Adapting Distributed Evolutionary Algorithms to Heterogeneous Hardware

Carolina Salto[1], Enrique Alba[2]

[1] Universidad Nacional de La Pampa - CONICET, General Pico, Argentina
saltoc@ing.unlpam.edu.ar
[2] Universidad de Málaga, Málaga, Spain
eat@lcc.uma.es

Abstract. Distributed computing environments are nowadays composed of many heterogeneous computers able to work cooperatively. Despite this, the most of the work in parallel metaheuristics assumes a homogeneous hardware as the underlying platform. In this work we provide a methodology that enables a distributed genetic algorithm to be customized for higher efficiency on any available hardware resources with different computing power, all of them collaborating to solve the same problem. We analyze the impact of heterogeneity in the resulting performance of a parallel metaheuristic and also its efficiency in time. Our conclusion is that the solution quality is comparable to that achieved by using a theoretically faster homogeneous platform, the traditional environment to execute this kind of algorithms, but an interesting finding is that those solutions are found with a lower numerical effort and even in lower running times in some cases.

1 Introduction

Parallel and distributed computing environments became popular in the past decades as a way to provide the needed computing power to solve complex problems, representing an effective strategy for the execution of distributed evolutionary algorithms (dEAs) [27]. Most of the reported results on dEAs assume that the underlying computing environments have identical features (homogeneous environment) regarding not only hardware (processors, memory, network) but also software (operating system) components [2, 20]. This kind of hardware homogeneity is increasingly difficult to find in modern labs. It is quite hard to maintain a cluster of similar processors along a period of time, because of their failures and the new and different hardware replacing them. Furthermore, the rapid development of technology in designing processors, networks, and data storage devices together with the constantly decreasing ratio between cost and performance allow researchers to use new up-to-date computational resources. As a consequence, the coexistence of new and old equipment in a computing environment has shown the grow of heterogeneous parallel platforms, which are nowadays very common in any laboratory, company, and research institution.

Despite the widespread scenario from the point of view of heterogeneous architectures, the field of metaheuristic algorithms that exploits the heterogeneous architectures in a especialized way has been seldom addressed. A seminal work dealing with heterogeneous computing environments and dEAs can be found in [3]. More recent works about heterogeneous environments can also be found in [6, 12, 15, 17, 22, 23]. These works were focused on solving a given problem, and not much in building a methodology that researchers could use when facing heterogeneous settlements. In this sense, in [12] authors made an original advance in the proposal of a general model to design heterogeneous algorithms depending on the underlying heterogeneous platform.

In the present article we propose a new methodological procedure, and a subsequent algorithmic design, to deal with heterogeneous parallel environments. We called it HAPA: *Hardware Aware Parallel Algorithms* methodology. The goal is to guide an efficient and numerically accurate deployment of a metaheuristic like a dEA or a dGA (distributed genetic algorithm) onto a set of machines with processors running at different clock speeds, dissimilar principal memory capacities, and operating systems. The dGA considered in this work is a multi-population (island) model which performs sparse exchanges of individuals (migration) between the component subpopulations. In short, we address two interesting research questions:

Q1 Can we build an algorithm using the HAPA methodology with a final accuracy comparable to that of existing algorithms running on faster homogeneous platforms?

Q2 Does the use of heterogeneous hardware allow to solve the problem in competitive execution times?

The methodology devised in this work is targeted to address these two questions (real challenges), and it can be summarized as follows. In a first phase, HAPA will analyze the heterogeneous hardware with several different benchmark programs and with the running times obtained from the dGA. These two different measures will allow us to obtain a quantitative measure for the speed of the different hardware involved. This quantitative value is obtained following a well defined methodology, what represents a deeper contribution than only the value itself. With this information, in a second phase, we will develop a novel mechanism to be used in the design of a dGA, engineered to get profit from a computing platform composed of both new and old computational equipment. Finally, in order to answer the two previous questions about the behaviour of the algorithms using HAPA, we have compared a dGA using our proposal against a traditional dGA executed on a homogeneous environment.This constitutes the third phase of the methodology related to the validation of the results.

After using HAPA we hope to be able to report that a dGA using our proposal can obtain similar hit rates as a dGA running over homogeneous hardware. We will show this in this work, as well as we will report on a reduction in the number of function evaluations needed by the new techniques, thus reinforcing the idea of the usefulness of HAPA in the design of competent dGA families of algorithms.

The remainder of this article is structured as follows. Section 2 provides a brief review of the literature dealing with dGAs and heterogeneous hardware. Section 3 presents the HAPA methodology. Section 4 introduces the test problem and the parameterizations used in the experimentation. Section 5 is devoted to describe the heterogeneous hardware. Section 6 provides two possible instantiations in the design of a dGA. Section 7 presents and examines the results validating our proposal. Section 8 summarizes our conclusions and sketches our future work.

2 Background

Let us suppose a company or laboratory has bought a cluster, composed of workstations interconnected by a communication network. Any (even new) cluster of computers will become old with time, and possibly heterogeneous due to changes in its components (partial memory or CPU updates, for example). These components are replaced with different (possibly more powerful) ones. In essence, the cluster is only homogeneous (if at all) when first installed. Additionally, any necessary increment in performance or capacity is usually achieved by replacing old/broken components with more powerful ones. This leads to the coexistence of "leftovers" from the previous installation and "new-comers" that are recently purchased, leading to the emergence of a heterogeneous computing environment in terms of performance and capacity. A situation as the one previously described is sketched in Figure 1.

There exist only a few works proposing new algorithms for heterogeneous platforms, and none on developing a methodology to do so. A seminal proposal concerning the heterogeneous execution of parallel metaheuristics was proposed by Alba *et* al. [3]. In this work, the authors analyzed the way in which heterogeneous environments affect the genetic search to solve a problem, reporting

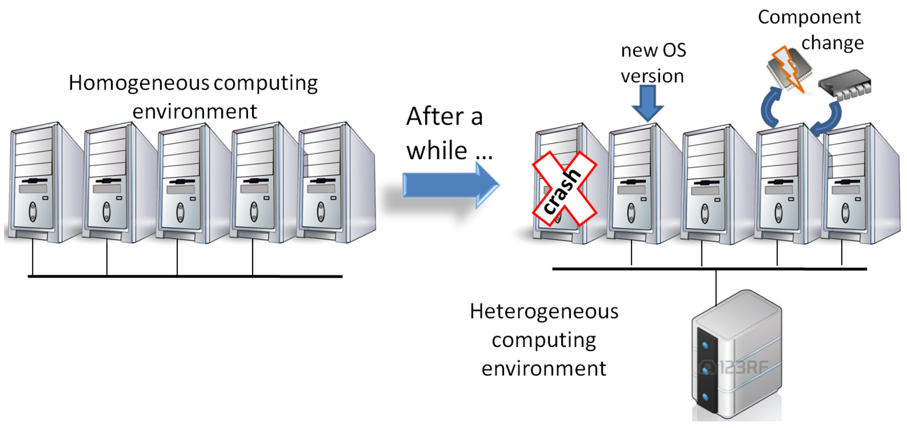

Homogeneous computing environment
After a while ...
crash
new OS version
Component change
Heterogeneous computing environment

Fig. 1. Heterogeneous computing platform

a very significant reduction in the number of steps needed to solve the problem when using heterogeneous hardware. In another interesting work, Chong [9] analyzed the impact of the asynchronous and synchronous communication on a heterogeneous hardware platform and concluded that communication should be non-blocking (i.e. asynchronous) and buffered, a result that has been also confirmed in [4, 6, 19].

Branke *et* al. [8] considered an island model targeted at heterogeneous computer grids and examined different aspects of migration, like the connectivity pattern or the time for migration. They experimented with different ways of sorting the islands: a random sorting of the heterogeneous processors on the ring topology and a minimum and maximum difference sum sorting of the processor velocity. They compared the performance of the standard island model on the homogeneous and the heterogeneous network. They simulated the underlying computer network. Their conclusion was that the result of sorting the computers appropriately in the ring structure is competitive to homogeneous networks. Another significant result was that convergence-based migration leads to a further significant improvement both in homogeneous and heterogeneous environments. Another proposal in this line is the work of Gong *et* al. [17], which also analyzed the influence of different arrangements of heterogeneous computing resources in the execution of dGAs.

A parallel genetic algorithm with a hierarchical distribution was presented in [18], developed using heterogeneous grid computing environments. In each cluster a subpopulation is evolved while the chromosome evaluation is carried out in a different node of the cluster. A theoretical analysis of the speed-up was presented. The empirical study was oriented to analyse the behavior of the algorithm under diverse grid environments having different communication protocols, cluster sizes, computing nodes, and geographically disparate clusters. The authors showed that speed-up can be attained. García *et* al. [15] and Meri *et* al. [22] considered the use of free cloud storage services to communicate a pool of distributed island running in a heterogeneous platform.

Bazterra *et* al. [7] defined performance metrics to understand the parallel efficiency of an algorithm running on heterogeneous systems. They proposed an adaptive parallel genetic algorithm which consists in a client-server model for heterogeneous environments. The server node evolves the population and assigns the evaluation of individuals at each processor depending of their processing velocity. They evaluated the performance of the proposal in a homogeneous and heterogeneous environment, achieving a higher efficiency in the last one. In this line of client-server models, Mostaghim [23] proposed a hybrid method using Multi-objective Particle Swarm optimization and Binary search methods for a multi-objective optimization task independent from the speed of the processors.

More recently, Dominguez and Alba [11, 12] proposed Ethane and HydroCM, two new heterogeneous parallel search algorithms specifically designed for their execution in a heterogeneous hardware platform. The proposed search algorithms, inspired by the chemical structures of ethane and hydrocarbons, were based on genetic algorithms (GA) and simulated annealing (SA). The objective was to give a general kind of parallel search technique that could later be customized to be used with different behavioral algorithms depending on the underlying hardware architecture. The reported results have shown that Ethane and HydroCM can perform better in terms of time and numerical effort when run in heterogeneous software/hardware systems than the component algorithms. Thus, it seems clear that this topic deserves more research since it is both interesting and not well-known at present.

3 The HAPA Methodology

In this section we present the basics of the proposed methodology.HAPA is a methodology in which a distributed population-based metaheuristic can deal with the differences between relative clock speeds of the processors present in a heterogeneous platform. Our methodology consists in computing (once) a ranking of processors in an offline fashion, plus an online use of this information inside the running distributed algorithm in some way (like defining new stopping conditions). This also goes in the sense of "measure once, use many" that can help modern labs to better build algorithms and better use their hardware at the same time. The HAPA methodology comprises three phases:

– Phase 1: Know your Platform.
– Phase 2: Design your Algorithm.
– Phase 3: Get the Results.

Phase 1 numerically describes the heterogeneous computing platform used in the work, with the aim of finding a ranking of processors. Consequently, this phase involves the computation of the relative differences in the velocity of each machine, taking the fastest one as a reference point. For this purpose, we use two different measures to evaluate the performance of machines: the scores from a standard scientific benchmarking software and a fresh ranking coming from the actual execution of a traditional dGA. The rationale behind this is that traditional benchmarks, although useful, are designed to run a set of operations that could not be fully representative of the operations performed in metaheuristics [5]. The objective is to determine which traditional benchmark software, if any, is able to rank the processors in the same order as that the actual dGA running benchmarking, and to use that information for the application of the HAPA methodology. This produces information to numerically know the hardware. Once a ranking is established, a relative velocity factor between each processor B and the fastest processor A can be obtained, denoted as $VF(B)$.

With the ranking obtained in the previous step, Phase 2 consists in the algorithm design of the dGA to deal with the heterogenous platform and to present the HAPA methodology. This online phase is carried out by all the component islands in an organized way. In a first step, each island i of our dGA using the HAPA methodology asks for the features of the processor where it was launched (let us say processor B), obtaining the $VF(B)$ value. Let us suppose, then, the considered parameter value for the island running in the fastest processor is set to X. Therefore, in a second step, each island i proceeds to locally compute the parameter value to a value equal to X divided by $VF(B)$; in that way the islands are coordinating the parameter values depending on the features of the processor where each one was launched. So, the methodology helps to have a more informed decision-making of the distributed algorithm's parameter values in each subpopulation, such as the total number of generations, the migration frequency, and so on. This methodology also prevents the situation that one of the islands is doing most of the work if there is a CPU much faster than the others.

Finally, Phase 3 consists in the evaluation of the proposed dGAs using the HAPA methodology to validate the assumptions made in the Introduction related to their comparative performance with respect to dGAs running in homogeneous platforms regarding final accuracy and execution times.

In Section 6, we explain two possible instantiations of the proposed methodology in the design of a dGA. In Section 6.1 we describe an application of the HAPA methodology to set the number of generations used as stop condition on each island, while in Section 6.2 we explain the use of HAPA to define the migration frequency. In both cases, the aim is a meaningful dynamic determination of the parameter values to profit from the differences in the hardware involved to build a more efficient/accurate algorithm. The two algorithms derived from HAPA set one parameter at a time, in order to identify the reasons of possible improvements in their performance.

4 Experimental Setup

In this section we present the necessary information to reproduce the experiments that have been carried out in this article. First we will introduce the problem used to assess the performance of our proposal: the Knapsack Problem (KP), a classical combinatorial optimization problem. In the present study we are not focusing on the solution of a particular problem (many different and specific heuristics exist for this [26]), but our aim is simply to use it to evaluate our proposals. Second, we will justify the parameters that our dGA will use.

4.1 The Knapsack Problem

The Knapsack Problem (KP) belongs to the class of NP-hard problems [16]. Given a knapsack capacity C, and a set N of n items with associated profit $p_i > 0$ and weight $w_i > 0$, the goal is to choose a subset of items such that maximizes the total profit keeping the total weight below the C capacity of the knapsack. We may assume that $w_i < C$, for $i = 1, \ldots, n$ to ensure that each item considered fits into the knapsack, and that the total weight of all the items exceeds C to avoid trivial solutions. The KP can be formulated as an integer programming model as presented in Equation 1, where x_i is the binary decision variable of the problem that indicates whether the item i is included or not in the knapsack.

Table 1. KP instances

Instance	n	R	C	Optimal Profit
KP1-1k	100	1000	1001	9147
KP1-10k	100	10000	10001	81021
KP2-1k	200	1000	1001	11238
KP2-10k	200	10000	10001	106285
KP3-1k	300	1000	1001	13643
KP3-10k	300	10000	10001	129441
KP4-1k	400	1000	1001	15939
KP4-10k	400	10000	10001	141774

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} p_i x_i \\ \text{subject to :} \quad & \sum_{i=1}^{n} w_i x_i \leq C, \\ & x_i \in \{0,1\},\ \forall i = 1,\ldots,n \end{aligned} \tag{1}$$

Four randomly generated data instances are considered as listed in Table 1, with n varying from 100 to 400 items and with two different C capacities (1001 and 10001). These instances were obtained using the generator described in [25] choosing the uncorrelated data instances type, i.e., p_j and w_j which are randomly distributed in $[1,\ldots,R]$ (no correlation between the weight and the profit of an item, in order to make the problem harder). The optimal solution of each instance (reported in Table 1) was found using the Minknap algorithm [24], an exact method based on dynamic programming.

4.2 Parameters

In our experiments, the global pool of solutions of the dGA is set to 512 solutions, which are organized into 8 islands of 64 solutions each. The tentative solutions for the KP are encoded as binary strings. The genetic operators are: binary tournament selection, two point crossover, and bit flip mutation. The crossover and mutation rates are 0.65 and $1/n$ (where n is the length of the solutions), respectively. Proportional selection is used to build up the next population from the set of parent and offspring solutions. The base migration frequency is set to 128 generations. A copy of the best individual of each subpopulation is sent and replaces the worst solution on the target island, only if it is better (only one individual is sent in each exchange). We would like to remind that the communication between islands is entirely asynchronous, so there are no sync points in the migration operation. The topology follows a unidirectional ring communication pattern. Table 2 summarizes the parameters used in the experimentation.

The code was developed using MALLBA [14], a C++ software library fostering rapid prototyping of hybrid and parallel algorithms, running under Linux. The considered hardware resources are the ones shown in Table 3 which are described in the next section.

Due to the stochastic nature of the algorithms, the final results are obtained after averaging the running times of 30 independent runs. A statistical analysis

Table 2. Experimental parameters of all dGAs

Population size	512 individuals
Number of islands	8 islands
Selection of parents	Binary tournament
Recombination	two-point, $pc = 0.65$
Bit mutation	Bit-flip, $pm = 1/n$
Replacement	Rep_better
Migration frequency	128 generations

has been performed in order to provide the results with statistical confidence and, therefore, obtain meaningful conclusions. We use the non-parametric Kruskal-Wallis test, to distinguish meaningful differences between the mean results of all algorithm. We have considered a level of significance of $\alpha = 0.05$, in order to indicate a 95% confidence level in the results.

5 HAPA Phase 1: Know your Platform

We now proceed to explain the first phase of the HAPA methodology (offline phase). It consists of three parts: getting general knowledge on the platform, fine tuning this knowledge for the class of algorithms we are interested in, and a final third part in which we summarize all this knowledge into a mathematical function to be able of designing algorithms based on it. In consequence, Section 5.1 introduces a characterization of the heterogeneous hardware. Section 5.2 shows the scores from a standard scientific benchmarking software to get a general knowledge, while Section 5.3 presents a ranking coming from the actual execution of a traditional dGA, the class of algorithms we are interested in. Once the machine performance has been obtained (i.e. the hardware has been transformed into numerical knowledge), the last Section 5.4 presents how to obtain the relative velocity factor between each processor and the fastest one by applying a mathematical function; this factor is the basis for the application of the HAPA methodology (online phase).

5.1 Hardware Description

The heterogeneous computing system consists of a wide range of diverse CPUs belonging to different families of processors, including single and multicore, single and multithreaded, mono and multiprocessors, 32 and 64 bits, and different processor vendors. The details about our heterogeneous environment are shown in Table 3, where specifications of each node (CPUi) are included, regarding processor, clock speed, memory, number of cores, number of nodes, and release year. From that table we can see that there is a number of commodity commercial computers and the release date of nodes corresponds to a wide range of years. Our heterogeneous computing platform is made up of one machine of each class of processor, except in the case of CPU7, where eight identical machines are included. All these machines are connected by a Gigabit Ethernet. Figure 2 sketches the heterogeneous hardware.

Table 3. Heterogeneous computing environment

Name	Features	RAM(GB)	#cores	#nodes	year
CPU1	AMD Athlon XP2000+ at 1.67 GHz	0.5	1	1	2002
CPU2	Intel Pentium IV at 2.8 GHz	0.5	1	1	2003
CPU3	AMD Athlon XP3000+ at 2 GHz	0.5	1	1	2003
CPU4	AMD Sempron 2800+ at 2 GHz	0.5	1	1	2004
CPU5	AMD Athlon XP3200+ at 2.11 GHz	1	1	1	2005
CPU6	AMD Athlon XP4000+ at 2.11 GHz	0.5	2	1	2006
CPU7	AMD Phenom8450 at 2GHz	2	3	8	2008
CPU8	Intel CI7 2600 at 3.40GHZ	4	4	1	2011

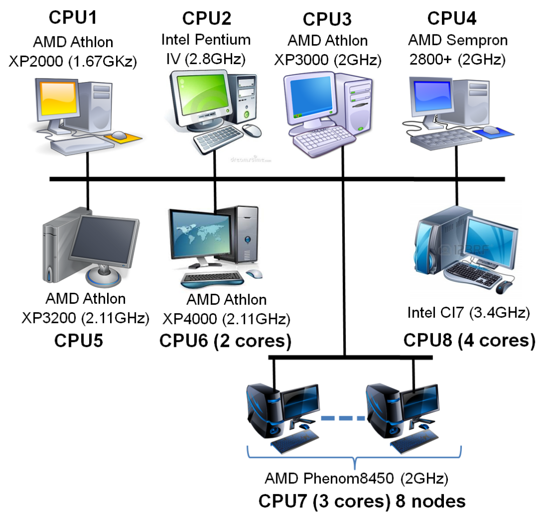


Fig. 2. Heterogeneous computing platform for experiments

5.2 Ranking Using Standard Benchmark Software

We have used scientific benchmarking software to obtain a quantitative measure of the speed of the different processors involved in our heterogeneous platform. The reason behind the use of such benchmarking software is the difficulty to know in advance which class of program will be run in the heterogeneous platform. Particularly, a GA manages many data types: integers (population size, number of generations, alleles, etc.), floats (fitness values, probabilities, alleles, etc.), among others. The different considered data types also depend of the problem to be solved. Consequently, the corresponding compiled program is full of different types of data. These standard benchmarks are very popular and widely used in the professional computer market. So, we first go for them and then, in the second part of this phase, we will try to fine tune the findings got here.

Six different widely-used benchmarking programs have been employed, namely Whetstone [10], Dhrystone version 1 and 2 [28], Livermore Loops [21], and finally Linpack [13] (see [5] for a classification and detailed explanation). The

Table 4. Normalize score value for each benchmark and CPU

	Dhrystone1	Dhrystone 2	Whetstone	Livermore	Linpack	Mean
CPU1	4.40	4.45	2.46	3.52	3.51	**3.67**
CPU2	4.66	4.45	2.48	3.27	2.25	**3.42**
CPU3	3.53	3.69	2.29	3.15	3.16	**3.17**
CPU4	3.66	3.71	2.39	3.11	3.04	**3.18**
CPU5	3.18	3.35	2.05	2.78	2.83	**2.84**
CPU6	3.04	3.22	1.97	2.67	2.83	**2.75**
CPU7	3.90	3.25	2.00	2.57	2.83	**2.91**
CPU8	1.00	1.00	1.00	1.00	1.00	**1.00**

source code can be obtained in Roy Longbottom's PC Benchmark Collection[3]. By running these benchmarking programs on each machine we obtain the scores after an operation that takes a few seconds (less than 10 seconds on average), independently of the benchmark used.

Table 4 presents the score values with respect to the fastest processor (baseline processor). Each benchmark gives different rankings for the processors, but all agree that CPU8 is the fastest one and CPU1 or CPU2 the slowest ones. Due to the diversity in the ranking obtained by each software benchmark, the last column of Table 4 shows the mean normalize score value for each machine. This new value is considered as the final score for each machine in our heterogeneous platform, generating the following global ranking of CPUs: 1, 2, 4, 3, 5, 7, 6, and 8. This ranking confirms our assumptions about the performance of each machine.

5.3 Ranking Using Traditional dGAs

In this section, we present an analysis of the relative velocities of the processors belonging to the heterogeneous computing environment from another point of view: the execution of the dGA for each problem instance. This analysis is based on the fact that a processor with a good score, running a general benchmark, may not be relevant for our metaheuristic, because of its specific kind of operations. The stop condition has been to reach a maximum number of generations (5000 for all the instances) in order to measure the elapsed time for the dGA executed by each processor of Table 3 under the same computational effort. The parameters of the dGA are the ones listed in Table 2.

Table 5 shows the relative performance of the processors regarding the baseline processor (CPU8) for each problem instance. From the analysis of the previous table, we can see that there are important differences in velocities between the considered processors. For example, CPU1 is more than four times slower than CPU8. CPU6 and CPU7 show similar relative velocities and their position in the ranking is hard to differentiate. A machine rank can be established from slow to fast processors: 1, 2, 4, 3, 5, 6/7, 8. Most of the used scientific benchmark software packages rank the processors in a different way than the running time does (let us compare Table 4 and Table 5), but the rank matches the one obtained by averaging the mean scores of the standard benchmark software (last

[3] http://www.roylongbottom.org.uk/

Table 5. Relative velocities between processors discriminated by instances (fastest processor CPU8 as baseline processor). Stop condition: to reach a maximum number of generations

	KP1-1k	KP1-10k	KP2-1k	KP2-10k	KP3-1k	KP3-10k	KP4-1k	KP4-10k	Mean
CPU1	4.40	4.39	4.67	4.68	4.60	4.60	4.75	4.75	**4.61**
CPU2	3.50	3.49	3.58	3.58	3.55	3.56	3.58	3.58	**3.55**
CPU3	2.66	2.64	2.77	2.76	2.78	2.78	2.81	2.82	**2.75**
CPU4	3.20	3.19	3.39	3.39	3.50	3.51	3.60	3.59	**3.42**
CPU5	2.43	2.42	2.54	2.55	2.58	2.58	2.63	2.64	**2.55**
CPU6	2.23	2.23	2.32	2.33	2.36	2.36	2.43	2.43	**2.34**
CPU7	2.25	2.24	2.35	2.34	2.42	2.42	2.45	2.44	**2.36**
CPU8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**

column of Table 4). Thus, we successfully matched the general benchmarks and our particular dGA benchmark into one single common ranking, what offers us a grounded way of thinking in the relative speeds of the processors so as to use it in the design of the algorithms later.

5.4 Mathematical Approximation of Running Times

As mentioned in Section 3, the application of the HAPA methodology requires the computation of the relative velocity factor between machines involved in the heterogeneous platform. Therefore, a comparison between benchmark and run-time results is carried out, by using the mean normalized benchmark scores and the normalized relative velocities shown in the two previous sections. However, it should be impractical to run also (all) the target algorithm to determine the ranking. Therefore, we need a mathematical function to reflect that relation, as a way to predict the running times depending on the node where an island is to be assigned, and to enable us to compute the velocity factor. Also, determining this mathematical function which encapsulates the practical knowledge allows us to extend the use of the methodology to other algorithm applications.

For that purpose (encapsulating the knowledge of the previous experiments into a mathematical tool), we have used the Open Source project *Pythonequations* [4]. *Pythonequations* is a collection of Python equations that can fit themselves to both 2D and 3D data sets (curve fitting and surface fitting).

Among the best fitting equations, we have chosen the Hyperbolic G 2D equation because it was the best one in the consistency with the expected behaviour of the processors, with a reasonably low number of coefficients. The resulting formula is outlined in Equation 2 while the coefficient values and error statistics of the regression are shown in Table 6.

$$f(x) = \frac{a \times x}{b + x} + \frac{c \times x}{d + x} \tag{2}$$

In Equation 2, the x value corresponds to the mean scores obtained by scientific benchmark for each CPUi machine (shown in the last column of Table 4)

[4] `https://code.google.com/p/pyeq2/`

Table 6. Coefficient (left) and fitting statistics (right)

coeff.	value
a	-0.766
b	-4.508
c	1.616
d	1.074

metric	value
R^2	0.9708
Adjusted R^2	0.9489
Root mean squared error	0.1698

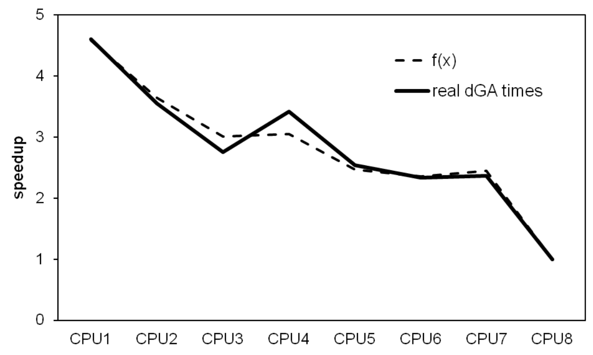


Fig. 3. Real time fitting

and the $f(x)$ value means the mean relative velocities regarding execution times of the dGA for that machine. This last value is important to apply the methodology devised in Section 3, because it is used to compute the *velocity factor* (VF) value which is necessary to set some parameters of every local dGA, such as maximum number of generations or migration frequency.

In Figure 3 we can see the curves representing the $f(x)$ values for each machine calculated by using Equation 2, and the real running times obtained in our tests. Consequently, Equation 2 can be efficiently used to predict the running times of the dGA and the $f(x)$ can be used to compute the VF for a particular machine. For example, if we should need to incorporate a new machine to the computational environment in a future, we should only obtain the scores for each software benchmark, and compute the average. This last value corresponds to the x value in Equation 2. After that, we obtain an approximation of the mean score time regarding execution times of the dGA for that incorporated machine. At this point, it is important to remark that no new runs of the dGA is needed in this new machine. Finally, we only have to compute the VF for that machine in order to be used in the algorithm.

The considered hardware platform is a very heterogeneous cluster (see Table 3), then there is no reason to think that the method (not the concrete results) will not work for other machines, because the same measures could be applied and the same conclusions should be produced: that is the basis of a methodology.

6 HAPA Phase 2: Design your Algorithm

This section is dedicated to the second phase of the methodology, which consists in the design of the dGA fitted to be run in the heterogeneous hardware platform. For that purpose, we describe the two possible instantiations of the proposed methodology in the design of a dGA.

6.1 dGA_HAPA: A First Algorithm Derived from HAPA

In this first algorithm derived from the HAPA methodology, we search for the unification of the completion time of the algorithm on all the islands. For that, the derived dGA establishes in an online way how many generations a subpopulation has to evolve depending on its underlying CPU. If the population in the fastest processor evolves for G number of generations (max_{gens} parameter), each island i of our dGA_HAPA executing in a slower processor B has to set its local number of generations (max_{gens-i}) to a value equal to G divided by $VF(B)$ (the computed relative velocity factor for processor B). In a first step, each island of our dGA derived from the HAPA methodology asks for the features of the processor where it was launched (obtaining the $VF(B)$ value), and also reads the configuration file where the number of generations parameter is generically set to G. With this information it proceeds to locally compute the maximum number of generations. Algorithm 1 shows how a dGA$_i$ node is developed from the HAPA methodology.

The rationale in this design is to avoid big deviations during the search process, usually making just one or two of the islands to produce interesting results, while the rest are stuck in old and lower quality solutions. Heterogeneity could easily produce this behavior, and we want to find out whether our two research questions (accuracy and efficiency) hold in this HAPA design.

Algorithm 1 dGA$_i$ using the HAPA methodology

```
$VF$=get(local Velocity Factor of processor)
$max_{gens-i} = max_{gens}/VF$
$t = 0$; {current generation}
initialize($P_i(t)$);
evaluate($P_i(t)$);
while ($t < max_{gens-i}$) do
    $P'_i(t)$ = evolve($P_i(t)$); {recombination and mutation}
    evaluate ($P'_i(t)$);
    $P'_i(t)$ = send/receive individuals from $dGA_j$;
    $P_i(t+1)$ = select new population from $P'_i(t) \cup P_i(t)$;
    $t = t + 1$;
end while
```

6.2 dGA_HAPA-FM: A Second dGA Derived from HAPA

The proposed HAPA methodology can, of course, be used to derive other algorithms. As a second example, we here now considered the migration frequency in such a way that all islands will finally receive the same number of migrants during the evolution. The aim is to prevent that fast islands probably end their evolution without information from slow islands, also a normal non-desired behavior of plain heterogeneous algorithms.

The same process as the one previously described to compute the maximum number of generations on each island depending on the processor's velocity is carried out (similar pseudocode, thus not shown). If the migration frequency is set to M generations on the fastest island, then this value is divided by the relative velocity factor $VF(B)$ of the rest of processors executing other island GAs. With this operation, the number of generations between consecutive steps of sending/receiving (the migration frequency parameter) is obtained, thus achieving an indirect numerical synchronization between the islands. This implicit synchronization comes from the proportional rate that the islands have between consecutive sending (in that way all islands exchange the same number of individuals during the evolution on average).

7 HAPA Phase 3: Obtaining the Results

This section is devoted to the third phase of the methodology, where the HAPA performance in the developing of dGAs to be run in the heterogeneous hardware platform is evaluated. For that purpose, we consider the two dGAs described in the previous section: dGA_HAPA and dGA_HAPA-FM. The heterogenous environment considered for their execution is made up of one node of each CPUi, with $i \in \{1, \ldots, 8\}$.

For comparison purposes we include the results of traditional dGAs under different homogeneous execution scenarios: ***i***) running a traditional dGA in a concurrent manner, i.e., mapping the eight islands onto only one processor, denominated dGA_1CPUi (CPUi with $i = 1, 2, \ldots, 8$) thus obtaining a dGA variant for each CPU shown in Table 3 and, ***ii***) running a traditional dGA in parallel (dGA_hom) using a parallel homogeneous configuration, where each island is mapped to a ring of eight processors (using the CPU7 hardware configuration). With this comparison we aim at determining whether using HAPA in dGAs has been useful to provide similar and even better results than dGAs running in homogeneous hardware platforms. The stop condition for all algorithms has been established to reach the optimum solution for each instance or the maximum number of generations, whichever happens first.

In what follows, we measure basic parameters such as the hit rate, numerical effort/time to locate a solution, and speedup. The goal is to offer a thorough study of dGAs using HAPA as a way to validating this idea when executing in heterogeneous computers. The section ends with an analysis of the search diversity to examine the behavior of the studied dGAs.

Table 7. Hit rate obtained by dGA_1CPU*i*, dGA_hom and HAPA variants for each problem size

	dGA_1CPUi								dGA_	dGA_	dGA_
	1	2	3	4	5	6	7	8	hom	HAPA	HAPA-FM
KP1-1k	93	83	77	87	83	73	93	80	87	63	80
KP1-10k	97	90	77	93	90	100	100	93	90	93	93
KP2-1k	57	77	70	87	73	73	77	70	70	60	50
KP2-10k	0	0	0	0	0	0	3	0	0	0	0
KP3-1k	13	13	7	0	3	7	7	7	7	7	13
KP3-10k	47	40	27	40	43	50	43	33	27	17	27
KP4-1k	0	3	3	0	3	0	0	3	0	0	3
KP4-10k	0	0	0	0	0	7	0	0	3	0	0

7.1 Hit Rate Analysis

The first considered quality indicator will be the hit rate, i.e., the number of times at which an algorithm finds the optimal solution for an instance, out of a constant number of 30 independent runs. Table 7 displays the obtained results. Let us begin by analyzing the results of the dGA running on one single processor (dGA_1CPU*i*) for all our processors independently. As expected, the different dGA_1CPU*i* present similar hit rate values, which can be explained by the fact that the dGA is exactly the same in all the tests (numerically speaking), being the processor velocity the only difference between them. The dGA running in the homogeneous hardware (dGA_hom), where each island is mapped onto a processor (8 islands and 8 equal processors), also obtains similar results as the different dGA_1CPU*i*, except for KP3-10k whose hit rate is nearly the half of the ones of dGA_1CPU*i*. Finally, the dGAs executed in a heterogeneous computing environment (dGA_HAPA and dGA_HAPA-FM) are able to find the optimal solution in a similar number of runs as those of the different dGA_1CPU*i* and dGA_hom. In general, the hit rate values are higher than the 60% for the three first instances (except for KP2-1k). In the case of instances KP2-10k, KP4-1k, and KP4-10k the dGA was in general not able to find the optimal solution independently of the hardware used, but our actual target is not problem solving, but algorithm design.

7.2 Numerical Effort Analysis

Table 8 shows the numerical effort to locate a solution, i.e., the number of evaluations of the objective function needed to locate the optimum (Table 1). There are no important differences between the results of the dGA_1CPU*i* and dGA_hom. However, an encouraging finding is this: dGA_HAPA variants reach the optimal solutions in a smaller number of evaluations than the rest, except for KP2-1k. The Kruskal-Wallis test confirms this situation, indicating that there are statistically significant differences between groups of ranks (p-values lower than 10^{-5}). There seems to be no statistical difference between the dGA executed in a uniprocessor environment. However, dGA_HAPA-FM presents statistically significant differences with respect to the rest of the algorithms (in its favor) for

Table 8. Numerical effort obtained by dGA_1CPUi, dGA_hom and dGA_HAPA variants for solving each problem size

	dGA_1CPUi								dGA_	dGA_	dGA_
	1	2	3	4	5	6	7	8	hom	HAPA	HAPA-FM
KP1-1k	65459	67646	64835	74102	69062	59774	54932	55964	72635	47739	**35991**
KP1-10k	56188	60197	46273	91385	50131	55569	55569	56493	49282	69662	**42794**
KP2-1k	88733	91411	92270	126219	92788	**67321**	92492	88828	94397	81542	83723
KP2-10k	–	–	–	–	–	–	110439	–	–	–	–
KP3-1k	145552	145599	144567	–	202702	157149	135002	220203	190293	**63336**	67106
KP3-10k	157951	151096	173798	164674	150141	157023	176838	146770	154821	**86918**	99457
KP4-1k	–	106168	93726	–	127657	–	–	252246	–	–	**61282**
KP4-10k	–	–	–	–	–	154351	–	–	**110824**	–	–

KP1-1k, KP1-10k and KP4-1k. In the case of KP2-10k and KP4-10k the optimal solution was too difficult to find for all the algorithms.

In the first two instances, the percentage of numerical effort reduction achieved by dGA_HAPA-FM with respect to dGA_1CPUi and dGA_hom is between 22% (dGA_1CPU3 for KP1-10k) and 61% (dGA_1CPU4 for KP1-10k) and for KP2-1k the values decrease by more than the 50%. For the rest of the instances the values decrease by the 75%. Analyzing the number of evaluations of dGA_HAPA with respect to dGA_1CPUi and dGA_hom for KP2-1k, a reduction of over 30% is achieved, while for KP3-1k and KP3-10k the values decrease by more than the 67%. Consequently, the exploration of the search space made by dGA_HAPA and dGA_HAPA-FM is not the same as for the other algorithms, because the final number of function evaluations is fairly smaller. The truly interesting point is that, even with such a lower effort, the hit rate is highly competitive compared to the ones of the other non-HAPA algorithms analyzed.

All the previous observations, i.e., similarities in hit rate values and numerical effort, represent a promising result, remarking the feasibility of merging different old hardware with new machines as a unified heterogeneous platform to execute a dGA. With these results, we can positively answer the first question (Q1) made in the introduction (Section 1): HAPA can help in designing new algorithms for heterogeneous hardware with competitive times (compared to new hardware) and with better problem-solution accuracy.

7.3 Runtime Analysis

Another interesting measure for a dGA is the total runtime needed to reach a solution, which is shown in Figure 4. In this case, the mean runtime in seconds is displayed. From the point of view of the uniprocessor configurations, we can see that dGA_1CPU8 is the fastest configuration, followed by dGA_1CPU6, dGA_1CPU7, and so on, i.e., the resulting ranking matches the underlying technologies' state of the art, an expected situation. The two HAPA variants are the fastest hardware configurations (lowest execution time). All the differences are statistically significant according to the Kruskal-Wallis test (p-values bellow to 10^{-5}), dGA_HAPA-FM and dGA_hom have ranks significantly different from dGA_HAPA. These results are the basis to answer the second question (Q2)

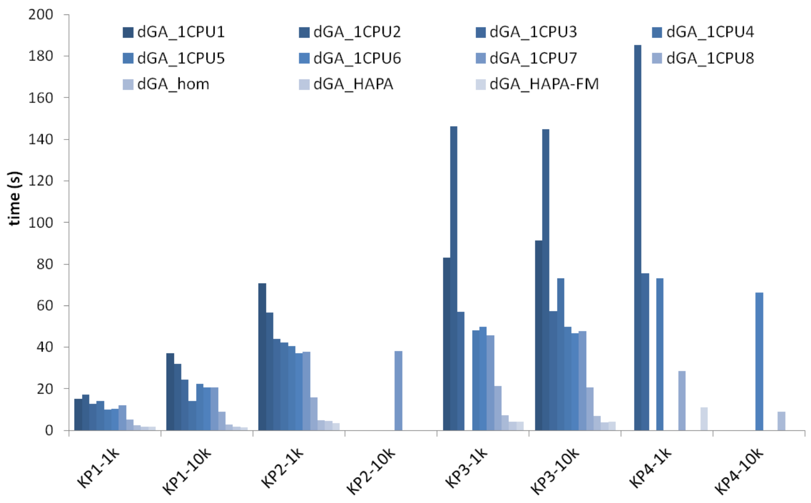


Fig. 4. Execution times obtained by the dGA for each problem size

formulated in Section 1: we do not observe a degradation in the execution time with the use of the heterogeneous computing environment (slower computers on it), which again is an interesting finding. Homogeneous dGAs can be only better, in theory, if all CPUs involved in the computing environment are equal to the fastest hardware configuration belonging to the heterogeneous parallel platform.

Related to the runtime measure, we have also studied the speedup s_m, which compares the run time of the parallel algorithm on one processor against the run time of the same algorithm on m processors to solve a particular problem [1]. For non-deterministic algorithms, the speedup is the ratio between the mean execution time of the dGAs on a uniprocessor configuration, denoted as $E[T_1]$, and the mean execution time of the dGAs on m processors, denoted as $E[T_m]$ with $m = 8$ (see Equation 3). The speedup has been computed here with respect to the fastest processor (CPU8, worst case analysis).

$$s_m = \frac{E[T_1]}{E[T_m]} \tag{3}$$

Figure 5 graphically shows the speedup values for each dGA (the line in this figure expresses the ideal linear speedup) . The dGA_HAPA-FM speedup is the best of the three algorithms for the most of the instances. Although the speedup is sub-linear ($s_m < m$), the heterogeneous results are quite good because they are approximately at 80% of the ideal speedup value, except for instances KP1-1k, KP4-1k and KP4-10k where the values are very small (less than 2.7). This last fact could be explained by the huge difference between the power of the different hardware configurations used: we must remember that the reference point for speedup is the best performing processor, a very fast one in global terms.

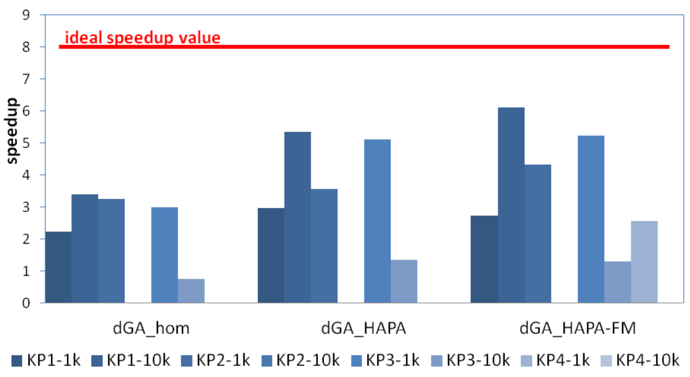
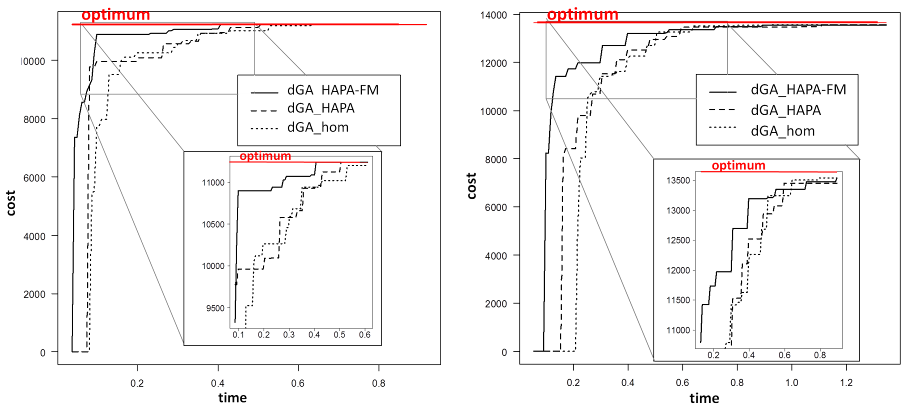


Fig. 5. Speedup obtained by the dGAs for each problem size



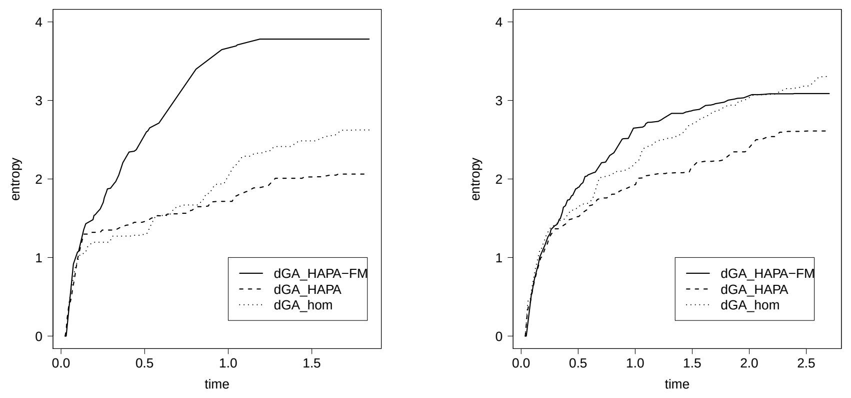
(a) KP2-1k (b) KP3-1k

Fig. 6. Evolution of the mean fitness for two illustrative problems

7.4 Search Diversity Analysis

We proceed now with an analysis of the evolution of the fitness and the diversity of the dGA_hom, dGA_HAPA and dGA_HAPA-FM. For this purpose, we are going to track the value of the best fitness and the mean population entropy along the search, which are the mean of ten runs made for each algorithm. We did just ten runs because it is a fairly stable process, as the results will show.

Figure 6 shows the evolution of the best fitness along the execution of each algorithm for some problem instances used as example (similar situations are observed in the rest of the instances). In the bottom right corner of each subfigure, we make a zoom into detailed moments of the evolution. Regardless of the problem dimension, the population diversity of dGA_HAPA-FM leads to good solutions faster than the rest. This observation suggests that the cooperation of



(a) KP2-1k (b) KP3k-1k

Fig. 7. Evolution of the entropy for two illustrative problems

slow processors and faster processors, together with the adjustment of the migration frequency, lead to work out high quality solutions. After an initial period of evolution, the curves belonging to dGA_HAPA and dGA_hom overlapped to the dGA_HAPA-FM.

Figure 7 presents the evolution of the mean population entropy along the search for each algorithm analyzed (measured for every bit position in all the individuals of a population). In this case, we show the accumulated entropy values over time. We are going to analyze this figure and compare values with the aforementioned evolution of the fitness. We can see how the fitness curve has a maintained growth rate for the dGA_HAPA-FM population, suggesting that it was able to sustain a higher diversity within its population during the whole search, which helps to produce good solutions. The algorithm dGA_hom reaches good entropy levels compared to the dGA_HAPA one.

Another factor to analyse is the quality of immigrants with respect to the best solution in the target population. Our aim is to give a measure of the degree of cooperation between the different subpopulations of an algorithm. We compute the percentage of times the incoming solution has a higher quality than the best solution in the target population. Table 9 shows the mean percentages for all dGAs and problem instances. In general, the percentages decrease as the complexity of the instances increases. The dGA_hom algorithm obtains the higher percentages than the rest, a typical excess of elitism that renders unproductive the algorithm. In most of the cases, the dGA_HAPA-FM has a higher percentage of acceptance than the dGA_HAPA, meaning that the received solution is incorporated to the target population. In a way, previous observations indicate the beneficial influence of collaboration between the subpopulations in different stages of the evolution. The exchange of solutions not only benefits slower islands (by the reception of optimized solutions) but also benefits faster islands because of solutions having portions of the optimum in them.

Table 9. Percentage of acceptance of the received solutions into de local subpopulation

%	dGA_hom	dGA_HAPA	dGA_HAPA-FM
KP1-1k	94.48	80.22	90.03
KP1-10k	91.15	71.90	84.90
KP2-1k	86.59	76.56	79.26
KP2-10k	82.01	78.54	73.43
KP3-1k	83.56	71.07	72.98
KP3-10k	82.19	73.63	73.95
KP4-1k	76.31	70.25	70.11
KP4-10k	75.54	67.77	62.20
Mean	83.60	73.74	75.86

8 Conclusions and Future Research

This article deals with the execution of a dGA using heterogeneous computing resources, where the processing nodes show a high level of heterogeneity: different CPUs belonging to a wide range of fabrication years and technologies. We developed a methodology, called HAPA, to deal with that heterogeneity and the execution of metaheuristics. HAPA consists of three phases: ***i***) the computation of a ranking of processors in order to know the platform (an offline phase), ***ii***) the algorithm design derived from the previous phase, and ***iii***) the validation of the proposed dGAs (an online phase). In this work, the HAPA methodology was applied to regulate the stopping conditions (dGA_HAPA algorithm) and the migration frequency (dGA_HAPA-MF algorithm).

We have performed a set of tests in order to assess the performance of our proposal, and we compared both algorithms derived from the HAPA methodology against a dGA running in a homogeneous computing environment. The results indicate that similar levels of accuracy and efficiency can be attained, but with a lower number of function evaluations, by using the proposed HAPA methodology, thus confirming RQ1. We have shown that the dGA_HAPA-MF algorithm can perform a search in a faster way than the homogeneous dGAs, while maintaining a higher diversity within the population, exhibiting a better balance between exploration and exploitation. Therefore, we can also finally confirm RQ2.

In short, we have contributed in this article with a way of avoiding pure ad-hoc design of algorithms for heterogeneous platforms, as well as we give a new line of research in how to inject hardware knowledge into software parameters of the algorithms. In addition of being innovative, this has shown to also be numerically competitive and within reduced run times.

Further research is necessary to understand the effects of factors such as the influence in the topology of communication between islands. We will also consider a self-control of the parameters of the distributed genetic algorithm during the evolution under a heterogeneous environment taking care of both, numerical and hardware issues.

Acknowledgments

We acknowledge the UNLPam, the ANPCYT, CONICET and PICTO-UNLPam-0278 in Argentina from which Dr. Salto receives regular support. The work of Prof. Alba has been partially funded by the University of Málaga UMA/FEDER FC14-TIC36, programa de fortalecimiento de las capacidades de I+D+I en las universidades 2014-2015, de la Consejería y Economía, Innovación, Ciencia y Empleo, with European FEDER, and also by the UMA Project 8.06/5.47.4142 with the VSB-Technical University of Ostrava (CR). Finally, we acknowledge the funding by the Spanish MINECO project TIN2014-57341-R (http://moveon.lcc.uma.es).

References

1. E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters, Elsevier*, 82(1):7–13, 2002.
2. E. Alba. *Parallel Metaheuristics: A New Class of Algorithms.* Wiley, 2005.
3. E. Alba, A.J. Nebro, and J.M. Troya. Heterogeneous computing and parallel genetic algorithms. *Journal of Parallel and Distributed Computing*, 62:1362–1385, 2002.
4. E. Alba and J.M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17(4):451–465, 2001.
5. J. Dominguez E. Alba. A methodology for comparing the execution time of metaheuristics running on different hardware. In Jin-Kao Hao and Martin Middendorf, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 7245 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2012.
6. J. Baugh and S. Kumar. Asynchronous genetic algorithms for heterogeneous networks using coarse-grained dataflow. In *Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartI*, GECCO'03, pages 730–741, Berlin, Heidelberg, 2003. Springer-Verlag.
7. Victor E. Bazterra, Martin Cuma, Marta B. Ferraro, and Julio C. Facelli. A general framework to understand parallel performance in heterogeneous clusters: analysis of a new adaptive parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, 65(1):48 – 57, 2005.
8. Jrgen Branke, Andreas Kamper, and Hartmut Schmeck. Distribution of evolutionary algorithms in heterogeneous networks. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation GECCO 2004*, volume 3102 of *Lecture Notes in Computer Science*, pages 923–934. Springer Berlin Heidelberg, 2004.
9. F. Chong. Java based distributed genetic programming on the internet. Technical report, School of Computer Science, University of Birmingham, 1999.
10. H.J. Curnow and B.A. Wichmann. A synthetic benchmark. *The Computer Journal*, 19(1):43–49, 1976.
11. J. Dominguez and E. Alba. Ethane: A heterogeneous parallel search algorithm for heterogeneous platforms. *DECIE11*, 2011.
12. J. Dominguez and E. Alba. Dealing with hardware heterogeneity: a new parallel search model. *Natural Computing*, 12(2):179–193, 2013.
13. J. Dongarra. Performance of various computers using standard linear equations software in a fortran environment. *Simulation*, 49(2):51–62, 1987.

14. E. Alba et al. *MALLBA: A Library of Skeletons for Combinatorial Optimisation*, volume 2400 of *LNCS*, pages 927–932. Springer, 2002.
15. M. García-Arenas, J. Merelo, P. Castillo, J. Laredo, G. Romero, and A. Mora. Using free cloud storage services for distributed evolutionary algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1603–1610, New York, NY, USA, 2011. ACM.
16. M.R. Garey and D.S. Johnson. *COmputers and Intractability: a Guide to the Theory of NP-Completeness.* Freeman, 1979.
17. Y. Gong, M. Nakamura, and S. Tamaki. Parallel genetic algorithms on line topology of heterogeneous computing resources. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 1447–1454, 2005.
18. Dudy Lim, Yew-Soon Ong, Yaochu Jin, Bernhard Sendhoff, and Bu-Sung Lee. Efficient hierarchical parallel genetic algorithms using grid computing. *Future Generation Computer Systems*, 23(4):658 – 670, 2007.
19. P. Liu, F. Lau, andd J. Lewis, and C. Wang. Asynchronous parallel evolutionary algorithm for function optimization. In *Parallel Problem Solving from Nature*, page 405409. Springer, 2002.
20. G. Luque and E. Alba. *Parallel Genetic Algorithms: Theory and Real World Applications*, volume 367 of *Studies in Computational Intelligence.* Springer, 2011.
21. F.H. McMahon. *The Livermore Fortran Kernels: A Computer Test Of The Numerical Performance Range.* Lawrence Livermore National Laboratory, 1986.
22. K. Meri, M. Arenas, A. Mora, J. Merelo, P. Castillo, P. Garca-Snchez, and J. Laredo. Cloud-based evolutionary algorithms: An algorithmic study. *Natural Computing*, 12(2):135–147, 2013.
23. S. Mostaghim, J. Branke, A. Lewis, and H. Schmeck. Parallel multi-objective optimization using master-slave model on heterogeneous resources. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1981 –1987, 2008.
24. D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.
25. D. Pisinger. Core problems in knapsack algorithms. *Operations Research*, 47:570–575, 1999.
26. D. Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32:2271–2282, 2005.
27. R. Tanese. Distributed genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–439, 1989.
28. R.P. Weicker. Dhrystone: a synthetic systems programming benchmark. *Communications of the ACM*, 27(10):1013 – 1030, 1984.