems that require a fast solution, or even real-time efficiency, in an easy-to-program platform. In this kind of hardware, parallel metaheuristics are often implemented using the shared-memory paradigm for parallel computing programming, which uses a common resource (the shared memory) to perform the synchronization and communication between multiple threads and processes. In the field of parallel metaheuristics, threads and light processes have been usually employed to parallelize both cooperative and semi-cooperative versions of multipopulation and multitrajectory methods, as well as hybrid algorithms combining population-based metaheuristics with local search operators.

Multicore platforms. The new generation of shared memory multicore platforms with multiple-threads parallel execution has provided a promising hardware platform for implementing new parallel metaheuristics. Population-based methods such as EAs, ACO, and PSO are natural candidates to exploit multithread architectures to take advantage of both implicit and explicit parallelism. Also, trajectory-based methods can take benefit of multithreading programming, e.g., by implementing the multiple walks parallel model. Implementations of parallel metaheuristics have been developed in a wide range of multithread platforms, including simple dual-core computers to modern computing servers with up to 24 cores, and also massively parallel processing (MPP) computers with thousands of cores. When using these parallel infrastructures, the communications and synchronizations are performed using the shared memory approach. The main advantage of multicore computers is that it is possible to implement real-time parallel metaheuristics, since the proximity of multiple CPU cores on the same chip allows to operate at a much higher clock-rate than traditional parallel computing using several computers. The universal availability of multiple cores in desktop, laptop, and even smartphones platforms means good news for the future of parallel metaheuristics on multicore hardware. In these multicore platforms, a significant rise of performance is usually achieved in lower response-times when running CPU-intensive processes, such as the ones required to solve optimization problems with complex mathematical functions. On the other hand, specific adjustments to the operating system support and/or to existing parallel metaheuristics software libraries can be required to fully exploit the parallelism. Researchers using existing libraries should check how different is to tune them to profit from multiple cores in the same computer.

Fine-grain implementations of parallel metaheuristics are well-suited to the multithreading approach, since the shared memory allows fast communications between individuals and subprocesses. Some parallel metaheuristics with recent implementations of the fine-grain model in multicore architectures includes EAs (Munawar et al., 2008), EDAs (Perez et al., 2009), MOEAs (Nebro and Durillo, 2010), ACOs (Tsutsui and Fujimoto, 2010), VND/ILS (Subramanian et al., 2010), TS and several other metaheuristics (Bozejko et al., 2008). The master–slave model for parallel metaheuristics has also been implemented in multicore processors, for ACO (Delisle et al., 2005a, 2005b; Guo et al., 2009; López-Ibánez, 2009; Tsutsui, 2008; Tsutsui and Fujimoto, 2010), EA (Cardenas et al., 2010), TS and branch and bound (Hung and Chen, 2010); in these methods, the main advantage is the ability of computing the fitness evaluation in parallel by using several threads. Multicore multipopulation methods have also been proposed for several metaheuristics, such as ACO (Delisle et al., 2009; Gao et al., 2010; Li et al., 2010; Lucka and Piecka, 2009; Xiong et al., 2008, 2010), EAs (Byun et al., 2009; He et al., 2007; Tsutsui, 2010), PSO (Tu and Liang, 2011), and the parallel artificial bee colony algorithm (Narasimhan, 2009). When using multiple populations, the shared-memory approach is used to perform the specific operators that communicate and synchronize the populations, allowing an efficient cooperative search. EAs have also been implemented following other ad-hoc decomposition approaches, such as dividing a problem into smaller parallel subproblems (Vrajitoru, 2010).

Graphic Processing units. Currently, GPU cards offer teraflop computing at low cost (less than one thousand dollars), allowing large-scale SIMD or SPMD computing with significant speed ups in the execution of graphic applications. The novel general purpose on GPU (GPGPU) paradigm extends this advantage to other applications areas (Owens et al., 2007). In particular, GPUs dramatically increase the computational speed for the resolution of real-world optimization problems tackled with parallel metaheuristics, and they allow researchers to face sophisticated problems and complex scenarios. The beneficial computing power/cost relationship has positioned GPU computing at the front of the next generation in high-performance computing infrastructures.

In GPGPU, the massive floating-point computational power of a GPU shader is turned into a very scalable computing resource for stream processing, able to obtain a significantly higher performance than the CPU. Actually, both modern supercomputers and desktop computers can take advantage of GPU acceleration, by using APIs (such as NVIDIA CUDA and GPL) to design parallel programs on GPU architectures. Applications are frequently based on map/reduce and scatter/gather operations performed on static and dynamic arrays. In the parallel metaheuristics field, GPUs have been used for accelerating the fitness evaluation in genetic programming and in population-based methods where complex fitness functions are used. In genetic programming, the trend is to simultaneously execute several copies of the same executable program in the GPU, taking advantage of the SIMD parallel model to lower the time to compute one generation (Harding and Banzhaf, 2007, 2008; Krömer et al., 2011; Langdon, 2010; Robilliard et al., 2009), but the use of interpreters (Langdon and Banzhaf, 2008), hybrid methods that combine CPU and GPU (Lewis and Magoulas, 2009), and clusters of workstations with GPU (Harding and Banzhaf, 2009) have also been proposed.

Regarding other metaheuristic techniques, both multitrajectory and distributed multipopulation models have been implemented on GPU, often following a fine-grain approach in order to fully exploit the computing capabilities of the large number of threads in the GPU cores. In these models of computation, the cooperation among threads is exclusively performed using the shared memory paradigm. EAs have been the preferred metaheuristic to parallelize on GPU, including the fine-grain master–slave model that implements the parallel fitness evaluation for EAs (Li et al., 2007a; Tsutsui, 2007; Wong and Wong, 2009; Yu et al., 2005), ES (Zhu, 2009) and hybrid EAs (Man-Leung and Tien-Tsin, 2006; Munawar et al., 2009), the cellular model (Luo and Liu, 2006; Vidal and Alba, 2010a), and the island based model (Luong et al., 2010; Maitre et al., 2009; Pospíchal et al., 2010a, 2010b; Risco-Martin et al., 2009). Proposals of GPU implementations for other metaheuristics have also been recently presented, such as the fine-grain parallel fitness evaluation in single-thread methods (Bozejko et al., 2009), the parallel independent runs of ACO (Bai et al., 2009), the master–slave parallel ACO (Catalá et al., 2007; Fu et al., 2010; Zhu and Curry, 2009), the fine-grained parallel immune algorithms (Li et al., 2009; Zhao et al., 2011), and the two-level approach for parallel metaheuristics (Bozejko et al., 2010).

Hybrid methods that combine CPU and GPU computations have also been used. In this parallel approach, usually the local search method to improve the solutions is run on the GPU, while the remaining tasks are executed on the CPU. Two examples of this hybrid CPU/GPU implementations are the coarse-grain master–slave ACO with local search (Zhu and Curry, 2009) and the parallel master–slave ACO (Catalá et al., 2007). Further new hardware such as APUs devices (2010) will come to change this picture allowing hardware heterogeneity that combines CPU and GPU programming, opening new challenges to profit from it and create heterogeneous parallel algorithms.

Most of the previously mentioned GPU implementations of parallel metaheuristics have been developed using NVIDIA and ATI programmable GPUs, but genetic programming has also been implemented in nontraditional heterogeneous hardware such as XBox360 and Zune (Wilson and Banzhaf, 2008, 2009).

5.1.3 Distributed computing architectures

Starting from small aggregations of homogeneous computers in the 1990s, as on today distributed computing environments include platforms formed by hundreds or thousands of heterogeneous computing resources widespread around the globe, providing the computing power needed for solving complex problems arising in many areas of application. Nowadays, a common platform for distributed computing usually comprises a heterogeneous collection of computers able to work cooperatively (cluster). At a higher level of abstraction, the expression "grid computing" has become popular to denote the set of distributed computing techniques that work over a large loosely coupled virtual supercomputer, formed by combining together many heterogeneous components of different characteristics and computing power. This infrastructure has made it feasible to provide pervasive and cost-effective access to a collection of distributed computing resources for solving problems that demand large computing power (Foster and Kesselman, 1998). Recently, "cloud computing" has emerged as a novel paradigm of location- and device-independent processing, where physical resources provide computing, software, and access to data, on demand (Velte et al., 2010). Distributed computing platforms provide a fully scalable environment that allows researchers to handle large optimization and learning problem instances within reasonable computing times. In this kind of infrastructure, parallel distributed metaheuristics are very often implemented following a message passing paradigm, usually for parallelizing multipopulation cooperative algorithms.

Clusters. Many implementations of parallel metaheuristics have been developed on cluster architectures, most of them following the cooperative parallel model that uses more than one population (the "distributed subpopulations" model). This approach provides a cooperative search mechanism that often allows obtaining superior results than the sequential model, as well as outperforming other parallel metaheuristics, by taking advantage of the multiple search and the increased diversity provided by a multipopulation model executing on a set of processors. Cluster computing platforms provide the most natural way to parallelize metaheuristics using a traditional hardware with a very good performance/cost ratio. In addition, the research on libraries for parallel computing on clusters is consolidated, and several high-level frameworks for parallel/distributed metaheuristics have been built on them (Parejo et al., 2012). While using cluster computing platforms, the cooperation and synchronization between the multiple processes that search in parallel is usually performed using the message passing paradigm for parallel computing.

In cluster architectures, recent implementations of distributed subpopulations metaheuristics included EAs (Alba et al., 2004; Alba and Luque, 2006b; Homberger, 2008; Homberger and Gehring, 2008; Huang et al., 2009; Jiménez et al., 2010; Nesmachnow et al., 2007, 2011, 2012a, 2012b), hybrid EAs (Alba et al., 2007d; Gaifang and Xueliang, 2010), multicolony ACO (Chu and Zomaya, 2006; Hongwei and Yanhua, 2009; Taskova et al., 2010; Xiong et al., 2008, 2010; Yang et al., 2007), and

PSO (Bouamama, 2010; Hereford, 2006), among others. Some implementations of the master–slave parallel model for ACO have been developed in clusters (Doerner et al., 2005, 2006; Peng et al., 2005, 2010; Weis and Lewis, 2009), as well as hybrids combining ACO and TS (Ibri et al., 2010). The cellular model has also been applied to EAs (Alba et al., 2007e; Luque et al., 2009) and ACO (Pedemonte and Cancela, 2010) in cluster platforms.

Regarding trajectory-based metaheuristics, some relevant implementations include parallel path-relinking (James et al., 2005), parallel SA (Lukasik, 2008; Zola et al., 2006), parallel cooperative GRASP (Ribeiro and Rosseti, 2007), parallel approaches for VNS (Aydin and Sevkli, 2008), parallel strategies for stochastic evolution (Sait et al., 2008), parallel multistart VNS/VND/ILS (Subramanian et al., 2010), and parallel SS (Mansour and Haidar, 2010). In addition, hybrid and/or cooperative algorithms combining two or more metaheuristics have been proposed, such as the COSEARCH metaheuristic (Talbi and Bachelet, 2006), a combination of TS/SA/GA (Cadenas et al., 2009), a combination of DE and evolutionary programming (Georgieva and Jordanov, 2008), and a parallel hybrid GRASP/GA using reinforcement learning (Dos Santos et al., 2009).

Multiobjective problems have also been tackled with parallel/distributed metaheuristics in clusters, allowing a better exploration of the search space while also avoiding the computational bottleneck associated to the fitness evaluation for complex real-world problems. Some of the recent works in this area include several multiobjective parallel metaheuristics applied to logistics (Liefooghe, 2010), master–slave MOEAs (Durillo et al., 2008), distributed subpopulations MOEAs (Boisson et al., 2008; Cancino et al., 2010; Nesmachnow and Iturriaga, 2012; Sasaki et al., 2006), cellular MOEAs (Zhang and Li, 2007; Zhang et al., 2009), DE (Mendes and Mohais, 2005), and parallel multiobjective PSO (Du and Li, 2008; Mostaghim, 2010).

Actually, cluster platforms built using multicore computers allow the researchers to develop two-level implementations of parallel metaheuristics. This approach combines a coarse-grained parallelization (e.g., the distributed subpopulation model) implemented with the message passing paradigm, and a fine-grained parallel model implemented following the shared-memory approach within each multicore computer (Nesmachnow et al., 2012a). This kind of implementations significantly improves the efficiency of the search in parallel metaheuristics, taking full advantage of the computing power availability.

P2P computing. Peer-to-peer (P2P) computing is a paradigm for distributed computing that shares the workload between equally privileged peers. These peers make their own computing resources available to other peers in the network, without any centralized coordination. The research on parallel P2P implementations of metaheuristics have just recently started. Some implementations have been proposed to take advantage of the cooperation model in P2P networks, by parallelizing hybrid multiobjective metaheuristics (Melab et al., 2006b), EAs (Laredo et al., 2008a, 2008b) and hybrids combining GA and Branch & Bound (Bendjoudi et al., 2008). The results showed that the distributed P2P model needs significantly less computational effort than other traditional architectures, thus making it suitable for tackling large instances of the faced problems.

Grid computing. In the last years, grid computing environments have been used to solve complex optimization problems with parallel metaheuristics. The main contribution of the distributed computing paradigm in grid is the exploitation of large-scale availability of computing resources that allows facing hard-to-solve problems. However, grid environments also provide the benefit of true (geographical) decentralization, improving the fault-tolerance of parallel metaheuristics. When executing in a grid environment, parallel metaheuristics can be implemented by adapting existing metaheuristic frameworks to work in large distributed systems, or by using ad-hoc programs including high-level grid/web services.

The most usual grid approach combines a wide-spectrum method (such as an EA), which significantly reduces the number of visited points in the search space, with a local search/intensification method to focus the search. This approach allows reducing the number of required communications between the geographically distributed subpopulations. Parallel implementations of EAs adapted to execute in grid environments have been often used in recent years, including the traditional master–slave model (Durillo et al., 2008; Nebro et al., 2008b), the distributed subpopulation model (Luna et al., 2008; Melab et al., 2006a; Talbi et al., 2007) and the cellular model (Dorronsoro et al., 2007; Luque et al., 2009). Hierarchical parallel models (Lim et al., 2007) and parallel hybrid MOEAs (Tantar et al., 2007) have also been proposed.

An alternative for executing parallel metaheuristics in grid computing environments consists in applying ad-hoc methods using high-level services, instead of adapting existing frameworks. This model has been applied in parallel ACO algorithms following the multicolony model (Mocholí et al., 2005) and the parallel evaluation of solutions (Weis and Lewis, 2009); and in the parallel GRASP implementation that executed in the grid environment provided by a virtual organization of the European project EGEE Almeida-Luz (2009).

The efforts carried out to improve the efficacy while reducing the computing time have led to explore the use of volunteer computing platforms. This type of grid/distributed computing infrastructure is able to combine the computer power provided by thousands of computers using a simple middleware that is independent of the computation. In this line of research, two recent works (Cole et al., 2010; Lombrana et al., 2009) presented parallel EAs conceived to execute over the Berkeley Open Infrastructure for Network Computing (BOINC) infrastructure, gathering the computational resources required to face hard-to-solve optimization problems.

Cloud computing. The novel cloud computing paradigm for execution of parallel applications relies on providing computational resources in demand using Internet or some other high-speed network. The model offers a set of abstract services – including computation, software, data access, and storage – conceived to be fully scalable, that can be efficiently provisioned and released (Buyya, 2009). Enterprises such as Microsoft, Amazon, and Google have been playing a very important role in the developing of cloud computing infrastructures and services such as Amazon EC2 or Google's App Engine.

Some recent works (Jin et al., 2008; Llorà et al., 2010; Verma et al, 2010) have demonstrated the potential utility of the novel cloud computing infrastructures for the parallelization of metaheuristics. By applying standard data-intensive computing frameworks (such as Hadoop and Meandre) to population-based metaheuristics (GA and EDA), efficient parallel implementations were designed, and they were able to achieve linear speedup for large-scale problems. However, since EAs cannot be directly expressed by the map/reduce model of computation, the previous works showed that when using the current state-of-the-art cloud computing paradigm, modifications of either the frameworks (Jin et al., 2008) or the EAs (Llorà et al., 2010; Verma et al, 2010) are needed.

Determining the adequate mechanisms for developing efficient and scalable parallel metaheuristics implementations in cloud computing environments is one of the main challenges for researchers in this line of work for the next years.

5.2 Software issues and tools

This section presents the main issues concerning the design and implementation of parallel metaheuristics. Our survey covers from ad-hoc low-level implementations of specific parallel metaheuristics to generic frameworks that implement many metaheuristics.

5.2.1 Ad-hoc parallel metaheuristics implementations

In the past years, C/C++ and Java have been the most commonly used languages to develop parallel metaheuristics implementations (Parejo et al., 2012). Some other languages and tools have been sporadically applied in specific research contexts, such as Matlab in the engineering community.

Parallel metaheuristics can be developed by designing ad-hoc implementations using low-level interprocess communication mechanisms, such as TCP/IP "sockets" or "pthreads," to perform the communication, synchronization, and cooperation between the parallel processes. This approach was frequently used in the pioneering implementations of parallel metaheuristics developed around 15 years ago, but is rarely used in practice today. At a higher level of abstraction, libraries for developing parallel computing applications have been employed in ad-hoc implementations of parallel metaheuristics. The most commonly used libraries include the implementations of the MPI standard (Gropp et al., 1994, MPICH, LAM/MPI) and the novel MPI-2 standard (Gropp et al., 1999, MPICH2, OpenMPI, LAM/MPI) for distributed memory platforms, and OpenMP (Chapman et al., 2007) for shared memory computers. When developing parallel metaheuristics in Java, the most commonly used method to implement the communication and synchronization is the built in remote method invocation (RMI, Grosso, 2001). Using RMI, a Java program is able to export an object that will be accessible across the network using a TCP port, where other processes can connect and call the methods that the distributed object provides.

Ad-hoc implementations of parallel metaheuristics are often closely tied to the particular problems solved. Many of these works essentially start the implementation from scratch, making it difficult to reuse the existing code, and to compare alternative methods. As a consequence, the approach of developing ad-hoc implementations of parallel metaheuristics has been progressively less used in the last years, where the main trend is to take advantage of using generic frameworks.

5.2.2 Parallel metaheuristics frameworks

Over the years, the metaheuristics research community has proposed and implemented frameworks including parallel versions of many well-known techniques. Generic frameworks help developing new parallel metaheuristic variants quickly, experimenting with existing ones, tackling new applications, and quickly performing fair comparisons in a well-known and stable environment. Most of the proposed frameworks implement EA, which has been consistently the parallel metaheuristic of preference, mainly due to its versatility for solving problems with diverse characteristics in many application domains. Other methods, including multiobjective parallel metaheuristics, have also been implemented.

The most used generic frameworks for parallel metaheuristics have been conceived to follow the object-oriented paradigm for software development. This choice allows having the benefits of modularity, reusability, and flexibility when applying them to a wide range of optimization problems. The design of such frameworks usually also includes the conceptually needed separation between the algorithms and the specification of the problems, which will be provided by the user.

In generic frameworks for parallel metaheuristics, the parallel implementation issues are often encapsulated in a specific module or class, providing a user-friendly way to deal with parallelism. In this way, the use of TCP/IP sockets and parallel libraries such as MPI or Java RMI are wrapped by high-level classes that make it easy for the user to exploit the parallel environment. When using large clusters or grid infrastructures, a specific general-purpose middleware layer between the hardware and the software is required to handle the distributed resources and virtual organizations. Globus (Foster, 2005) and Condor (Thain et al., 2002) have been the preferred choices to enable the parallel metaheuristics executions in large clusters (Dorronsoro et al., 2007; Luque et al., 2009; Tantar et al., 2007), while Sun Grid Engine and BOINC have also been frequently used as middleware on volunteer computing and grid environments (Cole et al., 2010; Lombrana et al., 2009). Recently, some parallel metaheuristics implementations have been developed and executed using programming tools available in the novel cloud computing environments, such as Hadoop and Amazon Web Services (Jin et al., 2008; Llorà et al., 2010; Verma et al, 2010). Until now, about 20 generic frameworks for parallel metaheuristics, have been proposed, and they have shown different degrees of maturity. The most relevant framework are as follows:

- *MALLBA* (Alba et al., 2006, 2007c): A C++ library of algorithms for optimization that can deal with parallelism (using MPI) on LAN or WAN platforms, in a user-friendly and efficient manner. Generic templates of metaheuristics (EAs, SA, ACO, PSO, DE, ILS, cellular and distributed versions, hybrids, and others) are implemented as "software skeletons," to be instantiated with the features of the problem by the user. These templates incorporate all the knowledge related to the resolution method, its interactions with the problem, and the considerations about the parallel implementations.
- *ParadisEO* (Cahon et al., 2004): A C++ white-box object-oriented framework dedicated to the reusable design of metaheuristics, portable on Windows, Unix, and MacOS. ParadisEO provides support for parallel implementations in parallel/distributed architectures (using MPI) and grid computing systems (with the ParadisEO-CMW extension, using Globus, and Condor-G/MW). The metaheuristics implementations include EAs, PSO, TS, ILS, hybrids, and others.
- *pALS* (Bernal and Castro, 2010): A Java object-oriented framework with a high degree of flexibility for the development of parallel and cooperative metaheuristics. pALS provides two main models of parallelization: (i) the parallel execution of algorithms and/or specific operations inside a metaheuristic, and (ii) the execution of separate instances or multistart models. Also, cooperation strategies such as the distributed subpopulations model for EAs or the parallel exploration of neighborhoods in trajectory-based metaheuristics are provided. A specific execution module allows the execution in grid infrastructures using Globus and Condor.
- *ECJ* (Luke et al., 2007): A Java framework that implements parallel models for EAs, with a particular emphasis in genetic programming. ECJ provides implementations of the master–slave

and the distributed subpopulation models, and new additions include multiobjective optimization algorithms, implementations of PSO and DE, coevolution, steady-state and evolution strategies methods, and various encodings (e.g., rule-sets) widely used in the genetic programming community. The parallel models are implemented using TCP/IP sockets.
- *OPT4J* (Lukasiewycz, 2009): A Java-based framework that currently includes a single-objective SA with some predefined cooling schedules, and multiobjective EA, PSO, and DE methods. A multithread implementation is provided to support the parallel execution on multiprocessors.
- *DGPF* (Weise and Geihs, 2006): An adaptable framework for distributed multiobjective search algorithms applied to Genetic Programming, which includes master–slave, P2P, and hybrid models of parallel GA/GP.
- *PMF* (Garrett, 2010): A high-level multicore-enabled framework for the construction of metaheuristics for single and multiobjective optimization implemented in C++ and using Intel Threading Building Blocks library Garrett (2010). PMF includes single-objective metaheuristics (steady-state EAs, local search operators, SA, TS), and multiobjective metaheuristics (NSGA-II, SPEA2, and ε-MOEA). The model of parallelism is highly adapted for algorithms that split the work done during each generation across multiple cores, and it is especially focused on hybrid algorithms with time-consuming local search operators. Support for other parallel models is not provided.

Other libraries and frameworks that only include a few implementations of parallel metaheuristics are as follows:

JDeal (Gehlsen and Page, 2001): A Java-based framework for EAs that implements the parallel evaluation of solutions following the master–slave model.

Distributed BEAGLE (Gagne et al., 2003): It provides a high-level environment for implementing master–slave and distributed subpopulations parallel EAs using TCP/IP sockets.

DREAM (Arenas et al., 2002): A Java-based framework that implements the distributed subpopulation model for EAs, using a P2P paradigm with TCP/IP sockets.

Java Grid-enabled Genetic Algorithm (JG^{2}A) (Bernal et al., 2009): It implements two simple parallel models: the multiple parallel independent executions and the distributed master–slave for the fitness function evaluation.

EASEA (Maitre et al., 2009): A platform for EAs that implements the master–slave model for fitness evaluation in GPUs.

HeuristicLab (Wagner and Affenzeller, 2005): An extensible framework for metaheuristics (EAs, MOEAs, PSO, SA, TS, and other methods) written in C# using .NET, which only implements the parallel distributed subpopulation model for GA, but the execution in grid computing environments is also possible.

Several other less used frameworks have been proposed for parallel EAs and ACO, such as the framework for parallel GA following the System-on-a-Programmable-Chip concept (Salmani et al., 2006); the reusable framework for executing master–slave parallel ACO algorithms (Craus and Rudeanu, 2004); the pCMALib parallel library for evolution strategy in Fortran 90 (Mueller et al., 2009); and a recently proposed framework for distributed subpopulations EAs in grids using Globus (Limmer and Fey, 2010).

Undoubtedly, MALLBA and ParadisEO are the most complete and comprehensive frameworks for parallel metaheuristics. MALLBA has been continually evolving since its creation in 2002, and currently it provides sequential and parallel implementations of the most well-known metaheuristics (different flavors of EA, ACO, PSO, SA, ES, VNS, DE, CLS, hybrids). ParadiSeO also includes a wide range of parallel metaheuristics (EAs, PSO, TS, ILS, hybrids) as well as metaheuristics for multiobjective optimization. These two frameworks are the current state-of-the-art libraries for implementing parallel metaheuristics.

6. Methodology

Unlike exact methods, where time-efficiency is a main measure for evaluating success, there are two chief issues in evaluating parallel metaheuristics: how fast solutions can be obtained (efficiency), and how far they are from the optimum (efficacy). We can distinguish between two different approaches for analyzing metaheuristics: a theoretical analysis (worst-case analysis, black-box analysis, etc.) and an experimental analysis. Although several theoretical analyses have been developed for a number of heuristics and problems, these theoretical achievements have a difficulty in their utilization for real problems and algorithms, severely limiting their range of application. As a consequence, most of metaheuristics are evaluated "empirically" in an "ad-hoc" manner.

An experimental analysis usually consists in applying the developed algorithms to a collection of problem instances and comparatively report the observed solution quality and consumed computational resources (usually the time required to perform the search and/or the time needed to find the best solution). Other researchers (Rardin and Uzsoy, 2001) have tried to offer a kind of methodological framework to deal with the experimental evaluation of heuristics in general. Important aspects of an evaluation are the experimental design, finding good sources of test instances, measuring the algorithmic performance in a meaningful way, sound analysis, and clear presentation of results. Due to the great difficulty in making all this correctly, the main issues of experimental evaluation are simplified to only "highlight" some guidelines for designing experiments, and reporting on their results. An excellent algorithmic survey about simulations and statistical analysis is given in McGeoch (2007, 2008). In these papers, McGeoch includes an extensive set of basic references on statistical methods and a general guide for designing experiments.

Some best practices that we should take into account when researching with parallel metaheuristics are as follows:

- Distinguish between instances and problem class.
- Report parameters.
- Perform 30–100 independent runs (at least).
- Always report times and numerical performance.
- Analyze scalability with problem dimension.
- Compare against standard algorithms of the same class of the new proposed, the best so far, and a random search (this last is a very important sanity check).
- Use meaningful metrics.
- Perform statistical assessment of the numerical metrics.

In the rest of this section, we focus on how the experiments should be performed, and how the results must be reported in order to make fair comparisons between parallel metaheuristics.

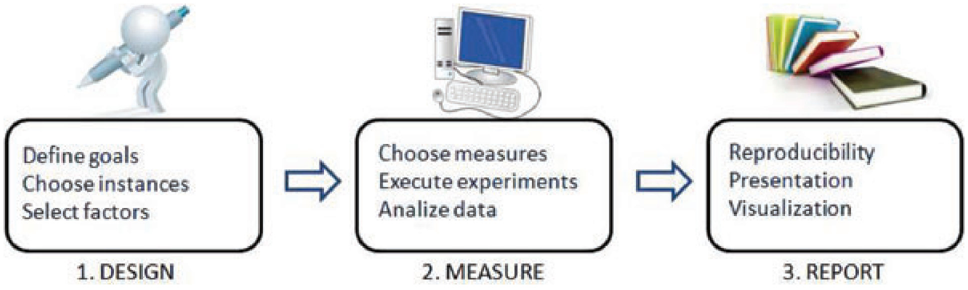

Fig. 4. Main steps for an experimental design.

Specially, our main interest is revising, proposing, and applying parallel performance metrics and statistical analysis guidelines to ensure that our conclusions are correct and reproducible by others. Finally, we have also added a subsection about recent theoretical developments in the parallel metaheuristic domain for those readers going this way.

6.1 Previous experimental design

In general, the goal of a scientific job is to present a new approach or algorithm that works better, in some sense, than existing algorithms. This requires experimental tests to compare the new algorithm with respect to the rest. It is in general hard to make fair comparisons between algorithms. The reason is that we need to ensure the same experimental setting: computer, network protocols, operating system, and all the other features. Assuming this is correctly done, we could even infer different conclusions from the same results depending on the metrics we use. This is specially important for nondeterministic methods. In this section we address the main issues on experimental testing for reporting numerical effort results, and the statistical analysis that "must" be performed to ensure that the conclusions are meaningful. The main steps are shown in Fig. 4.

The first choice that a researcher must make is the problem domain and the problem instances to test his/her algorithm. That decision depends on the goals of the experimentation. We can distinguish between two clearly different objectives: (1) optimization versus (2) understanding of the algorithms.

Optimizing (or searching or learning) is a commonly practiced sport in designing a metaheuristic that beats others on a given problem or set of problems. This kind of experimental research finishes by establishing the superiority of a given technique over others. In this scenario, researchers should not be limited to establishing "that" one metaheuristic is better than another in some way, but also to investigate "why," i.e., they must understand how the algorithms work. These last studies usually are developed using mathematical tools (run-time analysis, takeover time study or landscapes analysis). Although these mathematical methodologies are useful, in some cases they cannot be applied and then researchers should design experimental studies to analyze the behavior of the method.

For both goals, optimization and understanding the algorithm, one important decision is the instance to be used. The set of instances must be complex enough to obtain interesting results and must have a sufficient variety of scenarios to allow some generalization of the conclusions. Also, the number and dimension of instances is capital (size matters). Problem generators are specially good for a varied and wide analysis. In the next paragraphs we show the main classes of instances (a more comprehensive classification can be found in Eiben and Jelasity, 2002; Rardin and Uzsoy, 2001).

Real world instances. The instances taken from real applications represent a hard testbed for testing algorithms. Sadly, it is rarely possible to obtain more than a few real data for any computational experiment due to proprietary or economic considerations. An alternative is to use random variants of real instances, i.e., the structure of the problem class is preserved, but details are randomly changed to produce new instances (Eiben and Jelasity, 2002). This kind of instances represent instances that emerge from a specific real life situation, such as timetabling of a school. This class of instances has the advantage of being freely available. Specially, academic instances must be looked for and analyzed in the existing literature to not reinvent the wheel, and to avoid using straightforward benchmarks (Whitley, 2001).

Standard instances. In this class are included the instances, benchmarks, and problem instance generators that, due to their wide use in experimentation, became standard in the specialized literature. For example, Reinelt (1991) offers the "TSPLIB," a traveling salesman problem test instances, Demirkol et al. and Uzsoy et al. (1998) offer something similar for job scheduling problems. Such benchmarks allow to test specific issues of algorithms and also to compare our results against other methods. Other examples are CEC and GECCO instances for continuous global optimization (Finck et al., 2009; Suganthan et al., 2005; Tang et al., 2007a), and the OR-library Beasley (1990), as excellent examples of results (academy plus industry) for a large set of problem classes.

Random instances. Finally, when none of the mentioned sources provide an adequate supply for tests, the remaining alternative is pure random generation. This method is the fastest way to obtain a diverse group of test instances, but it is also the most controversial.

After having selected a problem or a group of instances, we must design the computational experiments. Generally, the design starts by analyzing the effects of several factors on the algorithm performance. These factors include problem factors, such as problem size, number of constrains, etc., plus algorithmic factors, such as parameters or components used for the search of the optimum. If the cost of the computer experiments are low, we can do a "full factorial" design, but in general it is not possible due to the large number of experiments: we usually need to reduce the set of studied factors. There is a wide literature on "fractional factorial" design in statistics, which seeks to assess the same effects of a fractional analysis without running all the combinations of influencing parameters (see e.g., Montgomery, 2000). RACE (Birattari et al., 2002) and SPO (Bartz-Beielstein and Preuss, 2009) are interesting approaches for validating and reducing the effort of the researcher in experimentation.

The next step in an experimental project is to execute the experiments, choose the measure of performance, and analyze the data. These steps are addressed in the next sections.

6.2 Metrics

The objective of a parallel metaheuristic is to find a "good" solution in a "reasonable" time. Therefore, the choice of performance measures for experiments necessarily involves both solution quality and computational effort. Measuring an algorithm properly is a crucial issue in order to perform fair comparisons and to obtain accurate conclusions. There is quite a large number of metrics for parallel metaheuristics (Alba and Luque, 2006a). Among the metrics to measure the performance of parallel metaheuristics, the most important and used one is the speedup, but in the scientific community a consent does not exist on its definition and use. If we denote with T_m the execution time for an algorithm using m processors, the speedup is the ratio between the (larger) execution time on one processor T_1 and the (smaller) execution time on m processors T_m is

$$s_m = \frac{E[T_1]}{E[T_m]}. \tag{1}$$

This metric compares wall-clock times. The main difficulty is that researchers could not agree on which execution time on a uniprocessor and which execution time for an algorithm using more than one processor should be used (Luque et al., 2009; Maitre et al., 2009; Vrajitoru, 2010). Actually, this situation is even more complex since new parallel platforms such as GPUs and multicores need to take into account other issues: throughput, cycle-per-instruction (-CPI-), memory access patterns for different core and thread combinations, core-scaling, and thread-scaling (Byun et al., 2009; Luong et al., 2010). Because of this, authors distinguish several definitions of speedup depending of the meaning of these values, creating different taxonomies.

From these taxonomies (Alba and Luque, 2006a; Barr and Hickman, 1993), it is clear that the evaluated parallel metaheuristics should compute solutions having a similar accuracy as the sequential ones. This accuracy could be the optimal fitness value (if known) or a relaxation of it (e.g., 90% of the optimal value), but in any case, the same value. Then, the stopping criterion of the sequential and parallel algorithms should be set to get a same fitness value. Just in this case we are allowed to compare resulting times. The used times are average mean times: the parallel code on one machine versus the parallel code on m machines (and not versus a sequential panmictic version, that in fact is a different algorithm). All this defines a sound way for comparisons, both practical (no best algorithm needed) and orthodox (same codes, same accuracy).

Recently, new definitions of speedup have been presented for specific platforms and parallel issues. Wang (2009) proposes the speedup metric taking into account checkpointing overhead, since the traditional speedup only measures the performance of failure-free systems. The new metric unifies performance and reliability measures, and evaluates the practical speedup of parallel application with checkpointing. Hoekstra and Sloot (2005) also present a new speedup definition for grid platforms. The concept of this metric is only to be used as a scalability metric for parallel applications on the grid.

Although the speedup is a widely used metric, there exist other measures of the performance of a parallel metaheuristic. The "efficiency" (Equation 2) is a normalization of the speedup ($e_m = 1$ means linear speedup, $e_m < 1$ means sublinear speedup, and $e_m > 1$ means superlinear speedup).

In some applications, stopping after a maximum time is mandatory, for example when using ACO (not having the concept of point visited in the landscape since they are mainly devoted to construct solutions instead of visiting the neighborhood of a given solution) or when stress is focused on comparing efficacy of two very different algorithms or implementations. In any case, real time is very important in parallelism, and should always be reported:

$$e_m = \frac{s_m}{m}. \tag{2}$$

Finally, Karp and Flatt (1990) have devised an interesting metric for measuring the performance of any parallel algorithm, that can help us to identify much more subtle effects than using speedup alone. They call it the "serial fraction" of the algorithm (Equation 3),

$$f_m = \frac{1/s_m - 1/m}{1 - 1/m}. \tag{3}$$

Ideally, the serial fraction should stay constant for an algorithm. If a speedup value is small we can still say that the result is good if f_m remains constant for different values of m, since the loss of efficiency is due to the limited parallelism of the program. On the other side, smooth increments of f_m are a warning that the granularity of the parallel tasks is too fine. A third scenario is possible in which a significant reduction in f_m occurs as m enlarges, indicating something akin to superlinear speedup. If this occurs, then f_m would take a negative value.

6.3 Statistical assessment of results

Once we have selected the appropriate metric to compare the algorithms, we need to perform a statistical analysis to ensure the correction of the conclusions since in most papers, the objective is to prove that a particular metaheuristic outperforms another one according to a concrete metric. We have discussed about the metrics, but as we said before, the comparison between two average values might be different from the comparison between two distributions, which is what we actually have when the algorithm is run 100 times.

Although it is quite usual to find papers without statistical analysis even in recent journal articles, statistical methods should be employed wherever possible to indicate the strength of the relations between different factors and performance measures. A general framework to perform the statistical analysis is to follow the next main steps (see Fig. 5): first, a normality test (e.g., Kolmogorov–Smirnov) should be performed in order to check whether the variables follow a normal distribution or not. If so, a Student t-test (two sets of data) or analysis of variance (ANOVA) test (two or more sets of data) should be done. If they are not normally distributed, we should perform a nonparametric test such as Kruskal–Wallis, Wilcoxon, or Friedman. This two-step procedure also allows to control the type I error (the probability of incorrectly rejecting the null hypothesis when it is true), since the two phases are independent (they test for different null hypotheses).

Usually, researchers use t-test or an ANOVA to ensure the "statistical significance" of the results, i.e., determining whether an observed effect is likely to be due to sampling errors (Janson et al., 2006; Nebro et al., 2006; Urlings et al., 2008). The two analyses, t-test and ANOVA, can only be applied if the source distribution is normal. In metaheuristics, the resulting distribution could easily be nonnormal (in fact, times are never normal formally, since the tail of distribution is cut in zero). For this case, there is a theorem that helps. The "central limit theorem" states that the sum of many identically distributed random variable tends to a Gaussian. So the mean of any set

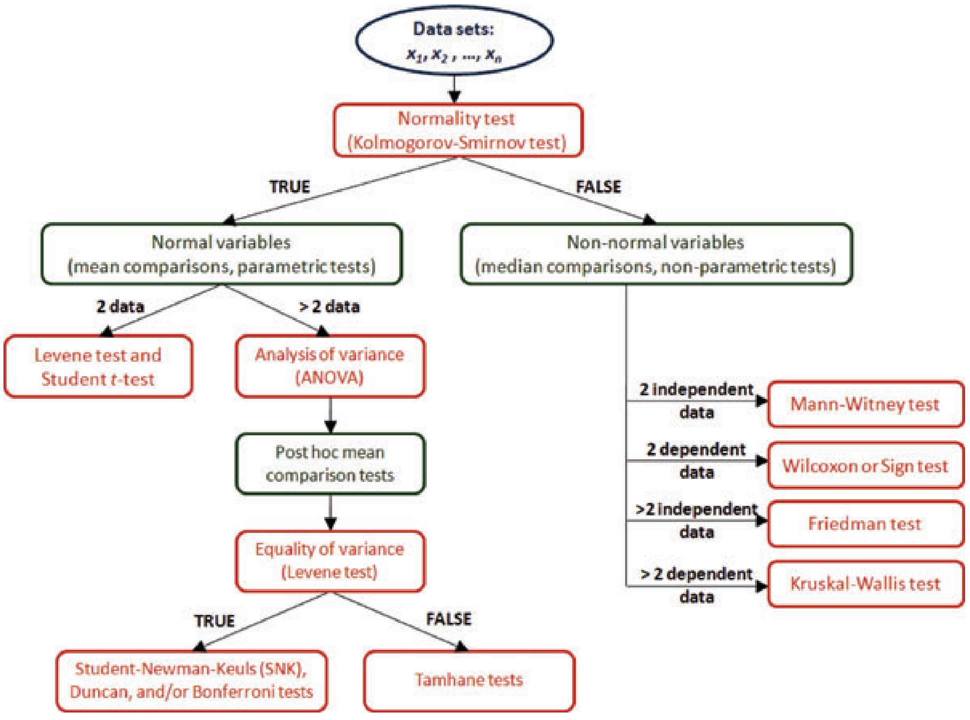
Data sets: $x_1, x_2, ..., x_n$
Normality test (Kolmogorov-Smirnov test)
TRUE
FALSE
Normal variables (mean comparisons, parametric tests)
Non-normal variables (median comparisons, non-parametric tests)
2 data
> 2 data
Levene test and Student t-test
Analysis of variance (ANOVA)
Post hoc mean comparison tests
Equality of variance (Levene test)
TRUE
FALSE
Student-Newman-Keuls (SNK), Duncan, and/or Bonferroni tests
Tamhane tests
2 independent data
Mann-Witney test
2 dependent data
Wilcoxon or Sign test
>2 independent data
Friedman test
> 2 dependent data
Kruskal-Wallis test

Fig. 5. Statistical assessment of results in metaheuristics.

of samples tends to a normal distribution. But in several cases the central limit theorem is not useful at all. In these cases, there are a host of nonparametric techniques that can and should be employed to sustain author's arguments, even if the results show no statistical difference between the quality of the solutions produced by the metaheuristics under study. One of the most commonly used nonparametric test is Kruskal–Wallis (Alba et al., 2007d; Pedemonte and Cancela, 2010).

An important observation is that, in order to report the results, we should show the mean value when the data follow a normal distribution, or the median otherwise. In parallel metaheuristics, authors need to report whether they are using the best values of averages, average of averages, or whatever other metric out of the parallel components running to solve the problem.

6.4 *Theoretical developments*

Although parallel metaheuristics have been known and used for decades, the number of theoretical studies analyzing their behavior is very limited. Just recently this situation is changing, and it exists a growing interest in theoretical issues about parallel metaheuristics. These new works follow three different approaches: experimental studies, theoretical analysis of the takeover time, and runtime analysis.

The first approach consists in performing a large number of general experiments, and from their results, try to obtain some global conclusions or guides to help other researchers. In this category, we can found the works (Alba and Luque, 2006b; Alba et al., 2004) where the authors analyze the influence of several parameters that control the parallel execution of the method, on different parallel platforms. In those works, authors concluded that platforms with slow communication mechanism (e.g., WAN environments) can be beneficial for some problems due to the higher isolation times among the subalgorithm components provoked by these slower communications. Other works study the impact of the parametrization of parallel models to characterize specific issues such as communication topology (Hijaze and Corne, 2009) and communication period (Skolicki and DeJong, 2005).

Another theoretical line of research is the exact (or approximate) computation of takeover times where the goal is to estimate how quickly information is spread throughout the system. This line has been explored for different parallel models such as cellular algorithms (Giacobini et al., 2005a, 2005b; Payne and Eppstein, 2007) or distributed ones (Alba and Luque, 2005; Luque and Alba, 2010; Rudolph, 2006). An interesting model is a closed formula for the takeover time of distributed algorithms proposed by Alba and Luque (2005):

$$t^* = per \cdot d(T) - \frac{1}{b} \cdot Ln\left(\frac{1}{a} \cdot \frac{\varepsilon}{N - d(T) - \varepsilon \cdot N}\right). \tag{4}$$

In this equation authors relate takeover to the most important parameters in the distributed model (migration period, *per* and the diameter of the topology $d(T)$). We must notice that this and other similar models could be used as pure mathematical descriptors of the behavior or even put to work actively inside self-adaptive new algorithms (interesting research line).

Finally, there are some recent studies considering the runtime analysis in parallel metaheuristics, in particular in distributed models. Lässig and Sudholt (2010b) present a theoretical analysis showing the essential role of the migration in some problems. The same authors proposed a general method for estimating the parallel running time of the island model Lässig and Sudholt (2010a) and finally, they also presented a study about the influence of the number of subalgorithms in a parallel metaheuristic (Lässig and Sudholt, 2011).

7. Challenges and new trends

In this section, we list some of the hot topics that will probably be main lines of research in the upcoming years. For their discussion, we distinguish four domains of development: "technology," "algorithms," "methodology," and "challenges." As to the technology, it is clear that in any parallel application there is a stress on the underlying network and parallel hardware. The reason is that the ultimate source for big improvements in time speedup is linked to the hardware of the used platform. Also, software technology plays an important role, since software tools are in charge of approaching the hardware technology to users; how much the software simplifies the utilization of the parallel hardware is also a goal for research, not only for mere utilization.

In addition to technology, a second big domain of research is represented by algorithms. First, algorithms are our main object of research when solving a complex problem, and making them parallel is by itself a difficult task. In dealing with algorithms, not only porting them to new technological platforms is important, but also to fit them to the special features of the new parallel architectures.

The third domain of discussion here will be methodology. Since researchers in parallel algorithms are either coming from the parallel part or from the algorithm part, it is difficult to have a common nomenclature and way of thinking. Since most PhD students are pressed to finish their theses, spending time in reading and knowing previous works is always limited in practice. This means that the needed effort to work at the same time in parallelism and algorithms, plus the needed time to get acquired of the literature preclude in many cases a research in the methodological application and experimentation of this field. However, building a "body of knowledge" is very important, and research will be welcomed if it works with benchmarks and best practices in the evaluation of parallel metaheuristics, as well as in their underlying theory.

Finally, a fourth domain can be identified: a set of abstract scientific challenges that will guide the three previous fields towards successful studies and applications. There are plenty of opportunities for research in the first three mentioned fields, but here in the fourth item we will highlight some of the most potentially beneficial ones for the community.

After this introduction, let us go for the details in the next sections.

7.1 Technology

We start this subsection considering the main hardware issues available for research

- *Multicore:* With the outstanding growth of multicore machines in both desktop and notebooks, unveiling the behavior of parallel metaheuristics when running in multicores is a must. Studies here are wanted in the utilization of the shared memory for communication, in the differences between multicores and clusters of machines, in the advantages of using hyperthreading on many core systems, and in developing algorithms suited to this architecture in order to extract all the power out of it for parallel metaheuristics.
- *GPUs:* General Processing with GPUs (GPGPU) is really a raising field. In addition to porting parallel algorithms to GPUs, it is very important to create algorithms that will profit from the thousands of cores/threads of GPUs in a proper way (Maitre et al., 2009). Present research exists on this, but we feel that researchers are sometimes happy when just getting modest 2×, 3×, 10× speedups, while a GPU could deliver much more higher values if the algorithms were suited to their architecture.
- *APUs:* New advances in the combination of GPU and CPU are gaining momentum in the industry; these new processors will open research to deal with asynchronous algorithms profiting from the two kinds of components. It will then be possible to run collaborative parallel metaheuristics both on GPU and CPU, and even connect them in a single computer. This powerful system-on-a-chip will also allow to create new unseen search models for complex problems, and can be one way to arrive to the computation of solutions in just a few seconds for problems that take days at present.
- *Smartphones:* The more our society develops, the larger is the impact of portable smartphones (tablets and handhelds, also). If researchers are concerned about the final utilization of their parallel metaheuristics (a reasonable concern, we must say) then a moment will come when we need to run algorithms on these platforms. There, the computational power is low compared to desktops but they represent the actual way to give final services to users: computing vehicle routes,

advising medicine to doctors, learning problems in changing environments, etc. Here, multicore systems are starting to appear (dual core at 1 GHz exists, now for example), and probably will be scaled up in the near future (dual core at 2 GHz planned along 2011 by Google). Also, new ways of computing for parallel metaheuristics rendering services to final users will appear, based on P2P or software models of business that need the intelligence of a parallel metaheuristic.
- *Cloud computing:* This model of exploitation for services can be used to harness the difficulty of running parallel metaheuristics in big problems. It is true that this field is probably nearer to final users of parallel metaheuristics than to experts in the field, because of the (undesired) loss of control when using cloud computing. However, the new services of Microsoft, Google, and Amazon are great tools for new models of computation in parallel.

After discussing on the hardware, we now provide some aspects related to new software features and potential developments

- *Language embedded:* For developers, languages having embedded communication methods through a network is an easy and standard way of programming new parallel metaheuristics. In this sense, Java (with RMI) is having an interest for the near future. It is not a slow language as many researchers think, although it is true that C and even C++ can still run clearly faster in many platforms. The portability and good efficiency makes it a very complete language for this field. Other languages exist that have been used (Perl or Fortran) (Douguet et al., 2000; MacCallum, 2003) but surely they are not as well spread in the parallel metaheuristic domain as Java and C/C++.
- *Communication standards:* Here, the Socket interface is the lower level way to ensure portability at a negligible overhead. The standard has existed and will exist for a very long time because of its close relation to Internet and TCP/IP, making it attractive for researchers looking for the most efficient way to communicate processes. The weak part is their really low level of programming, and that is the reason for the high acceptation of MPI. MPI is a standard (not a software or an implementation) that is widely supported in most parallel architectures, with a reasonably high level of programming primitives and a very good efficiency at the same time. The utilization of MPI will probably still give birth to new parallel metaheuristics in the forthcoming years, with nice implementations such as OpenMPI, MPICH (http://www.mcs.anl.gov/research/projects/mpich2/) and LAM MPI (http://www.lam-mpi.org/) standing on the top.
- *Optimization utilities:* We will probably see also an increasing interest in including parallel metaheuristics in well-known numerical packages such as Matlab, Excel, or R. Extending the existing algorithms in these packages will attract more users to perform research with parallel metaheuristics. It is however clear that the efficiency and control that one can keep when using these packages is not comparable to developing in a programming language (complex data structures, memory control, connectivity to others), and then most probably (again) this line will be exploited by users of parallel metaheuristic, and not by insiders.
- *Libraries:* Many libraries exist in this domain such as MALLBA (Alba et al., 2007d), ParadisEO (Cahon et al., 2004), or HeuristicLab (Wagner and Affenzeller, 2005). Their use will probably keep growing by users and developers of new parallel algorithms in the future. Research here is to be done carefully, since they are tools, not actually contributions in science, and thus many authors will still prefer to make their own software for testing new ideas. As to practitioners,

libraries are great tools for them, and research will probably come from their utilization in new applications where the focus is on the result and not on the algorithm used.

- *GPU programming:* Developments in CUDA (NVIDIA, 2007) and in OpenCL (Munshi et al., 2009) will subsist for a time. A probable merging of the two could happen in the future from the point of view of programming paradigms. Here, the useful research can come from their utilization to fully exploit the potential of GPUs in creating new techniques that cannot be run in CPUs. Only when this field is full of algorithms not easily portable and with high efficiency, most researchers will understand the potential of GPUs and devote their effort to create lines of research in this domain.
- *Others:* New concepts, specially in the sense of applications will be valuable for future research. Some of them are Software As A Service (SaaS), web services for optimization such as ROS (Alba and García-Nieto, 2005), remote pay sites for applications, etc.

7.2 Algorithms

As we said before, algorithms are our main object of research when solving a complex problem, and many topics should be considered when we are using parallel methods:

- *Heterogeneous:* New algorithms running on the very common network of heterogeneous clusters will be valuable in the future. Many labs and departments are having a vast network of different processors that can be used to develop new algorithms and to deploy on them tools such as Condor for their better exploitation (Nebro et al., 2008b). In addition to heterogeneous hardware, also parallel heterogeneous software algorithms are sought that combine existing ideas (representations, different parameters, different operators) in one single solution proposal to deal with harder problems. This line of research needs a careful definition of what speedup means, and how to fairly compare techniques in order to advance in the body of knowledge on parallel metaheuristic. It seems that dealing with heterogeneity is a must today, so let us do it in the algorithms (Hung and Chen, 2010) and in actual hardware (Wilson and Banzhaf, 2009).
- *Asynchronous:* For most new hardware technologies for parallelism, asynchronicity is an important issue. Researching in asynchronous exchanges of information or operations in general will give birth to techniques that will level up the power of existing parallel metaheuristics Lucka and Piecka (2009). Flexibility for combining ideas in one single algorithm and improved efficiency versus synchronous versions are strong pros for going this way in the future (Luque et al., 2010).
- *P2P:* Further development in P2P algorithms is of interest. The parallel execution on mobile systems (handhelds, vehicles such as cars (http://diricom.lcc.uma.es; http://roadme.lcc.uma.es, etc.) needs such approaches to compute solutions in parallel or to collaborate in one common task. The important key here is how to do this and how to make it in an efficient manner that is competitive against existing techniques.
- *Self-adaptation:* It is clear that most decisions done by researchers are directed by previous (too often, personal) experience or by a preliminary set of experiments to tune algorithms. Parallel metaheuristics represent an interesting domain where self-adaptive algorithms can be proposed. Here, the key question is what is the overhead of this adaptation (must be reported, must be linear-like!) and whether adaptive algorithms are really competitive to the best state-of-the-art

techniques. If so, they will be the way to arrive to the desired exploration/exploitation balance that all researchers are looking for (even without knowing it).

- *Theory driven:* Theory has been greatly dismissed in most articles and research lines today. This is an error: the more we know on the parallel algorithm and the problem landscape the better we can build new operators and collaborative algorithms. Being able to characterize, the takeover time Giacobini et al. (2005a), the (apparently different) influence of migration topologies and policies (Payne and Eppstein, 2007), the expected effects of the parallel metaheuristics on a given problem, etc. all represent hot topics that will mean highly cited papers in the future for other researchers. This will definitely help in creating a body of knowledge in parallel metaheuristics.
- *Architecture specific:* Creating algorithms that profit from the hardware architecture is a must in the future on this field. Speedup and gains come ultimately from hardware, and fitting the algorithm to it is a way to go when efficiency is the goal. In the case of GPUs, an interesting example is that of systolic algorithms, able of creating new metaphors of resolution for complex problems not only in GPUs but also for traditional CPUs and the coming APUs (Alba and Vidal, 2011). Also, creating compilers to translate existing parallel metaheuristics to new architectures is a very interesting line of research, to be able of directly using them with a low effort (Maitre et al., 2009).
- *Traditional parallel metaheuristics:* There are still many open lines dealing with creating parallel versions of well-known metaheuristics. One of them is the development of efficient and accurate parallel ACO algorithms, since the literature tells us that such an algorithm is still to appear. Also, trajectory-based algorithms are not that commonly found in parallel studies, and remain a field to discover in a sense (although much exists on some of them, such as SA).
- *Multiobjective:* Developing efficient parallel MOEAs is a true challenge (Garrett, 2010; Liefooghe, 2010; Nebro and Durillo, 2010; Sasaki et al., 2006). Existing field knowledge is a clear factor to develop new and efficient MOEA models, but not enough by itself, because of the required distributed utilization of archives with nondominated solutions, and because of the need to define a convenient exchange of solutions between the components of a parallel multiobjective algorithm. Extensions of existing sequential algorithms and new techniques are of interest to clearly make a step forward in their advantages, since it is very often that researchers can only provide parallel results that are not clearly (statistically) superior to existing sequential algorithm, a trend that we all should change with fresh ideas and rigorous experimentation.

7.3 Methodology

We already introduced the details on what methodological issues exist in parallel metaheuristics. We now summarize them from the point of view of their future in the field.

- *Theory:* Developments in theory on parallel metaheuristics is a must. As traditional and sequential algorithms are better and better characterized, the time will come for parallel models to extend the ideas and to create new concepts explaining their dynamic behavior and their convergence (Lässig and Sudholt, 2010b). Here, the connection to landscape analysis is also fruitful, in that elementary landscape analysis (for example) can guide in developing effective/efficient operators for parallel algorithms (Whitley et al., 2010).

- *Statistics:* This is an extremely important issue. More than an interesting line of research, performing statistical analysis on the results and the behavior of parallel algorithms will become a necessary condition to publish in international conferences and journals. More than 30 runs, checking normality on the values, using statistical multicompare tests, etc. will become (is almost, already) a fundamental piece in the evaluation of parallel stochastic algorithms (Alba and Luque, 2006a; Bartz-Beielstein and Preuss, 2009).
- *Benchmarks:* There is a need of creating benchmarks easing the comparison of parallel algorithms. Other domains are very developed in this sense (such as multiobjective research, continuous optimization, and traditional combinatorial problems (Finck et al., 2009; Tang et al., 2007a), but parallel techniques still wait for a set of problems that can uncover the advantages of the proposed techniques. With benchmarks, comparisons and knowledge will start to be objective and widely spread in the community.
- *Metrics:* A careful definition of metrics for existing and new parallel metaheuristics is of capital importance (Alba and Luque, 2006a). Ensuring that speedup is a meaningful metric is in the core of many studies of this field, and limits the conclusions obtained when done in an unorthodox manner. New metrics for summarizing extensive executions are needed also, and clear quantitative goals are needed in the future articles on parallel metaheuristics (Hoekstra and Sloot, 2005).

7.4 Challenges

Finally, we conclude this part of our work by showing some interesting challenges in the parallel metaheuristics field. These challenges represent important open research lines in this domain.

- *Few seconds execution:* One urban legend says that metaheuristics, especially population-based models, are slow solvers. This is true in some complex problems with large dimensionality and of high restriction. However, there exists a set of problems amenable for fast resolution when the appropriate technique is devised. A challenge for the future will be to run algorithms in a few minutes or seconds, solving interesting optimization and search tasks. Thus, the evaluation of algorithms under wall-clock restrictions of a few seconds are of interest as future research lines. Here, technology plays an important role also, since platforms such as GPUs and other highly efficient processors could be one of the key to arrive at this real-time response (Vidal and Alba, 2010a, 2010b).
- *Beating the state-of-the-art:* Most researches have the goal of either to solve problems that have not been previously solved or to solve existing problems at a higher efficiency. In this last case, determining the best existing techniques in sequential and developing new parallel ones to beat them is an interesting line of research. Often, many researchers do not want to enter parallelism because they do not see a clear advantage (and do see lots of skills required to succeed in the parallel field); only when researchers in parallel metaheuristics establish clear comparisons to other existing best results and outperform them, the rest of communities will become interested in how this is possible, and then come to use parallel algorithms. In any case, comparing to the best techniques in every single research paper is a must to enhance the knowledge in any field, and metaheuristics in general are not an exception. This can be considered a component of these new best practices, along with including statistical analyses and make the paper

self-contained for future reproduction (Bartz-Beielstein and Preuss, 2009). A final interesting issue: is the proposed algorithm better than a blind random search? It is difficult to find articles including this straightforward comparison, a must in any modern study.

- *Scalability:* Parallel metaheuristics have the potential to scale to larger problem dimensions and number of restrictions. An interesting challenge consists in applying "stress tests" to any given new approach, in order to characterize when and how much they are useful (Hoekstra and Sloot, 2005). Problem generators are a kind of software that eases such scalability study, thus showing the potential of the proposed technique as a solver for the problem class, and not only for one problem instance.
- *Robustness:* There is a general shift in research in many domains from pure optimization to combined robustness analysis. In general, this means that researchers should be considering not only the average accuracy, but also the dispersion in the results. The sensibility of the worked solutions and the proper management of noise or dynamic conditions can be mastered specially well when the resolution technique is having specialized components that cover the search space in a better form (Finck et al., 2009). Thus, the interest of this topic (robustness) in the parallel metaheuristic domain is very high, in fact a topic that is underexploited at this moment (Talbi and Bachelet, 2006).
- *Multiobjective:* The impact of a multiobjective modeling of a problem is huge. The kind of insights that a researcher can gain by approaching this domain is vast and rich. A big challenge in parallel metaheuristics is how to actually share the nondominated solutions among the components of the parallel algorithm, and how to build and evolve a single Pareto front in a decentralized manner. Far from being evident, this is an open topic that can bring multiobjective researchers with powerful tools once this problem is first solved (Nebro and Durillo (2010) and Garrett (2010) are examples in this line).
- *Interactive/online:* When the goal of a research is to build an application, it is often the case that human users want to interact with the system, check different scenarios, see the effects of some changes on the results and include strategic enterprise information on it. Whatever the goal is, the ability of the parallel method to run fast and present multiple solutions (possibly several niche-oriented solutions) is a key factor that could be exploited in future research with parallel methods.
- *Body of knowledge:* The strategic goal of all the previous challenges is to advance on the theory and the applications of parallel metaheuristics. By doing so in a convincing manner, in a general fashion, and with a basic set of best practices and sanity checks, researchers will be able to delimit the domain of knowledge of parallel metaheuristics. Such a domain is often misunderstood, leading to works of low quality that are not even using the appropriate nomenclature, given that they are dealing at the same time with parallelism and metaheuristics. That body is growing (Luque and Alba, 2011) and getting defined on time, and we hope that this last section helps in stretching its bounds (Alba and Luque, 2006a).
- *Trade-off usability/efficiency in software:* An additional interesting topic for future research is to explicitly address the balance between software usability and efficiency when dealing with parallel metaheuristics. Many of packages and algorithms are being constantly proposed for parallel metaheuristics, but there is no way to evaluate their relative merits because researchers are not quantifying things such as "flexibility", "extensibility", "ease utilization" and such a list of vague terms, which most times are argued to justify a new library or tool. Defining the

goal of the software, and performing quantitative analysis is mandatory in this case. Researchers could analyze the average time to use a software component, overhead in time of the proposed architecture, whether a serious design exists or not (UML or similar), how the software scales with problem dimension, etc. This is hardly found at present, and a big effort in comparison and quantification is needed if this is to be considered a science.

8. Conclusions

This article contains a modern survey of parallel models and implementations of metaheuristics. We have stressed not only the associated algorithmic issues, but also the parallel tools for building parallel metaheuristics. By summarizing the parallel algorithms, their applications, classes, and theoretical foundations, we intend to offer information not only for beginners but also for researchers working with metaheuristics in general.

The up-to-date list of references has been elaborated to serve as a directory for granting the reader access to the valuable results that parallel metaheuristics are offering to the research community. Most important trends have been discussed, yielding what we hope is a unified overview and a useful text.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Innovation and FEDER under contract TIN2008-06491-C04-01 (the MSTAR project) and TIN2011-28194 (the RoadME project). It has also been partially funded by the Andalusian Government under contract P07-TIC-03044 (the DIRICOM project).

References

Alba, E., (ed.), 2005. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, New York.

Alba, E., Almeida, F., Blesa, M., Cotta, C., Diaz, M., Dorta, I., Gabarro, J., León, C., Luque, G., Petit, J., Rodriguez C., Rojas A., Xhafa F. 2006. Efficient parallel LAN/WAN algorithms for optimization. The MALLBA project. *Parallel Computing* 32, 5–6, 415–440.

Alba, E., Blum, C., Isasi, P., León, C., Gómez, J.A., 2009. *Optimization Techniques for Solving Complex Problems*. John Wiley & Sons, New York.

Alba, E., Chicano, F., 2005. On the behavior of parallel genetic algorithms for optimal placement of antennae in telecommunications. *International Journal of Foundations of Computer Science* 16, 2, 343–359.

Alba, E., Chicano, F., 2008. Observations in using parallel and sequential evolutionary algorithms for automatic software testing. *Computers & Operations Research* 35, 10, 3161–3183.

Alba, E., Dorronsoro, B., 2005. The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation* 9, 2, 126–142.

Alba, E., Dorronsoro, B., 2008. *Cellular Genetic Algorithms, Operations Research/Computer Science Interfaces*, Vol. 42. Springer-Verlag, Heidelberg.

Alba, E., García-Nieto, J.M., 2005. ROS (Remote Optimization Service). *Informe Técnico ITI* 05–08.

Alba, E., Leguizamón, G., Ordoñez, G., 2007a. Two models of parallel ACO algorithms for the minimum tardy task problem. *International Journal of High Perfomance Systems Architecture* 1, 1, 50–59.

Alba, E., Luna, F., Nebro, A.J., Troya, J.M., 2004. Parallel heterogeneous genetic algorithms for continuous optimization. *Parallel Computing* 30, 5–6, 699–719.

Alba, E., Luque, G., 2005. Theoretical models of selection pressure for dEAs: topology influence. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation, 2005*, Vol. 1. IEEE Press, Washington, DC, pp. 214–221.

Alba, E., Luque, G., 2006a. Evaluation of parallel metaheuristics. In *PPSN-EMAA'06*, Reykjavik, Iceland, September 13, pp. 9–14.

Alba, E., Luque, G., 2006b. Performance of Distributed GAs on DNA Fragment Assembly. In Nedjah, N., Mourelle, L.M., Alba, E. (eds) *Parallel Evolutionary Computations,* Springer-Verlag, Heidelberg, pp. 97–116.

Alba, E., Luque, G., Araujo, L., 2006. Natural language tagging with genetic algorithms. *Information Processing Letters* 100, 5, 173–182.

Alba, E., Luque, G., Coello, C.A.C., Luna, E.H., 2007b. Comparative study of serial and parallel heuristics used to design combinational logic circuits. *Optimization Methods and Software* 22, 3, 485–509.

Alba, E., Luque, G., García-Nieto, J.M., Ordonez, G., Leguizamon, G., 2007c. MALLBA: a software library to design efficient optimisation algorithms. *International Journal of Innovative Computation and Application* 1, 74–85.

Alba, E., Luque, G., Luna, F., 2007d. Parallel metaheuristics for workforce planning. *Journal of Mathematical Modelling and Algorithms* 6, 3, 509–528.

Alba, E., Luque, G., Troya, J.M., 2004. Parallel LAN/WAN heuristics for optimization. *Parallel Computing* 30, 5–6, 611–628.

Alba, E., Saucedo, J.B., Luque, G., 2007e. A Study of Canonical GAs for NSOPs. In Doerner, K.F., Gendreau, M., Greistorfer, P., Gutjahr, W., Hartl, R.F., Reimann, M. (eds) *Metaheuristics: Progress in Complex Systems Optimization,* Springer, Heidelberg, pp. 245–260.

Alba, E., Tomassini, M., 2002. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 6, 5, 443–462.

Alba, E., Vidal, P., 2011. Systolic optimization on GPU platforms. In *Proceedings of the 13th International Conference on Computer Aided Systems Theory, EUROCAST 2011.*

Allaire, F., Tarbouchi, M., Labonté, G., Fusina, G., 2009. FPGA implementation of genetic algorithm for UAV real-time path planning. *The Journal of Intelligent and Robotic Systems* 54, 495–510.

Almeida-Luz, S., Rodríguez, M., Vega, M., Gómez-Pulido, J., Sánchez-Pérez, J., 2009. GRASP and grid computing to solve the location area problem. In *Proceedings of the 2009 World Congress on Nature and Biologically Inspired Computing*, pp. 164–169.

Araujo, L., Merelo, J., 2009. Multikulti algorithm: Using genotypic differences in adaptive distributed evolutionary algorithm migration policies. In *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE Press, Washington, DC, pp. 2858–2865.

Araujo, L., Merelo, J., Mora, A., Cotta, C., 2009. Genotypic differences and migration policies in an island model. In Rothlauf, F. (ed.) *GECCO*, ACM, New York, pp. 1331–1338.

Arenas, M., Collet, P., Eiben, A., Jelasity, M., Merelo, J., Paechter, B., Preuß, M., Schoenauer, M., 2002. A framework for distributed evolutionary algorithms. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, PPSN VII, Springer-Verlag, Berlin, pp. 665–675.

Asouti, V., Giannakoglou, K., 2009. Aerodynamic optimization using a parallel asynchronous evolutionary algorithm controlled by strongly interacting demes. *Engineering Optimization* 41, 3, 241–257.

Aydin, M., Sevkli, M., 2008. Sequential and parallel variable neighborhood search algorithms for job shop scheduling. *Studies in Computational Intelligence* 128, 125–144.

Babbar, M., Minsker, B., 2006. Groundwater remediation design using multiscale genetic algorithms. *Journal of Water Resources Planning and Management* 132, 5, 341–350.

Bai, H., OuYang, D., Li, X., He, L., Yu, H., 2009. MAX-MIN ant system on GPU with CUDA. In *Proceedings of the 2009 Fourth International Conference on Innovative Computing, Information and Control*, IEEE Press, Washington, DC, pp. 801–804.

Barr, R.S., Hickman, B.L., 1993. Reporting computational experiments with parallel algorithms: issues, measures, and experts' opinions. *ORSA Journal on Computing* 5, 1, 2–18.

Bartz-Beielstein, T., Preuss, M., 2009. The future of experimental research. In *GECCO '09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*, ACM, New York, pp. 3185–3226.

Beasley, J.E., 1990. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society* 41, 11, 1069–1072.

Bendjoudi, A., Guerdah, S., Mansoura, M., Melab, N., Talbi, E.-G., 2008. P2P B&B and GA for the Flow-Shop Scheduling Problem. In Xhafa, F., Abraham, A. (eds) *Metaheuristics for Scheduling in Distributed Computing Environments,* Studies in Computational Intelligence, Vol. 146, Springer, Berlin, pp. 301–322.

Bernal, A., Castro, H., 2010. pALS: an object-oriented framework for developing parallel cooperative metaheuristics. In *Proceedings of the 24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2019*, IEEE Press, Washington, DC, pp. 1–8.

Bernal, A., Ramirez, M.A., Castro, H., Walteros, J.L., Medaglia, A.L., 2009. Jg2a: A grid-enabled object-oriented framework for developing genetic algorithms. In *Proceedings of the Systems and Information Engineering Design Symposium, 2009. SIEDS'09*, IEEE Press, Washington, DC, pp. 67–72.

Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K., 2002. A racing algorithm for configuring metaheuristics. In *Proceedings of the GECCO*, Vol. 2, Morgan Kaufmann, San Mateo, CA, pp. 11–18.

Blackwell, T., Branke, J., 2004. Multi-swarm optimization in dynamic environments. In Proceedings of *Applications of Evolutionary Computing*, LNCS 3005, Springer, Berlin, pp. 489–500.

Blagojevic, F., Nikolopoulos, D., Stamatakis, A., Antonopoulos, C., 2007. Dynamic multigrain parallelization on the cell broadband engine. In *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ACM, New York, pp. 90–100.

Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys (CSUR)* 35, 3, 268–308.

Boisson, J., Jourdan, L., Talbi, E.-G., Horvath, D., 2008. Parallel multi-objective algorithms for the molecular docking problem. In *Proceedings of the 2008 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, IEEE Press, Washington, DC, pp. 187–194.

Bouamama, S., 2010. A new distributed particle swarm optimization algorithm for constraint reasoning. In *Proceedings of the 14th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems: Part II*, KES'10, Springer-Verlag, Berlin, pp. 312–321.

Bozejko, W., Pempera, J., Smutnicki, A., 2008. Multi-thread parallel metaheuristics for the flow shop problem. In *Proceedings of the International Conference on Artificial Intelligence and Soft Computing*, pp. 446–454.

Bozejko, W., Smutnicki, C., Uchroski, M., 2009. Parallel calculating of the goal function in metaheuristics using GPU. *Lecture Notes in Computer Science* 5544, 1, 1014–1023.

Bozejko, W., Uchrowski, M., Wodecki, M., 2010. Parallel hybrid metaheuristics for the flexible job shop problem. *Computers and Industrial Engineering* 59, 2, 323–333.

Bozejko, W., Wodecki, M., 2008. Parallel scatter search algorithm for the flow shop sequencing problem. In *Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics*, Springer-Verlag, Berlin, Heidelberg, pp. 180–188.

Branke, J., 2001. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Norwell, MA.

Buyya, R., 2009. Market-oriented cloud computing: vision, hype, and reality of delivering computing as the 5th utility. In *Proceedings of the Cluster Computing and the Grid, IEEE International Symposium on 0, 1.*

Byun, J., Datta, K., Ravindran, A., Mukherjee, A., Joshi, B., 2009. Performance analysis of coarse-grained parallel genetic algorithms on the multi-core Sun UltraSPARC T1. In *Conference Proceedings-IEEE SOUTHEASTCON*, pp. 301–306.

Cadenas, J.M., Garrido, M.C., Muñoz, E., 2009. Using machine learning in a cooperative hybrid parallel strategy of metaheuristics. *Information Science* 179, 3255–3267.

Cahon, S., Melab, N., Talbi, E.-G., 2004. ParadisEO: a framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics* 10, 357–380.

Cancino, W., Jourdan, L., Talbi, E.-G., Delbem, A., 2010. A parallel multi-objective evolutionary algorithm for phylogenetic inference. In Proceedings of 4th International Conference on Learning and Intelligent OptimizatioN (LION4), *Lecture Notes in Computer Science* 6073, pp. 196–199.

Cardenas, M., Melin, P., Cruz, L., 2010. Parallel genetic algorithms for architecture optimization of neural networks for pattern recognition. *Soft Computing for Recognition based on Biometrics* 312, 303–315.

Catalá, A., Jaen, J., Mocholí, J., 2007. Strategies for accelerating Ant Colony Optimization algorithms on graphical processing units. In *Proceeding of the IEEE Congress on Evolutionary Computation*, pp. 492–500.

Chapman, B., Jost, G., Pas, R., 2007. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. MIT Press, Cambridge, MA.

Chen, Y., Chen, L., Tu, L., 2006. Parallel Ant Colony algorithm for mining classification rules. In *Proceedings of the IEEE International Conference on Granular Computing*, IEEE Press, Washington, DC, pp. 85–90.

Chintalapati, J., Arvind, M., Priyanka, S., Mangala, N., Valadi, J., 2010. Parallel ant-miner (pam) on high performance clusters. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6466* LNCS, 270–277.

Chu, D., Zomaya, A.Y., 2006. Parallel Ant Colony Optimization for 3D protein structure prediction using the HP lattice model. In Nedjah, N., de Macedo, L., Alba, E. (eds.) *Parallel Evolutionary Computations, Studies in Computational Intelligence*, Vol. 22, Springer, Berlin, pp. 177–198.

Coello, C.A.C., Lamont, G.B., Van Veldhuizen, D.A., 2007. *Evolutionary Algorithms for Solving Multi-objective Problems*. Springer-Verlag, New York.

Cole, N., Desell, T., Lombrana Gonzalez, D., Fernandez, F., Magdon-Ismail, M., Newberg, H., Szymanski, B., Varela, C., 2010. Evolutionary algorithms on volunteer computing platforms: the MilkyWay@Home project. In Fernandez, F., Cantu-Paz, E. (eds.) *Parallel and Computational Intelligence*. Springer-Verlag, Berlin, Heidelberg, pp. 64–90.

Crainic, T.G., Crisan, G.C., Gendreau, M., Lahrichi, N., Rei, W., 2009. A concurrent evolutionary approach for rich combinatorial optimization. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. ACM, New York, pp. 2017–2022.

Crainic, T.G., Toulouse, M., 2002. Parallel strategies for meta-heuristics. In Glover, F., Cochenberger, G. (eds.) *State-of-the-Art Handbook in Metaheuristics*, Kluwer Academic Publishers, Norwell, MA.

Craus, M., Rudeanu, L., 2004. Multi-level parallel framework. *International Journal of Computing* 3, 3.

Da Silva, A., Ochi, L., 2009. New sequential and parallel algorithm for dynamic resource constrained project scheduling problem. *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IEEE Computer Society, Washington, DC, pp. 1–7.

De Jong, K.B., Potter, M.A., Spears, W.M., 1997. Using problem generators to explore the effects of epistasis. In *Proceedings of the 7th ICGA*. Morgan Kaufmann, Burlington, MA, pp. 338–345.

De Jong, K.A., Spears, W.M., Gordon, D.F., 1993. Using genetic algorithms for concept learning. *Machine Learning* 13, 2, 161–188.

Deb, K., 2001. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, New York.

Delisle, P., Gravel, M., Krajecki, M., Gagné, C., Price, W., 2005a. Comparing parallelization of an ACO: message passing vs. shared memory. In *2nd International Workshop on Hybrid Metaheuristics, Lecture Notes in Computer Science*, Vol. 3636. Springer, Berlin, pp. 1–11.

Delisle, P., Gravel, M., Krajecki, M., Gagné, C., Price, W., 2005b. A shared memory parallel implementation of Ant Colony Optimization. In *Proceedings of the 6th Metaheuristics International Conference*, University of Vienna, Vienna, Austria, pp. 257–264.

Delisle, P., Krajecki, M., Gravel, M., 2009. Multi-colony parallel ant colony optimization on smp and multi-core computers. In *2009 World Congress on Nature and Biologically Inspired Computing*, pp. 318–323.

Devices, Advanced M., 2010. *The AMD Fusion Family of APUs*. Available at http://sites.amd.com/us/fusion/APU/Pages/fusion.aspx.

Doerner, K., Hartl, R., Benkner, S., Lucka, M., 2006. Parallel cooperative savings based ant colony optimization-multiple search and decomposition approaches. *Parallel Processing Letters* 16, 3, 351–369.

Doerner, K., Hartl, R., Lucka, M., 2005. A parallel version of the d-ant algorithm for the vehicle routing problem. In *Proceedings of International Workshop on Parallel Numerics*, Ljubljana, Slovenia, pp. 20–23.

Dorronsoro, B., Arias, D., Luna, F., Nebro, A., Alba, E., 2007. A grid-based hybrid cellular genetic algorithm for very large scale instances of the CVRP. In Smari, W. (ed.) *2007 High Performance Computing & Simulation Conference (HPCS 2007)*, pp. 759–765.

Dos Santos, J., De Lima Junior, F., Magalhaes, R., De Melo, J., Neto, A., 2009. A parallel hybrid implementation using genetic algorithm, GRASP and reinforcement learning. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 2798–2803.

Douguet, D., Thoreau, E., Grassy, G., 2000. A genetic algorithm for the automated generation of small organic molecules: drug design using an evolutionary algorithm. *Journal of Computer-aided Molecular Design* 14, 5, 449–466.

Du, W., Li, B., 2008. Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences* 178, 15, 3096–3109.

Durillo, J.J., Nebro, A.J., Luna, F., Alba, E., 2008. A study of master-slave approaches to parallelize NSGA-II. In *Proceeding of the IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008*. IEEE Press, Washington, DC, pp. 1–8.

Eiben, A.E., Jelasity, M., 2002. A critical note on experimental research methodology in EC. In *Proceeding of the Congress on Evolutionary Computation 2002*. IEEE Press, Washington, DC, pp. 582–587.

Ellabib, I., Calamai, P., Basir, O., 2007. Exchange strategies for multiple Ant Colony System. *Information Sciences*, 177, 5, 1248–1264.

Fang, Y., Wu, L., 2010. Parallel ant colony algorithm for the logistics scheduling problem. In *Proceedings - 2010 International Conference on Multimedia Communications, Mediacom 2010*, pp. 116–119.

Farmahini-Farahani, A., Vakili, S., Fakhraie, S.M., Safari, S., Lucas, C., 2010. Parallel scalable hardware implementation of asynchronous discrete particle swarm optimization. *Engineering Applications of Artificial Intelligence* 23, 177–187.

Fernando, P., Sankaran, H., Katkoori, S., Keymeulen, D., Stoica, A., Salem, R., Ramesham, R., 2008. A customizable FPGA IP core implementation of a general purpose Genetic Algorithm engine. In *Proceedings of the IPDPS 2008*, pp. 1–8.

Ferreira, A., Pardalos, P.M., 1996. *Solving Combinatorial Optimization Problems in Parallel: Methods and Techniques*. Springer Verlag, Berlin.

Finck, S., Hansen, N., Ros, R., Auger, A., 2009. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical Report 2009/20, Research Center PPE.

Flynn, M., 1972. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers* C-21, 948–960.

Foster, I., 2005. Globus Toolkit Version 4: software for service-oriented systems. In Jin, H., Reed, D.A., Jiang, W. (eds.), *NPC, Lecture Notes in Computer Science*, Vol. 3779. Springer, Berlin, pp. 2–13.

Foster, I., Kesselman, C., 1998. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, Burlington, MA.

Fu, J., Lei, L., Zhou, G., 2010. A parallel ant colony optimization algorithm with GPU-acceleration based on all-in-roulette selection. In *Proceedings of the 3rd International Workshop on Advanced Computational Intelligence, IWACI 2010*, pp. 260–264.

Gagne, C., Parizeau, M., Dubreuil, M., 2003. Distributed beagle: an environment for parallel and distributed evolutionary computations. In *Proceedings of the 17th Annual International Symposium on High Performance Computing Systems and Applications*.

Gaifang, D., Xueliang, F., 2010. A hierarchical parallel algorithm of ant system and local search for TSPs. In *Proceedings of the 2nd International Conference on Information Science and Engineering, ICISE2010*, pp. 4834–4837.

Gallardo, J.E., Cotta, C., Fernandez, A.J., 2007. On the hybridization of memetic algorithms with branch-and-bound techniques. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 37, 1, 77–83.

Gao, D., Gong, G., Han, L., Li, N., 2010. Application of multi-core parallel ant colony optimization in target assignment problem. In *Proceedings of the ICCASM 2010-2010 International Conference on Computer Application and System Modeling*, Vol. 3, pp. V3514–V3518.

Gao, L., Dong, W., 2010 A parallel variable neighborhood search for the traveling salesman problem. In *ICAMS 2010 - Proceedings of 2010 IEEE International Conference on Advanced Management Science*, Vol. 3, pp. 150–152.

Garrett, D., 2010. PMF: a multicore-enabled framework for the construction of metaheuristics for single and multiobjective optimization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6239 LNCS(PART 2), 351–360.

Gehlsen, B., Page, B., 2001. A framework for distributed simulation optimization. In *Proceedings of the 33nd Conference on Winter Simulation*, IEEE Computer Society, Washington, DC, pp. 508–514.

Georgieva, A., Jordanov, I., 2008. Hybrid metaheuristics for global optimization: A comparative study. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5271* LNAI, 298–305.

Giacobini, M., Tomassini, M., Tettamanzi, A., 2005a. Takeover time curves in random and small-world structured populations. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, ACM, New York, pp. 1333–1340.

Giacobini, M., Tomassini, M., Tettamanzi, A., Alba, E., 2005b. Selection intensity in cellular evolutionary algorithms for regular lattices. *IEEE Transactions on Evolutionary Computation* 9, 5, 489–505.

Gomes, F.C., Meneses, C.N., Pardalos, P.M., Viana, G.V.R., 2008. A parallel multistart algorithm for the closest string problem. *Computers & Operations Research* 35, 11, 3636–3643.

Gropp, W., Lusk, E., Skjellum, A., 1994. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA.

Gropp, W., Lusk, E., Thakur, R., 1999. *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press, Cambridge, MA.

Grosso, W., 2001. *Java RMI* (1st edn.). O'Reilly & Associates, Inc., Sebastopol, CA.

Grouchy, P., Thangavelautham, J., D'Eleuterio, G., 2009. An island model for high-dimensional genomes using phylogenetic speciation and species barcoding. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, ACM, New York, pp. 1355–1362.

Guarracino, M.R., Cifarelli, C., Seref, O., Pardalos, P.M., 2006. A parallel classification method for genomic and proteomic problems. In *Prceedings of the International Conference on Advanced Information Networking and Application*. IEEE Computer Society, Washington, DC.

Gunes, O., Sima, A., 2010. Parallelization of an ant-based clustering approach. *Kybernetes* 39, 4, 656–677.

Guo, H., Lu, Q., Wu, J., Huang, X., Qian, P., 2009. Solving 2D HP protein folding problem by parallel ant colonies. In *Proceedings of the 2nd International Conference on BioMedical Engineering and Informatics*, pp. 1–5.

Hamdani, T., Alimi, A., Karray, F., 2006. Distributed genetic algorithm with bi-coded chromosomes and a new evaluation function for features selection. In *Proceedings of the IEEE Congress on Evolutionary Computation, 2006. CEC 2006*, IEEE Press, Washington, DC, pp. 581–588.

Harding, S., Banzhaf, W., 2007. Fast genetic programming on GPUs. *Genetic Programming* 4445, 3, 90–101.

Harding, S., Banzhaf, W., 2008. Genetic programming on GPUs for image processing. *International Journal of High Performance Systems Architecture* 1, 4, 231–240.

Harding, S., Banzhaf, W., 2009. Distributed genetic programming on GPUs using CUDA. In Risco-Martín, J.L., Garnica, O. (eds.) *WPABA'09: Proceedings of the Second International Workshop on Parallel Architectures and Bioinspired Algorithms*, pp. 1–10.

He, H., Sýkora, O., Salagean, A., Mäkinen, E., 2007. Parallelisation of genetic algorithms for the 2-page crossing number problem. *Journal of Parallel and Distributed Computing* 67, 2, 229–241.

Hereford, J., 2006. A distributed particle swarm optimization algorithm for swarm robotic applications. In Yen, G., Lucas, S., Fogel, G., Kendall, G., Salomon, R., Zhang, B., Coello, C., Runarsson, T. (eds.) *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, IEEE Press, Washington, DC, pp. 1678–1685.

Hereford, J., 2010. Bio-inspired search strategies for robot swarms. In Martinez, E. (ed.) *Swarm Robotics from Biology to Robotics*, InTech, Rijeka, Croatia, pp. 1–26.

Hijaze, M., Corne, D., 2009. An investigation of topologies and migration schemes for asynchronous distributed evolutionary algorithms. In *Proceedings of the World Congress on Nature and Biologically Inspired Computing*. IEEE Press, Washington, DC.

Hiroyasu, T., Miki, M., Kamiura, J., Watanabe, S., Hiroyasu, H., 2005. MOGADES: multi-objective genetic algorithm with distributed environment scheme. *Evolutionary Multiobjective Optimization*, Springer-Verlag, London, pp. 201–227.

Hoekstra, A.G., Sloot, P.M.A., 2005. Introducing grid speedup gamma: A scalability metric for parallel applications on the grid. In Sloot, P., Hoekstra, T., Priol, T., Reinefeld, A., Bubak. M. (eds.), *Advances in Grid Computing – EGC 2005, Lecture Notes in Computer Science* 3470. Springer, Berlin, Heidelberg, pp. 245–249.

Homberger, J., 2008. A parallel genetic algorithm for the multilevel unconstrained lot-sizing problem. *INFORMS Journal on Computing* 20, 1, 124–132.

Homberger, J., Gehring, H., 2008. A two-level parallel genetic algorithm for the uncapacitated warehouse location problem. In *Proceedings of the Annual Hawaii International Conference on System Sciences*.

Hongwei, X., Yanhua, L., 2009. Parallel ACO for DNA sequencing by hybridization. In *Proceedings of the World Congress on Computer Science and Information Engineering*. IEEE Computer Society, Washington, DC, pp. 602–606.

Huang, H., Tsai, C., Lin, S., 2009. Sopc-based parallel elite genetic algorithm for global path planning of an autonomous omnidirectional mobile robot. In *Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics*, IEEE Press, Piscataway, NJ, pp. 1959–1964.

Hung, Y., Chen, W., 2010. A heterogeneous cooperative parallel search of branch-and-bound method and tabu search algorithm. *Journal of Global Optimization*, 51, 1, 133–148.

Ibri, S., Drias, H., Nourelfath, M., 2010. A parallel hybrid ant-tabu algorithm for integrated emergency vehicle dispatching and covering problem. *International Journal of Innovative Computing and Applications* 2, 4, 226–236.

Islam, R., Ngom, A., 2006. Protein threading using parallel evolution strategy. In Yen, G.G., Lucas, S.M., Fogel, G., Kendall, G., Salomon, R., Zhang, B.-T., Coello Coello, C.A., Runarsson, T.P. (eds.) *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*. IEEE Press, Washington, DC, pp. 2347–2354.

Jaimes, A.L., Coello, C.A., 2007. MRMOGA: a new parallel multi-objective evolutionary algorithm based on the use of multiple resolutions. *Concurrency and Computation: Practice and Experience* 19, 4, 397–441.

James, T., Rego, C., Glover, F., 2005. Sequential and parallel path-relinking algorithms for the quadratic assignment problem. *IEEE Intelligent Systems* 20, 58–65.

Janson, S., Alba, E., Dorronsoro, B., Middendorf, M., 2006. Hierarchical cellular genetic algorithm. In Gottlieb, J., Raidl, G. (eds.) *Evolutionary Computation in Combinatorial Optimization*, *Lecture Notes in Computer Science*, Vol. 3906. Springer, Berlin, Heidelberg, pp. 111–122.

Jewajinda, Y., Chongstitvatana, P., 2008. FPGA implementation of a cellular compact genetic algorithm. In *Proceedings of the 2008 NASA/ESA Conference on Adaptive Hardware and Systems*, IEEE Computer Society, Washington, DC, USA, pp. 385–390.

Jiménez, J., Merelo, J., and Castillo, P., 2010. Evolvable agents: a framework for peer-to-peer evolutionary algorithms. In de Vega, F.F., Cantú-Paz, E. (eds.) *Parallel and Distributed Computational Intelligence, Studies in Computational Intelligence*, Vol. 269. Springer, Berlin, pp. 43–62.

Jin, C., Vecchiola, C., Buyya, R., 2008. MRPGA: an extension of mapreduce for parallelizing genetic algorithms. In *Proceedings of the IEEE International Conference on eScience*, pp. 214–221.

Jovanovic, R., Tuba, M., Simian, D., 2010. Comparison of different topologies for island-based multi-colony ant algorithms for the minimum weight vertex cover problem. *WSEAS Transactions on Computers* 9, 1, 83–92.

Kalinli, A., Sagiroglu, S., Sarikoc, F., 2010. Parallel ant colony optimization algorithm based neural method for determining resonant frequencies of various microstrip antennas. *Electromagnetics* 30, 5, 463–481.

Karnan, M., Gopal, N., 2010. Hybrid Markov random field with parallel Ant Colony Optimization and fuzzy C means for MRI brain image segmentation. In *Proceedings of the 2010 IEEE International Conference on Computational Intelligence and Computing Research, ICCIC 2010*, pp. 718–721.

Karp, A.H., Flatt, H.P., 1990. Measuring parallel processor performance. *Communications of the ACM* 33, 5, 539–543.

Khouadjia, M.R., Sarasola, B., Alba, E., Jourdan, L., Talbi, E.G., 2011. Multi-environmental cooperative parallel metaheuristics for solving dynamic optimization problems. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11), Workshop on Nature Inspired Distributed Computing (NIDISC'11)*. IEEE Press, Washington, DC.

Kokosiński, Z., Kwarciany, K., 2007. On sum coloring of graphs with parallel genetic algorithms. In *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms, Part I*, ICANNGA '07, Springer-Verlag, Berlin, Heidelberg, pp. 211–219.

Krömer, P., .Snåšel, V., Platoš, J., Abraham, A., 2011. Many-threaded implementation of differential evolution for the cuda platform. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, ACM, New York, pp. 1595–1602.

Langdon, W., 2010. Large scale bioinformatics data mining with parallel genetic programming on graphics processing units. In de Vega, F.F., Cantú-Paz, E. (eds.) *Parallel and Distributed Computational Intelligence, Studies in Computational Intelligence*, Vol. 269. Springer-Verlag, Berlin, Heidelberg, pp. 113–141.

Langdon, W., Banzhaf, W., 2008. A SIMD interpreter for genetic programming on GPU graphics cards. In *Proceedings of the 11th European Conference on Genetic Programming*, Springer-Verlag, Berlin, Heidelberg, pp. 73–85.

Laredo, J., Castillo, P., Mora, A., Merelo, J., Fernandes, C., 2008a. Resilience to churn of a peer-to-peer evolutionary algorithm. *International Journal of High Performance Systems Architecture* 1, 260–268.

Laredo, J., Eiben, A., Steen, M., Castillo, P., Mora, A., Merelo, J., 2008b. P2P evolutionary algorithms: a suitable approach for tackling large instances in hard optimization problems. In *Proceedings of the 14th International Euro-Par Conference on Parallel Processing*, Euro-Par '08, Springer-Verlag, Berlin, Heidelberg, pp. 622–631.

Lässig, J., Sudholt, D., 2010a. General scheme for analyzing running times of parallel evolutionary algorithms. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I*. Springer-Verlag, Berlin, Heidelberg, pp. 234–243.

Lässig, J., Sudholt, D., 2010b. The benefit of migration in parallel evolutionary algorithms. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York, pp. 1105–1112.

Lässig, J., Sudholt, D., 2011. Adaptive population models for offspring population and parallel evolutionary algorithms. In *Proceedings of the FOGA11*.

Lau, W., Lee, K., Leung, K., 2006. A hybridized genetic parallel programming based logic circuit synthesizer. In *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, ACM Press, New York, pp. 839–846.

León, C., Miranda, G., Segura, C., 2009. A memetic algorithm and a parallel hyperheuristic island-based model for a 2D packing problem. In *GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, pp. 1371–1378.

Lewis, T., Magoulas, G., 2009. Strategies to minimise the total run time of cyclic graph based genetic programming with GPUs. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, ACM, New York, pp. 1379–1386.

Li, J., Wang, X., He, R., Chi, Z., 2007a. An efficient fine-grained parallel genetic algorithm based on GPU-accelerated. In *Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops*, NPC '07, IEEE Computer Society, Washington, DC, pp. 855–862.

Li, J., Zhang, L., Liu, L., 2009. A parallel immune algorithm based on fine-grained model with GPU-acceleration. In *Proceedings of the 2009 Fourth International Conference on Innovative Computing, Information and Control*, ICICIC '09, IEEE Computer Society, Washington, DC, pp. 683–686.

Li, N., Gao, D., Gong, G., Chen, Z., 2010. Realization of parallel ant colony algorithm based on TBB multi-core platform. In *Proceedings - 2010 International Forum on Information Technology and Applications, IFITA 2010*, Vol. 1, pp. 177–180.

Li, X., Yu, X., Luo, X., 2007b. Parallel implementation of Ant Colony Optimization for vector quantization codebook design. In *Proceedings of the 3rd International Conference on Natural Computation*, IEEE Computer Society, Washington, DC, pp. 787–791.

Li, Z., Bai, H., 2010. Multi-Ant Colony Optimization algorithm for the route optimization of logistic distribution. In *Proceedings of the 2010 2nd International Conference on Computational Intelligence and Natural Computing, CINC 2010*, Vol. 1, pp. 141–144.

Liefooghe, A., 2010. Metaheuristics for multiobjective optimisation—cooperative approaches, uncertainty handling and application in logistics. *Journal on Operational Research*, 9, 1–4.

Lim, D., Ong, Y., Jin, Y., Sendhoff, B., Lee, B., 2007. Efficient hierarchical parallel genetic algorithms using grid computing. *Future Generation Computer Systems* 23, 4, 658–670.

Limmer, S., Fey, D., 2010 Framework for distributed evolutionary algorithms in computational grids. In *Proceedings of the 5th international conference on Advances in computation and intelligence*, ISICA'10, Springer-Verlag, Berlin, Heidelberg, pp. 170–180.

Liu, C., Li, L., Xiang, Y., 2008a. Research of multi-path routing protocol based on parallel Ant Colony algorithm optimization in mobile Ad Hoc networks. In *Proceedings of the 5th International Conference on Information Technology: New Generations*, IEEE Computer Society, Washington, DC, pp. 1006–1010.

Liu, C., Sun, Y., Zhang, C., 2008b. Chaotic parallel genetic algorithm with variable-scale learning and balancing strategy of ranking individuals. In *Proceedings of the 2008 Second International Symposium on Intelligent Information Technology Application, Vol. 01*, IITA '08, IEEE Computer Society, Washington, DC, pp. 818–822.

Llorà, X., Verma, A., Campbell, R., Goldberg, D., 2010. When huge is routine: scaling genetic algorithms and estimation of distribution algorithms via data-intensive computing. In Fernández, F., Cantú-Paz, E. (eds.) *Parallel and Distributed Computational Intelligence, Studies in Computational Intelligence*, Vol. 269. Springer, Berlin, Heidelberg, pp. 11–41.

Lombrana, D., Vega, F., Fernández, F., Trujillo, L., Olague, G., Araujo, L., Castillo, P., Merelo, J., Sharman, K., 2009. Increasing GP computing power for free via desktop GRID computing and virtualization. In *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Washington, DC, pp. 419–423.

Lopez, F., Torres, M., Batista, B., Pérez, J., Moreno-Vega, J., 2006. Solving feature subset selection problem by a parallel scatter search. *European Journal of Operational Research* 169, 477–489.

López-Ibánez, M., Prasad, T., Paechter, B., 2009. Parallel optimisation of pump schedules with a thread-safe variant of EPANET toolkit. In *Geotechnical Special Publication*, Vol. 187, ASCE, New York, pp. 462–471.

Lucka, M., Piecka, S., 2009. Multi-threaded ant colony optimization with asynchronous communications for the vehicle routing problem. *Komunikacie* 11, 4, 5–8.

Lukasiewycz, M., 2009. Meta-Heuristic Optimization Framework for Java. Available at http://www.opt4j.org (accessed September 2011).

Lukasik, S., Kokosinski, Z., Swieton, G., 2008. Parallel simulated annealing algorithm for graph coloring problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4967 LNCS, Springer-Verlag, Berlin, Heidelberg, pp. 229–238.

Luke, S., Panait, L., Balan, G., 2007. ECJ 16: A Java-based Evolutionary Computation Research System. Available at http://cs.gmu.edu/ eclab/projects/ecj/ (accessed September 2011).

Luna, F., Nebro, A., Alba, E., 2006. Parallel evolutionary multiobjective optimization. *Parallel Evolutionary Computations* 22, 33–56.

Luna, F., Nebro, A., Alba, E., Durillo, J., 2008. Solving large-scale real-world telecommunication problems using a grid-based genetic algorithm. *Engineering Optimization* 40, 11, 1067–1084.

Luo, Z., Liu, H., 2006. Cellular genetic algorithms and local search for 3-SAT problem on graphic hardware. In *Proceedings of the IEEE Congress on Evolutionary Computation, 2006. CEC 2006*, pp. 2988–2992.

Luong, T., Melab, N., Talbi, E.-G., 2010. GPU-based island model for evolutionary algorithms. In *GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, pp. 1089–1096.

Luque, G., Alba, E., 2010. Selection pressure and takeover time of distributed evolutionary algorithms. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, pp. 1083–1088.

Luque, G., Alba, E., 2011. *Parallel Genetic Algorithms. Theory and Real World Applications*. Springer-Verlag, Berlin.

Luque, G., Alba, E., Dorronsoro, B., 2009. An asynchronous parallel implementation of a cellular genetic algorithm for combinatorial optimization. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, pp. 1395–1402.

Luque, G., Luna, F., Alba, E., 2010. A new parallel cooperative model for trajectory based metaheuristics. *Distributed Computing and Artificial Intelligence* 79, 559–567.

Luque, G., Luna, F., Alba, E., Nesmachnow, S., 2011. Exploring the accuracy of a parallel cooperative model for trajectory-based metaheuristics. In *Proceedings of the 13th International Conference on Computer Aided Systems Theory, EUROCAST 2011*.

MacCallum, R., 2003. Introducing a Perl genetic programming system-and can meta-evolution solve the bloat problem? *Genetic Programming* 1, 17–49.

Maitre, O., Baumes, L., Lachiche, N., Corma, A., Collet, P., 2009. Coarse grain parallelization of evolutionary algorithms on GPGPU cards with EASEA. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, ACM, New York, pp. 1403–1410.

Man-Leung, W., Tien-Tsin, W., 2006. Parallel hybrid genetic algorithms on consumer-level graphics hardware. In Yen, G.G., Lucas, S.M., Fogel, G., Kendall, G., Salomon, R., Zhang, B.-T., Coello Coello, C.A., Runarsson, T.P. (eds.) *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, IEEE Press, Washington, DC, pp. 2973–2980.

Mansour, N., Haidar, G., 2010. Parallel metaheuristic algorithm for exam timetabling. In *Proceedings - 2010 6th International Conference on Natural Computation, ICNC 2010*, Vol. 1, pp. 471–475.

Martins, S., Ribeiro, C., Rosseti, I., 2006. Applications of parallel metaheuristics to optimization problems in telecommunications and bioinformatics. In Talbi, E.-G. (ed.), *Parallel Combinatorial Optimization*, John Wiley & Sons, New York, pp. 301–325.

McGeoch, C.C., 2007. Experimental algorithmics. *Communications of the ACM* 50, 11, 27–31.

McGeoch, C.C., 2008. Experimental methods for algorithm analysis. In Kao, M.-Y. (ed.) *Encyclopedia of Algorithms*. Springer-Verlag, Berlin, Heidelberg.

Melab, N., Mezmaz, M., Talbi, E.-G., 2006a. Parallel cooperative meta-heuristics on the computational grid: a case study: the bi-objective flow-shop problem. *Parallel Computing* 32, 643–659.

Melab, N., Talbi, E.-G., Mezmaz, M.-S., Wei, B., 2006b. Parallel hybrid multi-objective meta-heuristics on P2P systems. In Olariu, S., Zomaya, A.Y. (eds.) *Handbook of Bioinspired Algorithms and Applications*. Chapman & Hall/CRC, Taylor & Francis, Boca Raton, FL, pp. 649–663.

Mendes, R., Mohais, A.S., 2005. DynDE: a differential evolution for dynamic optimization problems. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation, 2005*, Vol. 3, IEEE, Washington, DC, pp. 2808–2815.

Mirghania, B., Mahinthakumar, K., Tryby, M., Ranjithan, R., Zechman, E., 2009. A parallel evolutionary strategy based simulation-optimization approach for solving groundwater source identification problems. *Advances in Water Resources* 32, 9, 1373–1385.

Mocholí, J., Martínez, J., Canós, J., 2005. A grid Ant Colony algorithm for the orienteering problem. In *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE Press, Washington, DC, pp. 942–949.

Mohammed, S., Noor, S., Kamakoti, V., 2011. Hardware based genetic evolution of self-adaptive arbitrary response FIR filters. *Applications of Soft Computing* 11, 842–854.

Montgomery, D.C., 2000. *Design and Analysis of Experiments* (5th edn). John Wiley & Sons, New York.

Mostaghim, S., 2010. Parallel multi-objective optimization using self-organized heterogeneous resources. In *Parallel and Distributed Computational Intelligence, Studies in Computational Intelligence*, Vol. 269. Springer-Verlag, Berlin, Heidelberg, pp. 165–179.

Muelas, S., Pena, J.M., Robles, V., LaTorre, A., 2008. Voronoi-initializated island models for solving real-coded deceptive problems. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, pp. 993–1000.

Mueller, C., Baumgartner, B., Ofenbeck, G., Schrader, B., Sbalzarini, I., 2009. pCMALib: a parallel fortran 90 library for the evolution strategy with covariance matrix adaptation. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, ACM, New York.

Munawar, A., Wahib, M., Munetomo, M., Akama, K., 2008. Solving large instances of capacitated vehicle routing problem over cell be. In *Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, IEEE Computer Society, Washington, DC, USA, pp. 131–138.

Munawar, A., Wahib, M., Munetomo, M., Akama, K., 2009. Hybrid of genetic algorithm and local search to solve MAX-SAT problem using nVidia CUDA framework. *Genetic Programming and Evolvable Machines* 10, 391–415.

Munshi, A., 2009. The OpenCL specification. *Khronos OpenCL Working Group* 1, 11–15.

Narasimhan, H., 2009. Parallel artificial bee colony (pabc) algorithm. In *Proceedings of the 2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009*, pp. 306–311.

Nebro, A., Durillo, J., 2010. A study of the parallelization of the multi-objective metaheuristic MOEA/D. *Learning and Intelligent Optimization* 6073, 303–317.

Nebro, A.J., Durillo, J.J., Luna, F., Dorronsoro, B., Alba, E., 2006. A cellular genetic algorithm for multiobjective optimization. In *Proceedings of the Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO 2006)*, Citeseer, pp. 25–36.

Nebro, A.J., Durillo, J.J., Luna, F., Dorronsoro, B., Alba, E., 2009. MOCell: a cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems* 24, 7, 726–746.

Nebro, A.J., Luque, G., Alba, E., 2008a. DNA fragment assembly using Grid systems. In *Grid Computing for Bioinformatics and Computational Biology*. Wiley, New York, pp. 357–374.

Nebro, A., Luque, G., Luna, F., Alba, E., 2008b. DNA fragment assembly using a grid-based genetic algorithm. *Computers & Operations Research* 35, 9, 2776–2790.

Nedjah, N., Alba, E., de Macedo Mourelle, L., 2006. *Parallel Evolutionary Computations*. Springer-Verlag, Berlin, Heidelberg.

Nesmachnow, S., Alba, E., Cancela, E., 2012a. Scheduling in heterogeneous computing and grid environments using a parallel CHC evolutionary algorithm. *Computational Intelligence* 28, 2, 131–155.

Nesmachnow, S., Cancela, H., Alba, E., 2007. Evolutionary algorithms applied to reliable communication network design. *Nature-Inspired Informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science and Engineering* 39, 7, 831–855.

Nesmachnow, S., Cancela, H., Alba, E., 2009. Nature-inspired informatics for telecommunication network design. In Chiong, R. (ed.) *Nature-Inspired Informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science and Engineering*. IGI Global, Hershey, PA, pp. 323–371.

Nesmachnow, S., Cancela, H., Alba, E., 2011. Heterogeneous computing scheduling with evolutionary algorithms. *Soft Computing* 15, 4, 685–699.

Nesmachnow, S., Cancela, H., Alba, E., 2012b. A parallel micro-CHC evolutionary algorithm for heterogeneous computing and grid scheduling. *Applied Soft Computing Journal* 12, 626–639.

Nesmachnow, S., Iturriaga, S., 2012. Multiobjective grid scheduling using a domain decomposition based parallel micro evolutionary algorithm. *International Journal of Grid and Utility Computing* (to appear).

Nguyen, Q.H., Ong, Y.S., Lim, M.H., Krasgonor, N., 2009. Adaptive cellular memetic algorithm. *Evolutionary Computation* 17, 2, 231–256.

NVIDIA C. 2007. *CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA, Santa Clara, CA, pp. 33, 129.

Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J., 2007. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum* 26, 80–113.

Ozbakir, L., Baykasoglu, A., Gorkemli, B., Gorkemli, L., 2011. Multiple-colony ant algorithm for parallel assembly line balancing problem. *Applied Soft Computing Journal* 11, 3, 3186–3198.

Parejo, J.A., Ruiz-Cortés, A., Lozano, S., Fernandez, P., 2012. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing – A Fusion of Foundations, Methodologies and Applications* 16, 3, 527–561.

Payne, J.L., Eppstein, M.J., 2007. Using pair approximations to predict takeover dynamics in spatially structured populations. In *Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation*, ACM, New York, pp. 2557–2564.

Pedemonte, M., Cancela, H., 2010. A cellular Ant Colony Optimisation for the generalised Steiner problem. *International Journal of Innovative Computing and Applications* 2, 3, 188–201.

Peng, C., Sun, H., Rao, P., Guo, J., 2010. An application of parallel ACO algorithm in power network node code multi-scheme optimization. In *Proceedings of the ICCET 2010-2010 International Conference on Computer Engineering and Technology*, Vol. 4, Washington, DC, pp. V4436–V4440.

Peng, W., Tong, R., Qian, G., Dong, J., 2006. A constrained Ant Colony algorithm for image registration. In *International Conference on Intelligent Computing, Lecture Notes in Computer Science*, Vol. 4115. Springer, Berlin, Heidelberg, pp. 1–11.

Peng, W., Tong, R., Tang, M., Dong, J., 2005. Ant Colony search algorithms for optimal packing problem. In *First International Conference on Advances in Natural Computation, Lecture Notes in Computer Science*, Vol. 3611. Springer, Berlin, Heidelberg, pp. 1229–1238.

Perez, C., Miguel, J., Mendiburu, A., 2009. Evaluating the cell broadband engine as a platform to run estimation of distribution algorithms. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, ACM, New York.

Pospíchal, P., Jaros, J., Schwarz, J., 2010a. Parallel genetic algorithm on the CUDA architecture. In *Applications of Evolutionary Computation*, LNCS 6024, Springer-Verlag, Berlin, Heidelberg, pp. 442–451.

Pospíchal, P., Schwarz, J., Jaros, J., 2010b. Parallel genetic algorithm solving 0/1 knapsack problem running on the GPU. In *Proceedings of the 16th International Conference on Soft Computing MENDEL 2010*, Brno University of Technology, pp. 64–70.

Rardin, R.L., Uzsoy, R., 2001. Experimental evaluation of heuristic optimization algorihtms: a tutorial. *Journal of Heuristics* 7, 3, 261–304.

Rausch, T., Thomas, A., Camp, N., Cannon L., Facelli, J., 2008. A parallel genetic algorithm to discover patterns in genetic markers that indicate predisposition to multifactorial disease. *Comput. Biol. Med.* 38, 826–836.

Reinelt, G., 1991. TSPLIB - a travelling salesman problem library. *ORSA – Journal of Computing* 3, 376–384.

Ribeiro, C., Rosseti, I., 2007. Efficient parallel cooperative implementations of GRASP heuristics. *Parallel Computing* 33, 1, 21–35.

Risco-Martin, J., Atienza, D., Hidalgo, J., Lanchares, J., 2008. A parallel evolutionary algorithm to optimize dynamic data types in embedded systems. *Soft Computing – A Fusion of Foundations, Methodologies and Applications* 12, 12, 1157–1167.

Risco-Martin, J., Colmenar, J., Gonzalo, R., 2009. A parallel evolutionary algorithm to optimize dynamic memory managers in embedded systems. In *WPABA'09: Proceedings of the Second International Workshop on Parallel*

Architectures and Bioinspired Algorithms (WPABA 2009), Universidad Complutense de Madrid, Raleigh, NC, USA, September 12–16, 2009, pp. 21–30.

Robilliard, D., Marion-Poty, V., Fonlupt, C., 2009. Genetic programming on graphics processing units. *Genetic Programming and Evolvable Machines* 10, 4, 447–471.

Roozmand, O., Zamanifar, K., 2008. Parallel Ant miner 2. In *9th International Conference on Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science*, Vol. 5097. Springer, Berlin, Heidelberg, pp. 681–692.

Rudolph, G., 2006. Takeover time in parallel populations with migration. In *BIOMA*. University of Ljubijana, Ljubijana, Slovenia.

Sadeg, S., Drias, H., 2007. A selective approach to parallelise bees swarm optimisation metaheuristic; application to MAX-W-SAT. *Int. J. Innov. Comput. Appl.* 1, 146–158.

Sait, S., Ali, M., Zaidi, A., 2007. Evaluating parallel simulated evolution strategies for vlsi cell placement. *Journal of Mathematical Modelling and Algorithms* 6, 3, 433–454.

Sait, S., Khan, K., Ali, M., 2008. Parallel strategies for stochastic evolution. *Journal of Universal Computer Science* 14, 15, 2471–2490.

Salmani, M., Kamal, M., Fakhraie, S., Ahmadabadi, M., 2006. SOPC-based parallel genetic algorithm. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation, CEC 2006*, IEEE Press, Washington, DC, pp. 2800–2806.

Sasaki, D., Keane, A., Shahpar, S., 2006. Multiobjective evolutionary optimization of a compressor stage using a grid-enabled environment. *44th AIAA Aerospace Sciences Meeting and Exhibit*, pp. 1–18.

Segura, C., Cervantes, A., Nebro, A., Jaraíz, M., Segredo, E., García, S., Luna, F., Gómez Pulido, J., Miranda, G., Luque, C., Alba, E., Vega Rodríguez, M., León, C., Galván, I., 2009. Optimizing the DFCN Broadcast protocol with a parallel cooperative strategy of multi-objective evolutionary algorithms. In *Evolutionary Multi-Criterion Optimization, 5th International Conference, EMO 2009*, Nantes, France, April 7–10, 2009. Proceedings, Lecture Notes in Computer Science, Vol. 5467. Springer, Berlin, Heidelberg, pp. 305–319.

Segura, C., Miranda, G., León, C., 2011a. Parallel hyperheuristics for the frequency assignment problem. *Memetic Computing* 3, 1, 33–49.

Segura, C., Segredo, E., León, C., 2011b. Parallel island-based multiobjectivised memetic algorithms for a 2d packing problem. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, ACM, New York, pp. 1611–1618.

Shylo, O.V., Middelkoop, T., Pardalos, P.M., 2011. Restart strategies in optimization: parallel and serial cases. *Parallel Computing* 37, 1, 60–68.

Sinha, E., Minsker, B., 2007. Multiscale island injection genetic algorithms for groundwater remediation. *Advances in Water Resources* 30, 9, 1933–1942.

Skolicki, Z., DeJong, K.A., 2005. The influence of migration size and intervals on island models. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, pp. 1295–1302.

Subramanian, A., Drummond, L., Bentes, C., Ochi, L., Farias, R., 2010. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers and Operations Research* 37, 11, 1899–1911.

Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.-P., Auger, A., Tiwari, S., 2005. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. *Technical Report KanGAL Report* 2005005, Nanyang Technological University, IIT Kanpur, India, May, 2005.

Talbi, E.-G. (ed.), 2006. *Parallel Combinatorial Optimization*. Wiley, New York.

Talbi, E.-G., Bachelet, V., 2006. COSEARCH: a parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms* 5, 1, 5–22.

Talbi, E.-G., Cahon, S., Melab, N., 2007. Designing cellular networks using a parallel hybrid metaheuristic on the computational grid. *Computer Communications* 30, 4, 698–713.

Tanese, R., 1989. Distributed genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, Burlington, MA, pp. 434–439.

Tang, K., Yao, X., Suganthan, P.N., MacNish, C., Chen, Y.P., Chen, C.M., Yang, Z., 2007a. Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China.

Tang, Y., Reed, P., Kollat, J., 2007b. Parallelization strategies for rapid and robust evolutionary multiobjective optimization in water resources applications. *Advances in Water Resources* 30, 3, 335–353.

Tantar, A. , Melab, N., Talbi, E.-G., Parent, B., Horvath, D., 2007. A parallel hybrid genetic algorithm for protein structure prediction on the computational grid. *Future Generation Computer Systems* 23, 3, 398–409.

Taskova, K., Korosec, P., Silc, J., 2010. A distributed multilevel ant-colony approach for finite element mesh decomposition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6068 LNCS(PART 2), Springer-Verlag, Berlin, Heidelberg, pp. 398–407.

Thain, D., Tannenbaum, T., Livny, M., 2002. Condor and the grid. In Berman, F., Fox, G., Hey, T. (eds.) *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, New York.

Tsutsui, S., 2007. Cunning Ant System for quadratic assignment problem with local search and parallelization. In *2nd International Conference on Pattern Recognition and Machine Intelligence, Lecture Notes in Computer Science*, Vol. 4815. Springer, Berlin, Heidelberg, pp. 269–278.

Tsutsui, S., 2008. Parallel Ant Colony Optimization for the quadratic assignment problems with symmetric multi processing. In *6th International Conference on Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science*, Vol. 5217. Springer, Berlin, Heidelberg, pp. 363–370.

Tsutsui, S., 2010. Parallelization of an evolutionary algorithm on a platform with multi-core processors. In *Proceedings of the 9th International Conference on Artificial Evolution*, Springer-Verlag, Berlin, Heidelberg, pp. 61–73.

Tsutsui, S., Fujimoto, N., 2009. Solving quadratic assignment problems by genetic algorithms with GPU computation: a case study. In *GECCO '09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*, ACM, New York, pp. 2523–2530.

Tsutsui, S., Fujimoto, N., 2010. Parallel ant colony optimization algorithm on a multi-core processor. *Lecture Notes in Computer Science* 6234, 488–495.

Tu, K., Liang, Z., 2011. Parallel computation models of particle swarm optimization implemented by multiple threads. *Expert Systems with Applications* 38, 5858–5866.

Urlings, T., Ruiz, R., Şerifoğlu, F.S., 2008. Genetic algorithms for complex hybrid flexible flow line problems. *International Journal of Metaheuristics* 1, 253–271.

Uzsoy, R., Demirkol, E., Mehta, S.V., 1998. Benchmarks for shop scheduling problems. *European Journal of Operational Research* 109, 137–141.

Van Luong, T., Melab, N., Talbi, E.-G., 2010. A GPU-based iterated tabu search for solving the quadratic 3-dimensional assignment problem. In *Workshop on Parallel Optimization in Emerging Computing Environments (POECE) in Conjunction with the International Conference on Computer Systems and Applications (AICCSA)*, Hammamet, Tunisia.

Velte, T., Velte, A., Elsenpeter, R., 2010. *Cloud Computing, A Practical Approach* (1st edn). McGraw-Hill, New York.

Verma, A., X. Llorà, Venkataraman, S., Goldberg, D., Campbell, R., 2010. Scaling eCGA model building via data-intensive computing. In *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE Press, Washington, DC, pp. 1–8.

Vidal, P., Alba, E., 2010a. A multi-GPU implementation of a Cellular Genetic Algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE Press, Washington, DC, pp. 1–7.

Vidal, P., Alba, E., 2010b. Cellular genetic algorithm on graphic processing units. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, Springer-Verlag, Berlin, Heidelberg, pp. 223–232.

Vrajitoru, D., 2010. Shared memory genetic algorithms in a multi-agent context. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, pp. 1097–1104.

Wagner, S., Affenzeller, M., 2005. Heuristiclab: a generic and extensible optimization environment. *Adaptive and Natural Computing Algorithms*, Springer-Verlag, Berlin, Heidelberg, pp. 538–541.

Walton, M., Grewal, G., Darlington, G., 2010. Parallel FPGA-based implementation of scatter search. In *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10*, ACM, New York, pp. 1075–1082.

Wang, D., Wang, D., Yan, Y., Wang, H., 2008. An adaptive version of parallel MPSO with OpenMP for uncapacitated facility location problem. *Control and Decision Conference, 2008. CCDC 2008. Chinese*, pp. 2387–2391.

Wang, Z., 2009. Reliability speedup: an effective metric for parallel application with checkpointing. In *Proceedings of the PDCAT*, IEEE Computer Society, Washington, DC, pp. 247–254.

Weis, G., Lewis, A., 2009. Using XMPP for ad-hoc grid computing – an application example using parallel Ant Colony Optimisation. In *International Symposium on Parallel and Distributed Processing*, 1–4.

Weise, T., Geihs, K., 2006. DGPF – an adaptable framework for distributed multi-objective search algorithms applied to the genetic programming of sensor networks. In Filipič, B., Šilc, J. (eds.), *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, International Conference on Bioinspired Optimization Methods and their Application (BIOMA). Jožef Stefan Institute, Ljubljana, Slovenia, October, 2006, pp. 157–166.

Whitley, D., 2001. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information & Software Technology* 43, 14, 817–831.

Whitley, D., Chicano, F., Alba, E., Luna, F., 2010. Elementary landscapes of frequency assignment problems. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, ACM, New York, pp. 1409–1416.

Wilson, G., Banzhaf, W., 2008. Linear genetic programming GPGPU on Microsoft's Xbox 360. In *Proceedings of the IEEE Congress on Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*, IEEE Press, Washington, DC, pp. 378–385.

Wilson, G., Banzhaf, W., 2009. Deployment of CPU and GPU-based genetic programming on heterogeneous devices. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*, GECCO '09, ACM, New York, pp. 2531–2538.

Wilson, S.W., 1995. Classifier fitness based on accuracy. *Evolutionary Computation* 3, 2, 149–175.

Wirawan, A., Keong, K., Schmidt, B., 2008. Parallel DNA sequence alignment on the cell broadband engine. In *Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics*, PPAM'07, Springer-Verlag, Berlin, Heidelberg, pp. 1249–1256.

Wong, M., Wong, T., 2009. Implementation of parallel genetic algorithms on graphics processing units. In *Intelligent and Evolutionary Systems*, Springer, Berlin, Heidelberg, pp. 197–216.

Xhafa, F., Abraham, A. (eds.), 2008. *Metaheuristics for Scheduling in Distributed Computing Environments*. Number 146 in Studies in Computational Intelligence. Springer-Verlag, Berlin, Heidelberg.

Xiong, J., Liu, C., Chen, Z., 2008. A new parallel ant colony optimization algorithm based on message passing interface. In *Proceedings of the 2008 Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, Vol. 2, pp. 178–182.

Xiong, J., Meng, X., Liu, C., 2010. An improved parallel ant colony optimization based on message passing interface. *Lecture Notes in Computer Science* 6145, 1, 249–256.

Yang, Z., Yu, B., Cheng, C., 2007. A parallel Ant Colony algorithm for bus network optimization. *Computer-Aided Civil and Infrastructure Engineering*, 22, 1, 44–55.

Yu, B., Yang, Z.-Z., Xie, J.-X., 2011. A parallel improved ant colony optimization for multi-depot vehicle routing problem. *Journal of the Operational Research Society* 62, 1, 183–188.

Yu, Q., Chen, C., Pan, Z., 2005. Parallel genetic algorithms on programmable graphics hardware. In *Proceedings of the Advances in Natural Computation, First International Conference, ICNC 2005, Part III, Lecture Notes in Computer Science*, Vol. 3612. Springer, Berlin, Heidelberg, pp. 1051–1059.

Zhang, Q., Li, H., 2007. MOEA/D: a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6, 712–731.

Zhang, Q., Liu, W., Li, H., 2009. The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances. In *Proceedings of the IEEE Congress on Evolutionary Computation, 2009. CEC'09*, IEEE, Washington, DC, pp. 203–208.

Zhao, B., Guo, C., Zhang, P., Cao, Y., 2005. Distributed cooperative particle swarm optimization algorithm for reactive power optimization. *Proceedings of the Chinese Society of Electrical Engineering* 25, 21, 1–7.

Zhao, J., Liu, Q., Wang, W., Wei, Z., Shi, P., 2011. A parallel immune algorithm for traveling salesman problem and its application on cold rolling scheduling. *Information Science* 181, 1212–1223.

Zhao, Y., Zhang, X., Kang, T., 2010. A parallel ant colony optimization algorithm for site location. *Acta Geodaetica et Cartographica Sinica* 39, 3, 322–327.

Zhu, W., 2009. A study of parallel evolution strategy: pattern search on a GPU computing platform. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, GEC '09.

Zhu, W., Curry, J., 2009. Parallel ant colony for nonlinear function optimization with graphics hardware acceleration. In *Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics*, IEEE Press, Washington, DC, pp. 1803–1808.

Zhu, Y., Zhang, J., Li, L., Peng, W., 2010. Multiple ant colony routing optimization based on cloud model for WSN with long-chain structure. In *Proceedings of the 2010 6th International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2010*.

Zola, J., Trystram, D., Tchernykh, A., Brizuela, C., 2006. Parallel multiple sequence alignment with local phylogeny search by simulated annealing. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, Vol. 2006.