Noname manuscript No.
(will be inserted by the editor)

Influence of population size in distributed Evolutionary Algorithms in homogeneous and heterogeneous clusters

P. García-Sánchez · M. G. Arenas · P. A. Castillo · C. Fernandes · J. González · A. M. Mora · J. J. Merelo

Received: date / Accepted: date

Abstract This paper presents a study on population size tuning in a distributed Evolutionary Algorithm. The proposed adaptation strategy is done taking into account the computational power of each node of an heterogeneous cluster. The same population sizes are also tested in an homogeneous scientific cluster. Two problems with different characteristics have been tested. Results show that setting the population size according to the computational power decreases the time required to obtain the optimum in both problems when using heterogeneous clusters. Also, a study of the influence of the different population sizes in each stage of the algorithm is presented.

Keywords Evolutionary Algorithms · Genetic Algorithms · Service Oriented Architecture · Heterogeneous computation · Distributed computing

1 Introduction

New trends in distributed computing such as Cloud Computing [6], GRID [3] or Service Oriented Science [11] are leading to heterogeneous computational devices, for instance laptops, tablets or desktop PCs, working in the same environment. Thus, many laboratories, which do not count with classic clusters but the usual workstations used by scientists, can leverage this motley set as an heterogeneous cluster. In fact, distributed Evolutionary Algorithms (dEAs) [4] have been tested successfully in these systems.

P. García-Sánchez
Dept. of Computer Architecture and Computer Technology, E.T.S. Ing. Informática y Telecomunicación and CITIC-UGR
University of Granada, Granada, Spain
E-mail: pgarcia@atc.ugr.es

Evolutionary Algorithms (EAs) are a general technique for solving optimization and search problems based on the evolution of species and natural selection. These algorithms are formed by a population of possible solutions (called *individuals*) that competes using their *fitness* (quality of adaptation) with the rest of solutions. In each iteration of the algorithm (or *generation*) the individuals are evolved by means of selection and recombination/mutation to create a new set of candidates, until a *stop criterion* (i.e. number of generations) is met. Fitness function is a quality function that gives the grade of adaptation of an individual respect the others. This function is usually the problem to solve.

There are different ways to parallelize the EAs, being the most extended:

– *Farming model (centralized EAs)*: A central node coordinate several slave nodes. The central node executes the EA in a sequential way, but distributes the individuals of the population to the slaves just for being evaluated. An example can be seen in [18], where slave nodes evaluates fitness function for simullation of nuclear devices.
– *Island model (distributed EAs)*: A number of nodes executes simultaneously the EA, working with different sub-populations at the same time. Each certain number of generations is interchanged (migrated) between populations. Figure 1 shows this model with a ring topology.
– *Cellular EAs (fine grain EAs)*: Each node has one individual of the population, and selection and reproduction is limited with the individuals of the neighbourhood of the node [9]. Usually a bi-dimensional grid is used for topology.

These distributed EAs are very popular because the implementation is not complicated and they exploit a

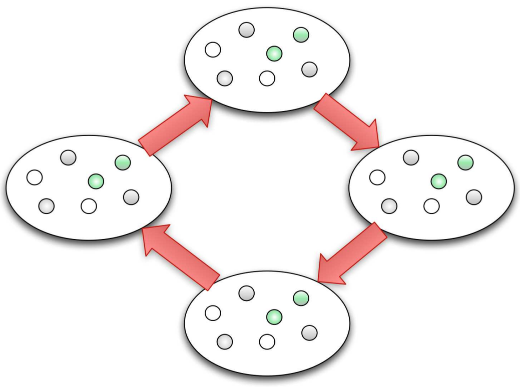

Fig. 1 Island model using a ring topology.

coarse grain parallelism with sporadic communications, being fit to be executed in distributed architectures such as clusters or GRIDs [20].

Heterogeneous dEAs that can be used over these ad-hoc networks can be divided into two categories: dEAs with different parameter settings in each node (heterogeneous parameters), or dEAs running the same algorithm in heterogeneous nodes. It has also been proved [2] that this type of algorithms are even more efficient in heterogeneous hardware configurations than in homogeneous devices. This can be explained by different reasons, such as different memory access times, cache sizes, or even implementation languages or compilers in each machine, leading to a different exploitation/exploration rate of the search space. Heterogeneous parameters configuration has also been proved more efficient time-wise than a fixed set of parameters for different problems [16].

These proofs have motivated us to propose in this paper a combination of both ideas: dEAs in heterogeneous hardware with parameter adaptation. In this study, the parameter to adapt to the computational power of each node has been the population size of each island.

In this work, a distributed system has been developed to solve the following questions:

– Can a distributed EA be adapted to leverage the capability of an heterogeneous cluster?
– What is the effect of adapting the population size to the computational power, as proposed in this paper?
– Is there any difference between the same populations in an homogeneous or heterogeneous cluster?
– How is each stage of the algorithm (selection, migration...) affected by the different configurations?

The rest of the work is structured as follows: after a presentation of the state of the art in the parameter adaptation and load-balancing in dEAs , we present the developed algorithms and experimental setting (Sections 3 and 4). Then, the results of the experiments are shown (Section 5), followed by conclusions and suggestions for future work lines.

2 State of the art

In the field of Evolutionary Computation (EC) there are two different approaches about the algorithm parameter setting: *parameter control* and *parameter tuning* [10]. The first one refers to setting up a number of parameters of an Evolutionary Algorithm (EA) and changing these parameters in running time. The parameter tuning consist in establishing a good set of parameters before the run (and do not change them during runtime).

Computational performance of nodes or network speed can also be inherent parameters of an algorithm. In [2] the authors compared a distributed Genetic Algorithm (dGA), one of the sub-types of EAs, in homogeneous and heterogeneous clusters. Super-linear performance was obtained in the heterogeneous ones, being more efficient that the same algorithm running in homogeneous machines. Some authors have expanded this idea by adapting the algorithm to be executed: in [8] the authors presented a distributed hybrid meta-heuristic that combines two different EAs: Genetic Algorithms (GAs) and Simulated Annealing (SA). Their system executes the heavy (in computational terms) algorithms (GAs) in faster nodes, and simpler meta-heuristics (SA) in slower nodes, obtaining better results than other configurations. In [17] different configurations of heterogeneous machines for a tree topology were studied. However, the heterogeneity was simulated in an homogeneous cluster with programs to add computational load. Load-balancing was also applied taking into account the computational load of the nodes in [13]: a small benchmark was executed in all nodes at the beginning of the algorithm in order to distribute individuals of an Evolutionary Strategy (ES). However, there was no communication between the nodes.

In the area of heterogeneous parameters, but homogeneous hardware, setting a random set of parameters in each node can also increase the performance of a distributed Genetic Algorithm, as explained in [16]. That model outperformed a tuned canonical dGA with the same parameter values in all islands. Finally, adapting the migration rate has produced better results than homogeneous periods in homogeneous clusters, as explained in [21].

Our work presents a combination of some of the previous ideas: an initial parameter tuning given by the computational power of each machines is performed and compared in different systems. To our knowledge, there are not works that modify parameters of the EA depending of the node where the island is being executed.

3 Service Oriented Evolutionary Algorithms

As discussed in [14] the evolutionary algorithms research area is a propitious environment to migrate to SOA for several reasons: SOA fits with the genericity advantages in the development of software for EAs [12] and adds new features, such as language independence and distribution mechanisms. Moreover, there are a wide number of frameworks for EAs mostly incompatible with others, due to different programming languages, operating systems or communication protocols (see [19] for a survey). In addition, new research trends, like self-adaptation [5], require many changes and modifications in the algorithms behaviour in real-time. And finally, the increase of technologies such as GRID and Cloud Computing [6], where the computation elements are distributed in different machines, with many operating systems and programming languages.

In order to deal with the operating system and architecture heterogeneity, the OSGiLiath framework [14], based in Java, has been used in this work. This is a service-oriented evolutionary framework that automatically configures the services to be used in a local network. In this case, each node offers a migration buffer to accept foreign individuals. Also, in order to reduce bottlenecks in distributed executions, asynchronous communication has been provided to avoid idle time using reception buffers (that is, the algorithm does not wait until new individuals arrive, but the buffers cannot be used until again until the reception is done). This kind of communication offers an excellent performance when working with different nodes and operating systems, as demonstrated in [2]. The transmission mechanism is based in ECF Generic server (over TCP)[1]. The source code of the algorithms used in this work is available in `http://www.osgiliath.org` under a GPL V3 License.

4 Experimental setup

This section presents the parameters and platforms to conduce the experiments. Four configurations have been tested (each one has been run 30 times):

– HoSi/HoHa: Homogeneous Size/Homogeneous Hardware. The same population size in each island in an homogeneous cluster.

[1] `http://www.eclipse.org/ecf/`

```
population ← initializePopulation()
while stop criterion not met do
    parents ← selection(population)
    offspring ← recombination(parents)
    offspring ← mutation(offspring)
    population ← population + offspring
    if time to migrate then
        migrants ← selectMigrants(population)
        remoteBuffer.send(migrants)
    end if
    if localBuffer.size ≠ zero then
        population ← population + localBuffer.read()
    end if
    population ← removeWorst(population)
end while
```

Fig. 2 Pseudo-code of the used dEA: a distributed Genetic Algorithm (dEA).

– HoSi/HeHa: Homogeneous Size/Heterogeneous Hardware. The same population size in each island in an heterogeneous cluster.
– HeSi/HoHa: Heterogeneous Size/Homogeneous Hardware. Different population sizes in each island in an homogeneous cluster.
– HeSi/HeHa: Heterogeneous Size/Heterogeneous Hardware. Different population sizes in each island in an heterogeneous cluster.

Two different computational systems have been used: an *heterogeneous cluster* and an *homogeneous cluster*. The first one is formed by four different computers of our lab with different processors, operating systems and memory size. The latter is a dedicated scientific cluster formed by homogeneous nodes. Table 1 shows the features of each system.

The EA to study is a dGA. Figure 2 shows the pseudo-code of the used algorithm. Parameters are described in Table 2. The algorithm is steady-state, i.e. the offspring is mixed with the parents and the worst individuals are removed. The used neighborhood topology between islands (nodes) is a ring (see Figure 1. The best individual is sent after a fixed number of generations in each node (64).

The problems to evaluate are the Massively Multimodal Deceptive Problem (MMDP) [15] and the OneMax problem [22]. Each one requires different actions/abilities by the GA at the level of population sizing, individual selection and building-blocks mixing. The MMDP is designed to be difficult for an EA, due to its multimodality and deceptiveness. Deceptive problems are functions where low-order building-blocks do not combine to form higher order building-blocks. Instead, low-order building-blocks may mislead the search towards local optima, thus challenging search mechanisms. MMDP it is composed of k subproblems of 6

Table 1 Details of the clusters used.

Name	Processor	Memory	Operating System	Network
Homogeneous cluster				
HoN[1-4]	Intel(R) Xeon(R) CPU E5320 @ 1.86GHz	4GB	CentOS 6.7	Gigabit Ethernet
Heterogeneous cluster				
HeN1	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	4GB	Ubuntu 11.10 (64 bits)	Gigabit Ethernet
HeN2	Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz	4GB	Ubuntu 11.04 (64 bits)	Gigabit Ethernet
HeN3	AMD Phenom(tm) 9950 Quad-Core Processor @ 1.30Ghz	3GB	Ubuntu 10.10 (32 bits)	100MB Ethernet
HeN4	Intel (R) Pentium 3 @ 800MHz	768 MB	Ubuntu 10.10 (32 bits)	10MB Ethernet

Table 2 Parameters used.

Name	Value
Total individuals	256
Population size in HoSi	64
Population size in HeSi	98, 84, 66, and 8
Crossover type	Uniform crossover
Crossover rate	0.5
Mutation rate	1/individual size
Selection	2-tournament
Replacement	Steady-state
Generations to migrate	64
Individual size for MMDP	150
Individual size for OneMax	5000
Runs per configuration	30

Table 3 Basic deceptive bipolar function (s_i) for MMDP.

Unitation	Subfunction value
0	1.000000
1	0.000000
2	0.360384
3	0.640576
4	0.360384
5	0.000000
6	1.000000

bits each one (s_i). Depending of the number of ones (unitation) s_i takes the values shown in Table 4.

The fitness value is defined as the sum of the s_i subproblems with an optimum of k (Equation 1). The search space is composed of 2^{6k} combinations from which there are only 2^k global solutions with 22^k deceptive attractors. Hence, a search method will have to find a global solution out of 2^{5k} additionally to deceptiveness. In this work $k = 25$.

$$f_{MMDP}(\mathbf{s}) = \sum_{i=1}^{k} fitness_{s_i} \tag{1}$$

OneMax is a simple linear problem that consists in maximising the number of ones in a binary string. That is, maximize the expression:

$$f_{OneMax}(\mathbf{x}) = \sum_{i=1}^{N} x_i \tag{2}$$

To determine the computational power of the heterogeneous machines we have compared the average number of generations obtained in the HoSi/HeHa configuration for the MMDP problem (after the 30 runs). This comparison takes into account all the evolutionary process in a fair manner (proportional to the memory, processor and network usage), instead a traditional benchmark that usually relies only the CPU speed. This is a possible way to establish the computational power for the experiments in this work and determine if changing the population size according the computational power reduces the time of the whole system.

Thus, after executing the HoSi/HeHa we have used the obtained results to set the sizes in the HeSi/HeHa and HeSi/HoHa configurations: 98, 84, 66, and 8 individuals (from node HeN1 to node HeN4; and from HoN1 to HoN4). Note that, having two nodes with the same processors and memory (HeN1 and HeN2), they have different computational power: this might be produced by different operating systems, virtual machine versions, or number of processes being executed.

5 Results

The main objetives of parallel programming are to tackle large computational problems, increase the performance of algorithms in a finite time, or reduce computational time to solve the problem. In this work we focus in the last objective. As claimed by the authors in [1], assessing the performance of a parallel EA by the number of function evaluations required to attain a solution may be misleading. In our case, for example, the evaluation time is different in each node of the heterogeneous cluster, so the real algorithm speed could not be reflected correctly. However, the number of evaluations has been included in this section to better understand the results. The total number of generations, and the maximum number of generations required by the slower node in each configuration are also shown. It is difficult to compare the performance of HoHa and HeHa for the same reason: the evaluation time is different in each system (and even in each node).

5.1 MMDP results

Table 4 shows the results for the MMDP problem. These results are also shown in the boxplots of Figure 3 (time)

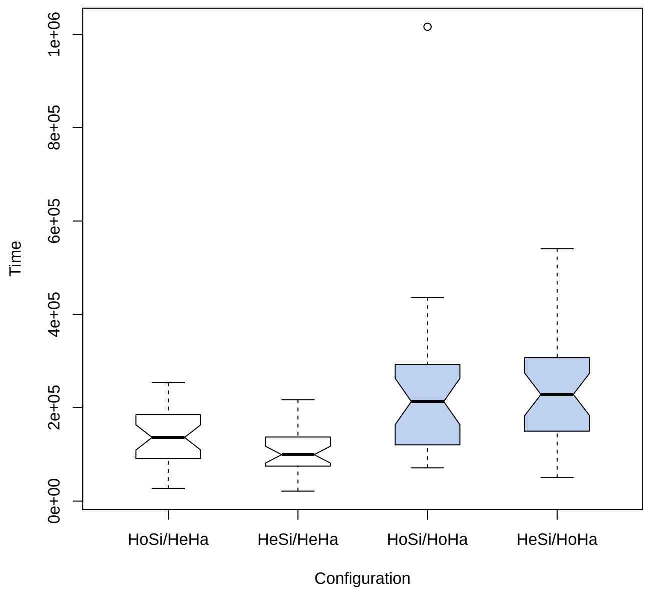

Fig. 3 Time to obtain the optimum in the MMDP problem (milliseconds). White boxplots correspond to the heterogeneous cluster and gray ones to the homogeneous.

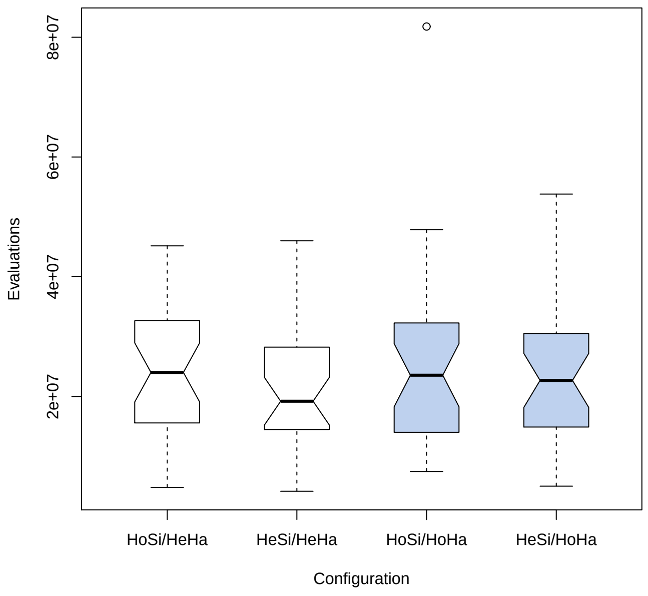

Fig. 4 Number of evaluations for MMDP problem. White boxplots correspond to the heterogeneous cluster and gray ones to the homogeneous.

and Figure 4 (evaluations). Table 6 shows the statistical significance of the results. First, a Kolmogorov-Smirnov test is performed to assess the normality of the distributions. If the results fit a normal distribution, then a Student's T-Test is calculated. Otherwise, the non-parametric test Wilcoxon signed rank is applied (see [7] for a tutorial for comparing EAs).

In the HeHa system, adapting the population to the computational power of each node makes the algorithm finish significantly earlier, but with the same number of evaluations (with no statistical significance). This can be explained because the evaluation time is different in all nodes. On the other hand, in the HoHa system, setting the same population sizes makes no difference in time and evaluations, that is, changing this parameter has no influence in the algorithm's performance.

To see the difference of how the evolution is being performed, the average fitness in each node of HeHa is shown in Figures 5 and 6. As it can be seen, with the HeSi (Figure 6), the local optima are overtaken in less time than HoSi (Figure 5). This can be explained because in HeSi, the migration from HeN4 to HeN1 is performed faster, adding more heterogeneity to the whole system. Gaps in the figures correspond to the time while the nodes are sending the migrant individual to other nodes (not while they are receiving them). In the HoHa systems, the populations are evolved at the same time, being the average fitness similar in all nodes during all run.

5.2 OneMax results

Results for this problem are shown in Table 5 and Figures 7 and 8. In this case, adapting the population sizes significantly decreases the running time for solving in the heterogeneous cluster, and as before, the number of evaluations remains the same (see statistical significance in Table 6). In the homogeneous system, the effect of changing the population sizes is clearer, and this time the number of evaluations (and therefore, the time) are reduced (both significantly).

The efficiency on OneMax problem depends mainly on the ability to mix the building-blocks, and less on the genetic diversity and size of the population (as with MMDP). No genetic diversity is particularly required. When properly tuned, a simple Genetic Algorithm is able to solve OneMax in linear time. Sometimes, problems like OneMax are used as control functions, in order to check if very efficient algorithms on hard functions fail on easier ones. As it can be seen in Figure 9, the average fitness of all populations are increasing in linear way in the HoSi/HeHa configuration. However, the slower node evaluates extremely fewer times. On the other side, in Figure 10, smaller population sizes make that slower nodes increase the number of evalua-

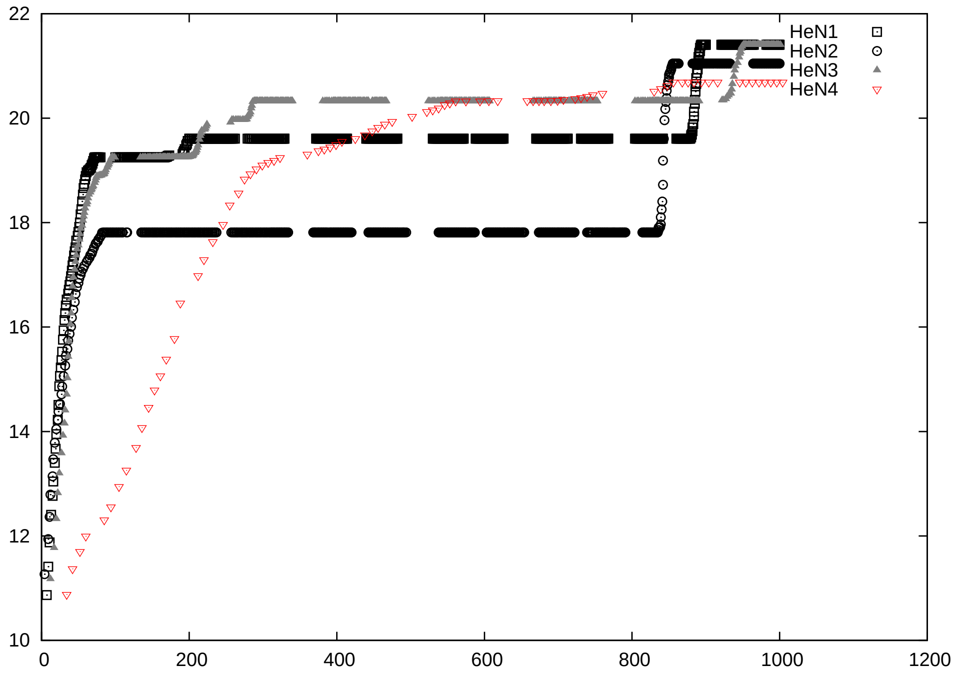

Fig. 5 Average fitness in the first 1000 milliseconds of execution of the four nodes of the heterogeneous cluster with the same population sizes (HoSi/HeHa) for the MMDP problem.

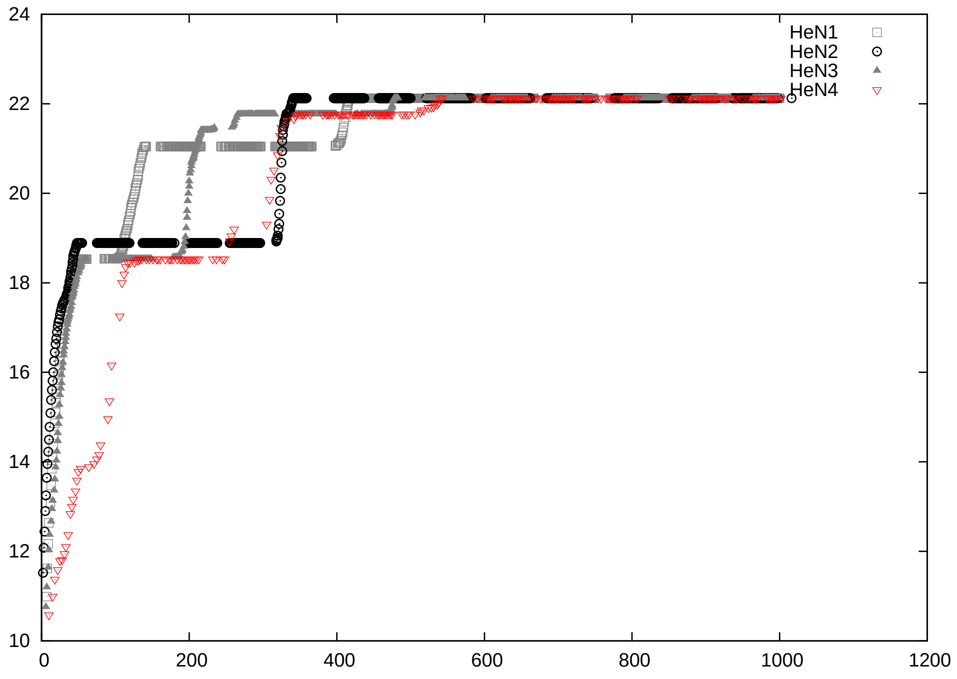

Fig. 6 Average fitness in the first 1000 milliseconds of execution of the four nodes of the heterogeneous cluster with different population sizes (HeSi/HeHa) for the MMDP problem.

Table 4 Results for the MMDP problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	146401,48 ± 65699,69	380967,25 ± 168568,84	24382416,51 ± 10788405,87	136914,03 ± 60028,48
HeSi/HeHa	96051,5 ± 45110,90	289282,3 ± 135038,10	21784528,66 ± 10161989,38	109875,76 ± 49185,51
HoSi/HoHa	107334,46 ± 78167,19	393119,86 ± 241835,27	25273201,06 ± 15386663,12	237759,43 ± 178709,86
HeSi/HoHa	149732,6 ± 81983,74	438171,16 ± 240169,19	24430043,46 ± 13395037,34	245776,93 ± 134715,52

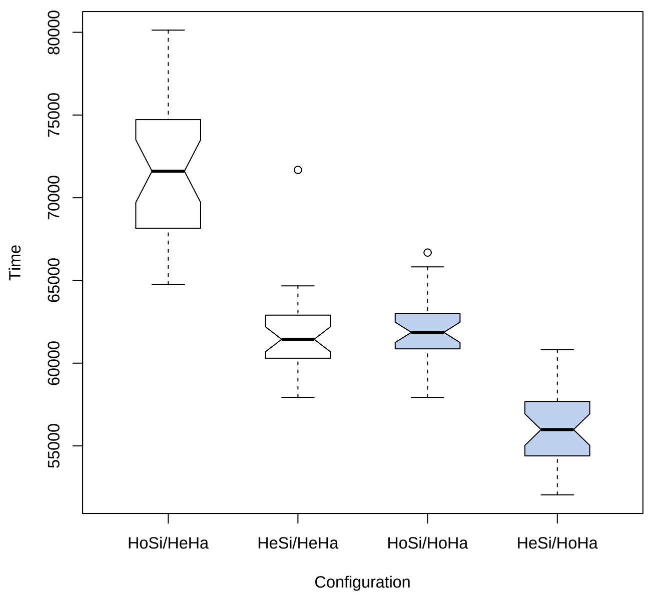

Fig. 7 Time to obtain the optimum in the OneMax problem (milliseconds). White boxplots correspond to the heterogeneous cluster and gray ones to the homogeneous.

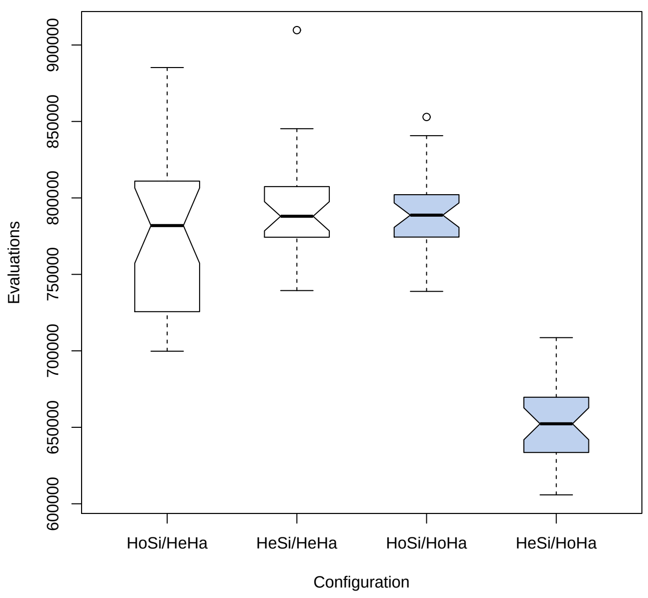

Fig. 8 Number of evaluations for OneMax problem. White boxplots correspond to the heterogeneous cluster and gray ones to the homogeneous.

tions, but the average fitness is also maintained in linear way (and in smaller increase rate) between migrations. However, the other nodes still perform a higher number of evaluations. That is the reason why the number of evaluations is higher in HeHa, and lower in HoHa. Computational time is more efficiently spent in faster nodes, having a higher chance to cross the individuals. In addition, due to the larger size of individuals in the OneMax problem (5000 bits vs. 150 of the MMDP), the transmission time is larger, (white gaps in the figures). It also implies that HeN4 sends its best individual to HeN1 in an extremely large amount of time when using HoSi (every 64 generations).

5.3 Running time analysis

This sub-section analyses the time spent by each node of the clusters in every stage of the EA for each configuration. Tables 7 and 8 show the average and standard deviation of the time spent in each stage of the algorithm (He=Heterogeneous cluster, Ho=Homogeneous cluster). Figures 11 and 12 graphically compare these results. As it can be seen, the migration is the most time consuming operation in all configurations, being the migration in HeHa more expensive than in HoHa. This happens because we are using the multi-purpose laboratory network to communicate the nodes, instead of the specific one used in the HoHa system. Note that the std. deviation of the migration is larger in the HeHa cluster. In the MMDP problem (Table 7) changing the population size does not affect the migration time, but it affects the rest of the algorithm's stages. However, with larger data communications (individuals of 5000 elements of the OneMax problem), the population size affects the migration time of all nodes. This might be due to the synchronization of migration buffers: if the slowest machine is sending/receiving, deadlocks can be propagated (as it can seen in Figure 9).

Results also show how the stages of the algorithms depends on the node of execution. For example, recombination needs more time than mutation in both prob-

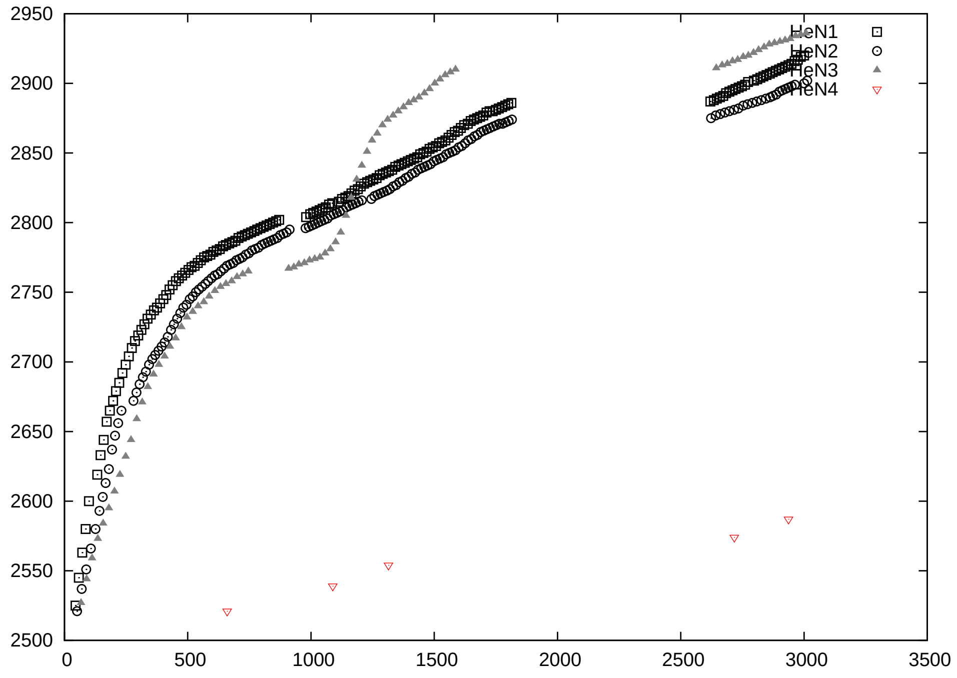

Fig. 9 Average fitness in the first 3000 milliseconds of execution of the four nodes of the heterogeneous cluster with the same population sizes (HoSi/HeHa) for the OneMax problem.

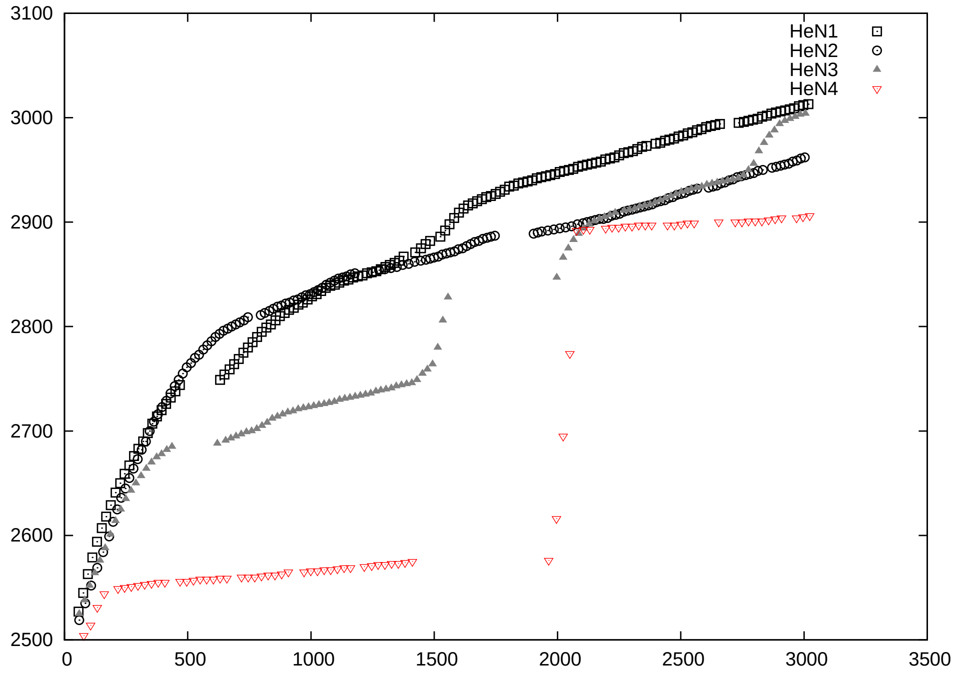

Fig. 10 Average fitness in the first 3000 milliseconds of execution of the four nodes of the heterogeneous cluster with different population sizes (HeSi/HeHa) for the OneMax problem.

Table 5 Results for the OneMax problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	4739,41± 305,32	12081,51± 776,35	773729,03± 49686,72	72152,32± 4994,71
HeSi/HeHa	3438,03 ± 149,47	11277,33± 471,77	794157,73± 31843,10	61870,2 ± 2518,74
HoSi/HoHa	3133,36± 101,70	12347,83± 394,99	790773,33± 25279,52	62105,03± 1964,75
HeSi/HoHa	13897,86± 625,27	20725,63± 929,43	651952,8 ± 29114,54	56120,53± 2491,92

Table 6 Statistical significance of the results.

Configuration	Normal	Test applied	P-value	Significant difference?
Time for MMDP				
HoSi/HeHa vs HeSi/HeHa	Yes	T-Test	0.032	Yes
HoSi/HoHa vs HeSi/HoHa	No	Wilcoxon	0.567	No
Evaluations for MMDP				
HoSi/HeHa vs HeSi/HeHa	Yes	T-Test	0.231	No
HoSi/HoHa vs HeSi/HoHa	No	Wilcoxon	0.958	No
Time for OneMax				
HoSi/HeHa vs HeSi/HeHa	Yes	T-Test	9×10^{-15}	Yes
HoSi/HoHa vs HeSi/HoHa	No	Wilcoxon	1×10^{-6}	Yes
Evaluations for OneMax				
HoSi/HeHa vs HeSi/HeHa	No	Wilcoxon	0.14	No
HoSi/HoHa vs HeSi/HoHa	Yes	T-Test	2×10^{-27}	Yes

lems only in the node HeN4. The reason might be the creation of new objects (memory allocation), which in Java and in limited memory (and SWAP access) requires more time than iteration of elements previously created (for example, in the mutation). Adapting the population size makes the slower node of HeHa behaves in similar way than the other nodes (same time in each stage). Moreover, the size of the individuals affects some parts of the EA; for example, in the OneMax the mutation requires more time than the replacement. However, it must be taken into account that the duration of each part of the algorithm is not related with the time to attain the optimum, but to how the diversity and search guidance is maintained in the whole system.

6 Conclusions

In this paper we have performed a study about adapting the population size of an Evolutionary Algorithm to the computational power of different nodes in an heterogeneous cluster. To obtain a fair parameter configuration, this parameter has been obtained proportionally to the attained average generations of each node in executions with the same number of individuals. Results show that adapting the population size to the computational power decreases the execution time significantly in heterogeneous clusters, while changing this parameter in homogeneous clusters does not always performs better. Moreover, changing this parameter affects to stages of the algorithm that are independent of the population size, such as the migration. These results are a promising start for adapting EAs to the performance of each execution node.

In the future it would be interesting to check the scalability of this approach, using more computational nodes and larger problem instances. In addition, other parameters such as migration rate or crossover probability could be adapted to the execution nodes. Different benchmarks will be also used to lead to automatic parameter adaptation in runtime, with different nodes entering or exiting in the topology, or adapting to the current load of the system.

Acknowledgements This work has been supported in part by FPU research grant AP2009-2942 and projects EvOrq (TIC-3903), SINECA (0100DGT21285, Spanish Direccion General de Trafico) and TIN2011-28627-C04-02.

References

1. E. Alba and G. Luque. Evaluation of parallel metaheuristics. In Springer, editor, *Parallel Problem Solving from Nature (PPSN)*, volume 4193 of *LNCS*, pages 9–14, 2006.
2. Enrique Alba, Antonio J. Nebro, and José M. Troya. Heterogeneous computing and parallel genetic algorithms. *Journal of Parallel and Distributed Computing*, 62(9):1362 – 1385, 2002.
3. Mine Altunay, Paul Avery, Kent Blackburn, Brian Bockelman, Michael Ernst, Dan Fraser, Robert Quick, Robert Gardner, Sebastien Goasguen, Tanya Levshina, Miron Livny, John McGee, Doug Olson, Ruth Pordes, Maxim Potekhin, Abhishek Rana, Alain Roy, Chander Sehgal, Igor Sfiligoi, Frank Wuerthwein, and Open Sci Grid Executive Board. A Science Driven Production Cyberinfrastructure-the Open Science Grid. *Journal of GRID Computing*, 9(2, Sp. Iss. SI):201–218, JUN 2011.

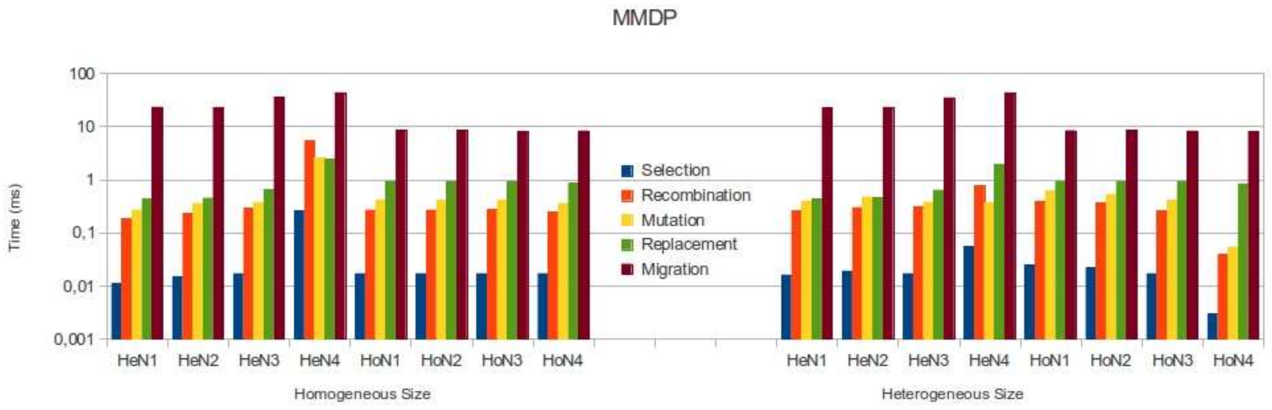

Fig. 11 Average running time in each stage of the algorithm for the MMDP problem.

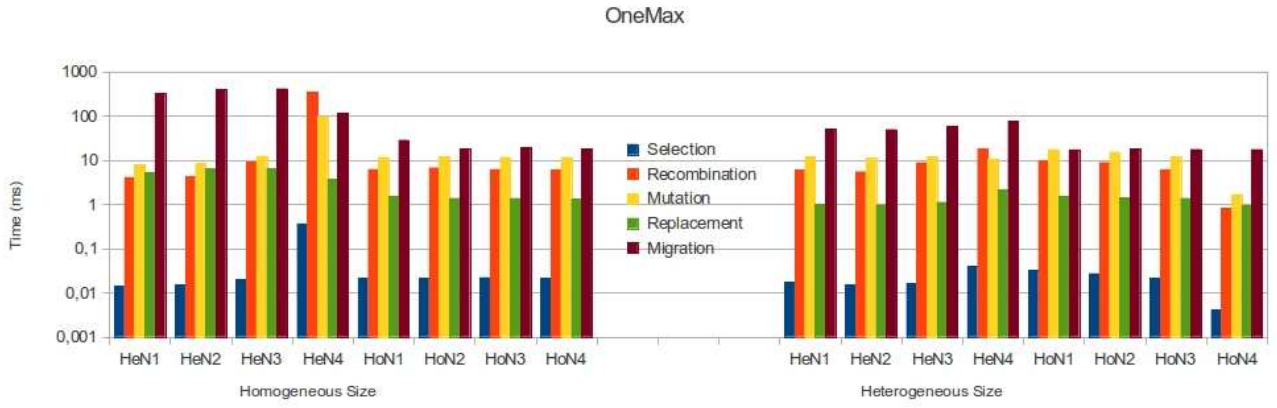

Fig. 12 Average running time in each stage of the algorithm for the ONEMAX problem.

Table 7 Times of the stages of the algorithm for the MMDP problem (in ms).

Homogeneous Size					
Node	Selection	Recombination	Mutation	Replacement	Migration
HeN1	0,011 ± 0,023	0,181 ± 0,149	0,269 ± 0,064	0,429 ± 4,873	23,097 ± 31,917
HeN2	0,015 ± 0,009	0,231 ± 0,116	0,357 ± 0,043	0,449 ± 5,214	22,928 ± 35,187
HeN3	0,017 ± 0,016	0,291 ± 0,132	0,372 ± 0,117	0,655 ± 6,533	36,139 ± 38,434
HeN4	0,257 ± 0,371	5,381 ± 14,941	2,556 ± 1,611	2,400 ± 5,588	43,490 ± 9,475
HoN1	0,017 ± 0,016	0,268 ± 0,555	0,405 ± 0,051	0,913 ± 1,350	8,428 ± 5,276
HoN2	0,017 ± 0,029	0,267 ± 0,409	0,405 ± 0,037	0,911 ± 1,419	8,441 ± 4,869
HoN3	0,017 ± 0,021	0,272 ± 0,589	0,404 ± 0,249	0,914 ± 1,420	8,177 ± 4,072
HoN4	0,017 ± 0,010	0,247 ± 0,479	0,356 ± 0,042	0,857 ± 1,636	8,284 ± 4,770
Heterogeneous Size					
Node	Selection	Recombination	Mutation	Replacement	Migration
HeN1	0,016 ± 0,012	0,259 ± 0,402	0,384 ± 0,086	0,435 ± 4,760	22,389 ± 31,184
HeN2	0,019 ± 0,015	0,297 ± 0,408	0,467 ± 0,256	0,464 ± 4,956	23,044 ± 32,704
HeN3	0,017 ± 0,016	0,303 ± 0,522	0,376 ± 0,118	0,634 ± 6,156	34,804 ± 35,557
HeN4	0,055 ± 0,161	0,769 ± 4,056	0,361 ± 0,653	1,957 ± 8,160	43,300 ± 26,672
HoN1	0,025 ± 0,017	0,389 ± 0,676	0,603 ± 0,060	0,929 ± 1,300	8,396 ± 4,147
HoN2	0,022 ± 0,017	0,362 ± 0,523	0,530 ± 0,228	0,921 ± 1,265	8,498 ± 4,694
HoN3	0,017 ± 0,011	0,259 ± 0,558	0,403 ± 0,050	0,916 ± 1,409	8,250 ± 4,516
HoN4	0,003 ± 0,005	0,039 ± 0,333	0,054 ± 0,029	0,836 ± 1,513	8,089 ± 4,602

Table 8 Times of the stages of the algorithm for the OneMax problem (in ms).

Homogeneous Size					
Node	Selection	Recombination	Mutation	Replacement	Migration
HeN1	0,014 ± 0,016	4,063 ± 3,290	7,826 ± 0,513	5,300 ± 62,474	328,262 ± 387,402
HeN2	0,015 ± 0,019	4,221 ± 3,251	8,325 ± 1,348	6,390 ± 71,926	398,119 ± 428,012
HeN3	0,020 ± 0,022	8,896 ± 3,561	12,289 ± 0,393	6,505 ± 77,069	410,141 ± 480,055
HeN4	0,362 ± 0,920	341,888± 381,390	97,075 ± 119,923	3,638 ± 14,627	112,680 ± 43,776
HoN1	0,021 ± 0,013	6,181 ± 2,386	11,545 ± 0,468	1,506 ± 3,642	28,355 ± 8,606
HoN2	0,021 ± 0,013	6,685 ± 1,927	11,779 ± 1,242	1,364 ± 2,437	18,034 ± 7,422
HoN3	0,022 ± 0,016	6,154 ± 2,238	11,585 ± 0,506	1,363 ± 2,437	18,948 ± 6,072
HoN4	0,021 ± 0,007	6,124 ± 2,346	11,560 ± 0,519	1,314 ± 2,308	17,920 ± 4,816
Heterogeneous Size					
Node	Selection	Recombination	Mutation	Replacement	Migration
HeN1	0,017 ± 0,002	5,955 ± 3,569	11,761 ± 0,500	1,000 ± 11,748	50,470 ± 81,518
HeN2	0,015 ± 0,002	5,448 ± 3,102	10,879 ± 1,545	0,972 ± 11,253	48,942 ± 77,468
HeN3	0,016 ± 0,003	8,733 ± 2,180	11,672 ± 0,870	1,113 ± 7,352	59,133 ± 8,825
HeN4	0,040 ± 0,035	17,943 ± 23,543	10,751 ± 1,683	2,144 ± 9,815	76,816 ± 15,500
HoN1	0,032 ± 0,014	9,587 ± 5,671	17,506 ± 1,083	1,482 ± 2,245	17,121 ± 6,302
HoN2	0,027 ± 0,015	8,826 ± 5,850	15,262 ± 1,091	1,422 ± 2,766	17,831 ± 14,158
HoN3	0,021 ± 0,013	6,108 ± 3,461	11,655 ± 0,534	1,365 ± 2,294	17,440 ± 6,578
HoN4	0,004 ± 0,002	0,807 ± 0,749	1,653 ± 0,051	0,922 ± 2,672	17,411 ± 12,177

4. Lourdes Araujo and Juan Julián Merelo Guervós. Diversity through multiculturality: Assessing migrant choice policies in an island model. *IEEE Trans. Evolutionary Computation*, 15(4):456–469, 2011.
5. O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, and A. van Moorsel. The self-star vision. *Self-star Properties in Complex Information Systems*, pages 1–20, 2005.
6. Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25:599–616, June 2009.
7. Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.
8. Julián Domínguez and Enrique Alba. HydroCM: A hybrid parallel search model for heterogeneous platforms. In El-Ghazali Talbi, editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 219–235. Springer Berlin Heidelberg, 2013.
9. Bernabé Dorronsoro and Pascal Bouvry. Cellular genetic algorithms without additional parameters. *The Journal of Supercomputing*, 63:816–835, 2013.
10. A. E. Eiben and Selmar K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.
11. I Foster. Globus Toolkit version 4: Software for service-oriented systems. In Jin, H and Reed, D and Jiang, W, editor, *Network and Parallel Computing Proceedings*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13, 2005.
12. C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case-study. *International Journal on Artificial Intelligence Tools*, 15(2):173, 2006.
13. J.F. Garamendi and J.L. Bosque. Parallel implementation of evolutionary strategies on heterogeneous clusters with load balancing. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., april 2006.
14. P. García-Sánchez, J. González, P.A. Castillo, M.G. Arenas, and J.J. Merelo-Guervós. Service oriented evolutionary algorithms. *Soft Computing*, pages 1–17, 2013. 10.1007/s00500-013-0999-5.
15. David E. Goldberg, Kalyanmoy Deb, and Jeffrey Horn. Massive multimodality, deception, and genetic algorithms. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature, 2*, pages 37–48, Amsterdam, 1992. Elsevier Science Publishers, B. V.
16. Yiyuan Gong and Alex Fukunaga. Distributed island-model genetic algorithms using heterogeneous parameter settings. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011, New Orleans, LA, USA, 5-8 June, 2011*, pages 820–827. IEEE, 2011.
17. Yiyuan Gong, Morikazu Nakamura, and Shiro 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, New York, NY, USA, 2005. ACM.
18. Antonio Gómez-Iglesias, Miguel A. Vega-Rodríguez, Francisco Castejón-Magaña, Miguel Cárdenas-Montes, and Enrique Morales-Ramos. Evolutionary computation and grid computing to optimise nuclear fusion devices. *Cluster Computing*, 12:439–448, 2009.
19. Jos Parejo, Antonio Ruiz-Cortés, Sebastin Lozano, and Pablo Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 16:527–561, 2012. 10.1007/s00500-011-0754-8.
20. Andres J. Ramirez, David B. Knoester, Betty H. C. Cheng, and Philip K. McKinley. Plato: a genetic algorithm approach to run-time reconfiguration in autonomic computing systems. *Cluster Computing*, 14(3):229–244, 2011.
21. Carolina Salto and Enrique Alba. Designing heterogeneous distributed gas by efficiently self-adapting the migration period. *Applied Intelligence*, 36:800–808, 2012.
22. J.D. Schaffer and L.J. Eshelman. On Crossover as an Evolutionary Viable Strategy. In R.K. Belew and L.B.

Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 61–68. Morgan Kaufmann, 1991.