Population size adaptation in distributed evolutionary algorithms on heterogeneous clusters

Pablo García-Sánchez[a], Jesús González[a], Maribel García Arenas[a], Pedro A. Castillo[a], Carlos Fernandes[b], Antonio Miguel Mora[a], Juan Julián Merelo[a]

[a]*Department of Computer Architecture and Computer Technology and CITIC-UGR, University of Granada, Granada, Spain. Tel: +34958241778. Fax: +34958248993*
[b]*LaSEEB-ISR-IST, Technical University of Lisbon (IST), Lisbon, Portugal*

Abstract

This paper presents a study on population size adaptation in a distributed Evolutionary Algorithm executed on a heterogeneous cluster. Two adaptation schemes have been tested: an offline and an online parameter setting. Results show that setting the population size according to the computational power in the heterogeneous cluster decreases the time required to obtain the optimum in two well-known problems with different characteristics and computational demands (MMDP and OneMax). Meanwhile the same parameter configuration could not improve the time in a homogeneous cluster. In addition, a study of the influence of the different population sizes on each stage of the algorithm is presented. This opens a new research line on the adaptation (offline or online) of parameters to the computational power of the devices.

Keywords: Evolutionary Algorithms, Genetic Algorithms, Heterogeneous computation, Distributed computing, Parameter Tuning, Parameter Control

1. Introduction

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 compete using their *fitness* (quality of adaptation) with the rest of solutions. Every iteration of the algorithm (or *generation*) the population evolves by means of selection and recombination/mutation to create a new set of candidates, until a *stop criterion* (e.g. 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 usually describes the problem to solve.

New trends in distributed computing such as Cloud Computing [**?**], GRID [**?**] or Service Oriented Science [**?**] are leading to heterogeneous computational devices, including for

Email addresses: pgarcia@atc.ugr.es (Pablo García-Sánchez), jesusgonzalez@ugr.es (Jesús González), maribel@ugr.es (Maribel García Arenas), pedro@atc.ugr.es (Pedro A. Castillo), cfernandes@laseeb.org (Carlos Fernandes), amorag@geneura.ugr.es (Antonio Miguel Mora), jmerelo@geneura.ugr.es (Juan Julián Merelo)

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 a heterogeneous cluster. Distributed Evolutionary Algorithms (dEAs) [**?** **?**] have been tested successfully in this type of systems [**?**] and they have become very popular because their implementation is not complex. The most extended model exploits a coarse grained parallelism with sporadic communications, being fit to be executed in distributed architectures such as clusters or GRIDs [**?**].

In distributed EAs a set of nodes executes simultaneously the EA, working with different sub-populations (or islands) at the same time. Every certain number of generations one or more individuals are interchanged (migrated) between sub-populations, which are connected following a specific topology. Figure 1 shows this model with a ring topology.

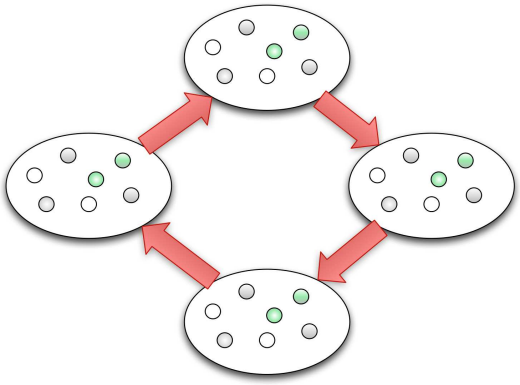


Figure 1: Island model scheme using a neighborhood ring topology.

Distributed EAs can be executed in homogeneous clusters with the same parameters in all nodes (homogeneous dEAs), or with different parameters or nodes's features (heterogeneous dEAs). It has also been showed [**?**] that dEAs with the same parameter configuration are even more efficient in time and evaluations on heterogeneous hardware configurations than on clusters with 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 parameter configuration has also been showed more efficient time-wise than a fixed set of parameters for different problems [**?**].

These facts have motivated us to study a combination of both ideas in this paper: dEAs on a heterogeneous set of nodes with different parameter values adapted to each node. In this study, the parameter to adapt to the computational power of each node has been the population size of each island. The population size is a key to obtain good performance in EAs [**?**] and it has been studied as a fixed [**?**] or adaptive parameter during runtime [**?**], but without taking into account the computational power of each machine.

In this work, a heterogeneous distributed system has been used to give an insight to the following questions:

- Can a distributed EA be adapted to leverage the capability of a 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 using the same population sizes in a homogeneous and a 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 (Section 3). Then, the results of the experiments are shown (Section 4), 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* [**?**]. The first one refers to setting up a number of parameters of an Evolutionary Algorithm and changing these values in running time (online). The parameter tuning consists in establishing a good set of parameters before the run (offline), and do not change them during the execution.

Computational performance of nodes or network speed can also be inherent parameters of an algorithm. In [**?**], Alba et al. 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 than the same algorithm running in homogeneous machines, although the parameter setting was the same in both clusters. Some authors have expanded this idea by adapting the algorithm to be executed: Domínguez et al. [**?**] 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 (computational devices), and simpler meta-heuristics (SA) in slower ones, obtaining better results than other configurations. As in previous works, the parameters were not adapted to the node. Gong et al. in [**?**] studied different configurations of heterogeneous machines for a tree topology. However, the heterogeneity was simulated in a homogeneous cluster with programs to add computational load. Load-balancing was also applied taking into account the computational load of the nodes in the work of Garamendi et al. [**?**]: 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 and the algorithm parameters were not adapted.

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 by Gong and Fukunaga in [**?**]. That model outperformed a tuned canonical dGA with the same parameter values in all islands. Finally, adapting the migration rate produced better results than homogeneous periods, as explained by Salto and Alba in [**?**]. Previous works were only tested in homogeneous clusters.

Our work presents a combination of some of the previous ideas: a set of parameters are adapted (offline and online) to the computational power of each machine used, and compared in different hardware 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. Experimental setup

This section presents the parameters and platforms to conduce the experiments.

3.1. Algorithm used

The experimentation is centered in a distributed GA. Figure 2 shows the pseudo-code of the used algorithm. The algorithm is steady-state, i.e. every generation the offspring is mixed with the parents and the worst individuals are removed. The used neighborhood topology for migration between islands (nodes) is a ring (see Figure 1). The best individual is sent to the neighbour in the ring, after a fixed number of generations in each island. The algorithm stops when the optimum (the solution to the problem) is found.

```
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
```

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

3.2. Problems

The problems to evaluate are the Massively Multimodal Deceptive Problem (MMDP) [**?**] and the OneMax problem [**?**]. 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 bits each one (s_i). Depending of the number of ones (unitation) s_i takes the values shown in Table 1.

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 have to find a global solution out of 2^{5k} additionally to deceptiveness. In this work $k = 25$.

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

Table 1: 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

Table 2: 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

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

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

3.3. Hardware and parameter configurations

Five configurations have been tested:

- HoSi/HoHa: Homogeneous Size/Homogeneous Hardware. The same population size in each island on a homogeneous cluster.
- HoSi/HeHa: Homogeneous Size/Heterogeneous Hardware. The same population size in each island on a heterogeneous cluster.
- HeSi/HoHa: Heterogeneous Size/Homogeneous Hardware. Different population sizes in each island on a homogeneous cluster.
- HeSi/HeHa: Heterogeneous Size/Heterogeneous Hardware. Different population sizes in each island on a heterogeneous cluster.
- AdSi/HeHa: Adaptive Size/Heterogeneous Hardware. Online adaptation of population sizes in each island on a heterogeneous cluster.

Two different computational systems have been used: a *heterogeneous cluster* and a *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 2 shows the features of each system and the name of the nodes.

Table 3: Average number of generations in each node when the optimum is found on the heterogeneous cluster with heterogeneous size.

Node	HeN1	HeN2	HeN3	HeN4
MMDP problem				
Generations	10990.25	10732.075	7721.15	717.95
Proportion	36.43	35.58	25.59	2.38
OneMax problem				
Generations	2430.27	2353.77	1423.77	91.5
Proportion	38.58	37.36	22.6	1.45

3.4. Homogeneous Size configuration

In this configuration, each node has 256 individuals (so, the total amount is 1024). After executing the algorithm 40 times per problem on the heterogeneous cluster, we have obtained the average number of generations in each node, as it can be seen in Table 3. Note how the generations attained (and their proportion in every node) to reach the optimum depends on the problem considered (besides the hardware).

3.5. Heterogeneous Size configuration

Our aim consists in validating the following hypothesis: adapting the population size to the computational power of the nodes of a heterogeneous cluster presents an improvement in execution time. In this work, we have used the average number of generations obtained in the HoSi/HeHa configuration for both problems to determine the computational power of the heterogeneous machines. 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 on the CPU speed. Although this is not obviously the best way, it is a possible way to establish the computational power for the experiments of this work and to determine if changing the population size according the computational power reduces the computing time of the whole approach. It should be considered that the contribution of this work is not the way we have computed these sizes, but compare the algorithm with parameters adapted to their power.

Thus, we have used the obtained average number of generations in the previous sub-section to set proportionally the sizes in the HeSi/HeHa and HeSi/HoHa configurations, by dividing the total number of individuals (1024). Note that, even having two nodes with the same processors and memory (HeN1 and HeN2), they could have different computational power: this may be produced by different operating systems, virtual machine versions, or number of processes being executed (inside a node).

3.6. Adaptive Size configuration

Finally, to validate our hypothesis we propose a third configuration to adapt the population size to the computational power of the nodes of the heterogeneous cluster. In this experiment, the adaptation of the island size to the computational power of the islands is performed during runtime (online). Each time a node (N) receives an individual, it compares their current number of generations (Gen_N) with the individual's sender (node $N-1$ in the ring). Then, the population size is adapted proportionally to the difference in the number of generations, following the next equation:

Table 4: Parameters used in all configurations.

Name	Value
Crossover type	Two-points crossover
Crossover rate	0.5
Mutation rate	1/individual size
Selection	2-tournament
Replacement	Steady-state
Generations to migrate	64
Number of individuals to migrate	1
Stop criterion	Optimum found
Individual size for MMDP	150
Individual size for OneMax	5000
Runs per configuration	40
Total individuals in HoSi and HeSi	1024
Population size in each node in HoSi	256
Population sizes in HeSi for MMDP	374, 364, 262 and 24 (from N1 to N4)
Population sizes in HeSi for OneMax	396, 382, 232 and 14 (from N1 to N4)
Maximum island size in AdSi	1024
Minimum island size in AdSi	16
Initial island size in AdSi	256

$$size'_N = \frac{Gen_N}{Gen_{N-1}} size_N \tag{3}$$

If the new size is larger than the actual size the new individuals are added to the population cloning random existent ones. Otherwise, to reduce the size, the worst are removed.

With this possible online adaptation scheme, each node only requires to receive information of one of the neighbours and not from the whole system. Thus, each node tends to have a number of individuals proportional to their computational power with respect to the other nodes. Experiments in homogeneous cluster do not alter the population sizes, as the number of current generations are equal in all nodes during runtime.

Table 4 summarizes all the parameters used in the experiments.

3.7. Framework

In order to deal with the operating system and architecture heterogeneity (different operating systems, processors, compilers, etc.), the OSGiLiath framework [**?**], 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 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 [**?**]. The transmission

Table 5: Results for the MMDP problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	11194.8 ± 18810.08	30161.42 ± 50722.03	7723372.8 ± 12984841.71	27871.075 ± 44583.14
HeSi/HeHa	2506.1 ±5308.872	8683.9 ± 18459.58	2453677 ±5217896.18	8110.9 ± 17162.86
AdSi/HeHa	2407.10 ±3938.43	8376.35 ± 14140.55	2948946.15 ± 5165324.99	10235.89 ± 17193.98
HoSi/HoHa	2614 ±5889.93	10259.22 ± 23153.23	2628409.6 ± 5927278.22	11560.8 ± 26072.14
HeSi/HoHa	5411.92 ±15608.81	10689.15 ± 30790.7	1844908.1 ± 5314771.88	9520.325 ± 27237.35

mechanism is based on 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 LGPL V3 License.

4. Results

The three main objectives 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 (reaching the optimum). In this work, we focus in the last objective. As claimed by Alba and Luque in [**?**], assessing the performance of a parallel EA by the number of fitness 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 carried out by all nodes, and the maximum number of generations required by the faster 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). That is, in this work, we are not making the heterogeneous cluster comparable or better in time than the homogeneous one (because they are, obviously, different), but show that the same parameter configuration can improve performance in time in heterogeneous clusters and could not have effect in homogeneous ones.

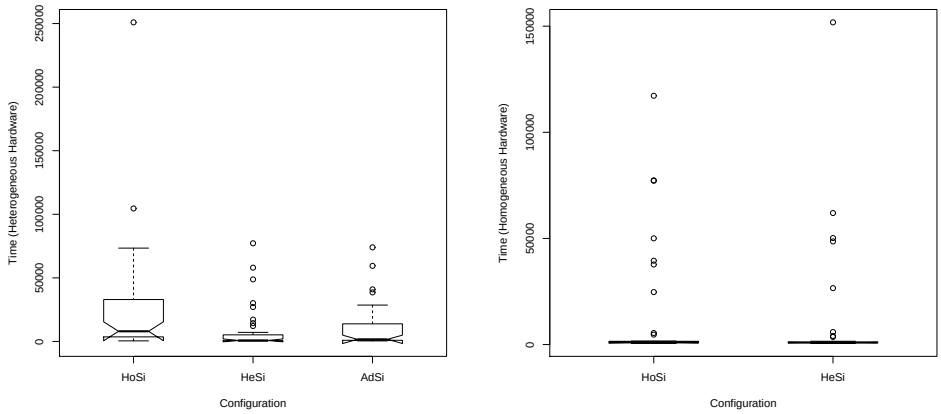
4.1. MMDP results

Table 5 shows the results for the MMDP problem. These results are also shown in the boxplots of Figure 3 (time) and Figure 4 (evaluations). Table 9 shows the statistical significance of the results. First, a Kolmogorov-Smirnov test is performed to assess the normality of the distributions. As all distributions are not normal, we use non-parametric tests. To compare between two methods (HoSi and HeSi in the homogeneous cluster) a Wilcoxon test has been applied. For a three methods comparison (HoSi, HeSi and AdSi in heterogeneous cluster) a Kruskal-Wallis test has been used.

In the HeHa system, adapting offline the population to the computational power of each node makes the algorithm finish significantly earlier, and also, needing a lower number of evaluations to reach the solution. 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 (p-value=0.52 for time and 0.08 for evaluations).

To see the differences on 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

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



(a) Heterogeneous cluster (b) Homogeneous cluster

Figure 3: Time to obtain the optimum in the MMDP problem (milliseconds).

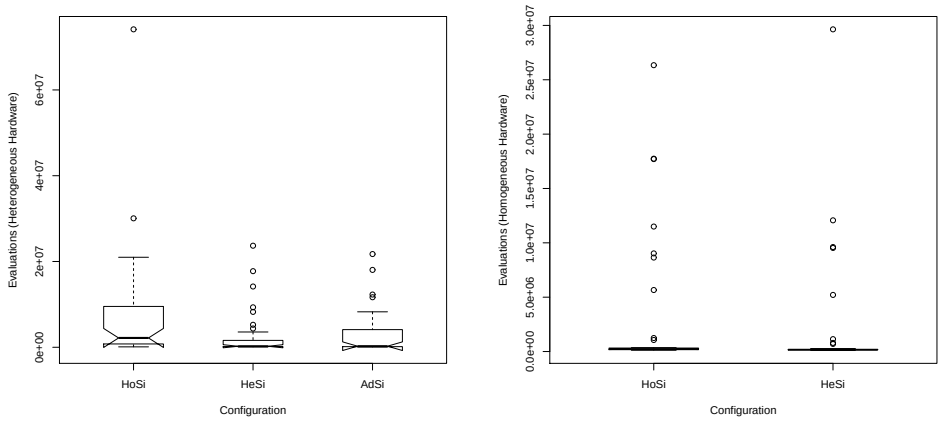
Table 6: Average population size in each node on the heterogeneous cluster with adaptive size (MMDP).

Node	HeN1	HeN2	HeN3	HeN4
Size	556.31	504.30	321.15	19.81
Proportion	39.69	35.98	22.91	1.41

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 spent in the nodes for sending the migrant individual to other nodes (not while they are receiving them). In the HoHa configurations, the populations are evolved at the same time, being the average fitness similar in all nodes during all run.

With respect to AdSi/HeHa, results are significantly equal (p-value 0.139) to HeSi/HeHa (and, therefore, better than HoSi/HeHa), but this time no previous tuning is required. Average population sizes in each node are shown in Table 6. The proportion of sizes are similar to the proportions in Table 3. Figure 7 shows all the possible sizes in each node during all the runs. This figure show that the variation of the population sizes lies proportionally to the computational power of each node. The outliers in boxplots are produced during the size changing, as can be seen in Figure 8. N4 is the slower node with difference and its size keeps always near the minimum (16 individuals).

Summarizing, adapting the population sizes to the computational power of each machine (offline and online) has reduced the time to obtain the optimum. The same heterogeneous fixed sizes in the homogeneous cluster does not produces a significant decrease of running time, so the improvement is produced by the heterogeneity and not for the different island sizes. Also, the AdSi proposal is not applicable here because there is not differences of generations during runtime.



(a) Heterogeneous cluster (b) Homogeneous cluster

Figure 4: Number of evaluations for MMDP problem.

4.2. OneMax results

Results for this problem are shown in Table 7 and Figures 9 and 10. In this case, adapting the population sizes significantly decreases the running time for solving it in the heterogeneous cluster, but in this case, the number of evaluations is increased (see statistical significance in Table 10). 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 11, 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 12, smaller population sizes make that slower nodes increase the number of evaluations, 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).

In the AdSi/HeHa configuration significantly better results in terms of execution time (and number of evaluations) are also attained, even more clearly than with HeSi. Average sizes (Table 8) and boxplots (in Figure 13) during all the runs also show proportionality to the computational power of each machine. As in MMDP, some oscillations (outlines in boxplots) may appear during the execution (as can be seen in Figure 14).

Table 7: Results for the OneMax problem.

Configuration	Max. generations	Total generations	Total evaluations	Time (ms)
HoSi/HeHa	2430.34 ± 70.16	6299.31 ± 250.87	1614673.45 ± 64223.09	160713.65 ± 8873.46
HeSi/HeHa	2643.34 ±150.82	7969.58 ±214.92	1802321.65 ± 30511.96	151822.75 ±4764.95
AdSi/HeHa	3698.30 ± 494.56	9465.25 ± 635.07	1149277.43 ± 58887.13	103919.33 ± 6296.39
HoSi/HoHa	1791.32 ± 31.64	7111.05 ±125.11	1822476.8 ±32029.78	141176.1 ±2493.72
HeSi/HoHa	13698.12 ± 406.85	16012.625 ± 482.61	895698.2 ± 29520.99	77898.85 ± 2935.57

Table 8: Average population size in each node on the heterogeneous cluster with adaptive size (OneMax).

Node	HeN1	HeN2	HeN3	HeN4
Size	267.09	158.63	74.20	16.29
Proportion	51.73	30.72	14.37	3.15

Table 9: Statistical significance of the results for MMDP.

Time					
Kruskal-Wallis chi-squared = 20.3042, df = 2, p-value = 3.899e-05					
Configuration	Test	obs.dif	critical.dif	p-value	difference
AdSi-HeSi/HeHa	K-W	13.19231	18.38851	0.1390	FALSE
AdSi-HoSi/HeHa	K-W	21.11538	18.38851	0.0067	TRUE
HeSi/HeHa-HoSi/HeHa	K-W	34.30769	18.38851	9×10^{-5}	TRUE
HoSi/HoHa-HeSi/HoHa	Wilcoxon	-	-	0.52	FALSE
Evaluations					
Kruskal-Wallis chi-squared = 11.9676, df = 2, p-value = 0.002519					
AdSi-HeSi/HeHa	K-W	2.794872	18.38851	1.0	FALSE
AdSi-HoSi/HeHa	K-W	21.487179	18.38851	0.0207	TRUE
HeSi/HeHa-HoSi/HeHa	K-W	24.282051	18.38851	0.0028	TRUE
HoSi/HoHa-HeSi/HoHa	Wilcoxon	-	-	0.08	FALSE

Table 10: Statistical significance of the results for OneMax.

Time					
Kruskal-Wallis chi-squared = 66.4965, df = 2, p-value = 3.635e-15					
Configuration	Test	obs.dif	critical.dif	p-value	difference
AdSi-HeSi/HeHa	K-W	33.27586	15.87987	2.3×10^{-10}	TRUE
AdSi-HoSi/HeHa	K-W	53.56897	15.87987	$<2\times10^{-16}$	TRUE
HeSi/HeHa-HoSi/HeHa	K-W	20.29310	15.87987	4.2×10^{-6}	TRUE
HoSi/HoHa-HeSi/HoHa	Wilcoxon	-	-	3×10^{-8}	TRUE
Evaluations					
Kruskal-Wallis chi-squared = 75.7342, df = 2, p-value < 2.2e-16					
AdSi-HeSi/HeHa	K-W	57.72414	15.87987	$<2\times10^{-16}$	TRUE
AdSi-HoSi/HeHa	K-W	29.27586	15.87987	$<2\times10^{-16}$	TRUE
HeSi/HeHa-HoSi/HeHa	K-W	28.44828	15.87987	$<1.3\times10^{-14}$	TRUE
HoSi/HoHa-HeSi/HoHa	Wilcoxon	-	-	3×10^{-8}	TRUE

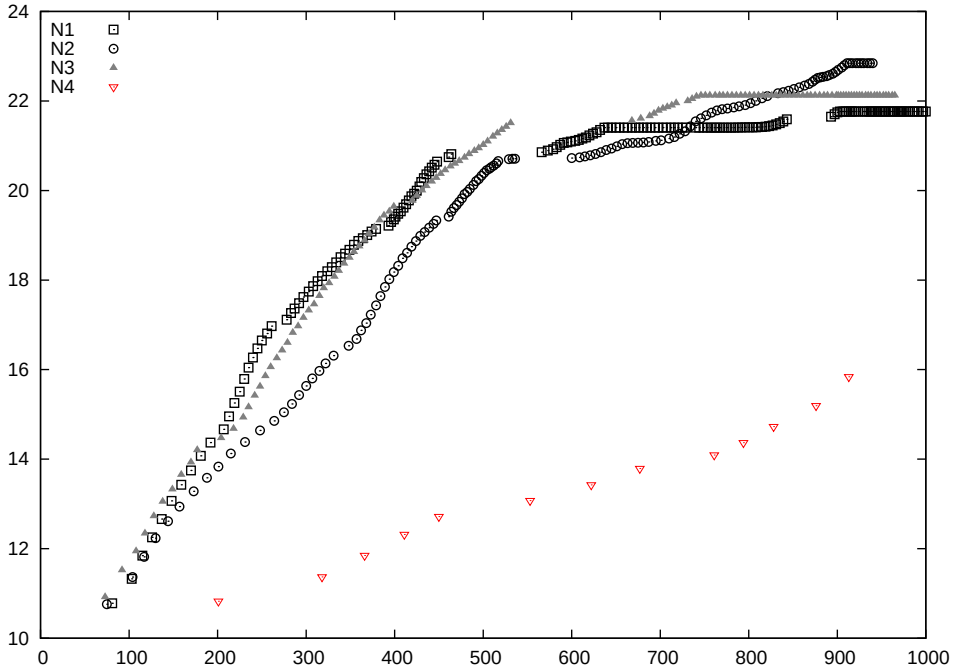


Figure 5: Average fitness in the first 1000 milliseconds of execution of the four nodes of the heterogeneous cluster with the same population sizes (HoSi/HeHa) for the MMDP problem.

4.3. 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 with fixed sizes (HoSi and HeSi). Tables 11 and 12 show the average and standard deviation of the time spent in each stage of the algorithm (He=Heterogeneous cluster, Ho=Homogeneous cluster). Figures 15 and 16 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 standard deviation of the migration is larger in the HeHa cluster because the network is having real conditions of traffic during the experiment. In the MMDP problem (Table 11) 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, bottlenecks can be propagated (as it can be seen in Figure 11).

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 problems only in the node HeN4. The reason might be the creation of new objects (memory allocation), which in Java and in limited memory (and swapping) requires more time than the 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

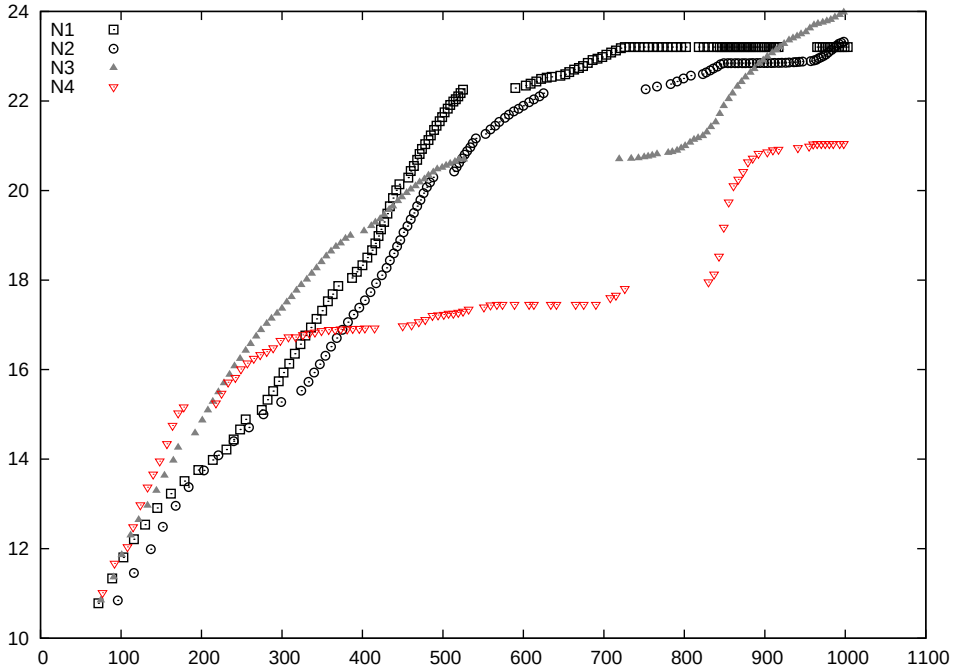


Figure 6: Average fitness in the first 1000 milliseconds of execution of the four nodes of the heterogeneous cluster with different population sizes (HeSi/HeHa) for the MMDP problem.

of the individuals affects to some parts of the EA; for example, in 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.

5. Conclusions

In this paper we describe a study on the adaptation of the population size of a distributed EA to the computational power of the different nodes of an heterogeneous cluster. Two adaptation schemes (offline and online) have been tested.

Results show that adapting (online or offline) the population size to the computational power of each node in the heterogeneous cluster yields significantly better results in time than keeping the same parameter in all nodes. This advantage is due to the combination of the heterogeneous parameters with the heterogeneity of the machines. On the contrary, the same (heterogeneous) parameter setting in all islands of the homogeneous cluster could not improve the results than considering the same parameter value in all nodes.

Moreover, changing the population size affects to stages of the algorithm that are independent of the population size, such as the migration. The population size adaptation is also affected by the problem to solve.

In this work, we calculate the computational power of each node proportionally to the average number of generations for the homogeneous parameter set. Also, a possible way to adapt

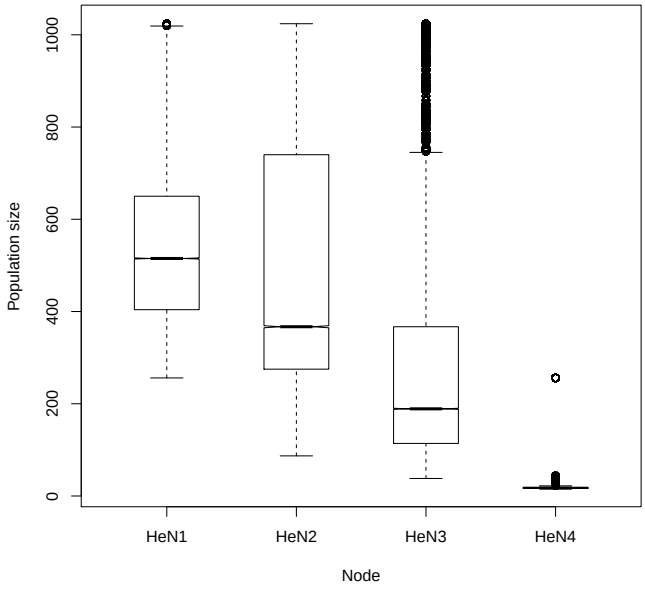


Figure 7: Boxplots of the population sizes in each node during all the runs for the MMDP problem.

online the population sizes is performed comparing the current generation with the neighbour generation. These results are a promising starting for adapting EAs to the performance of each execution node, using more adequate benchmarks or in a dynamic way.

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 (online), with different nodes entering or exiting in the topology, or adapting the parameters 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), CANUBE (CEI2013-P-14) and ANYSELF (TIN2011-28627-C04-02).

References

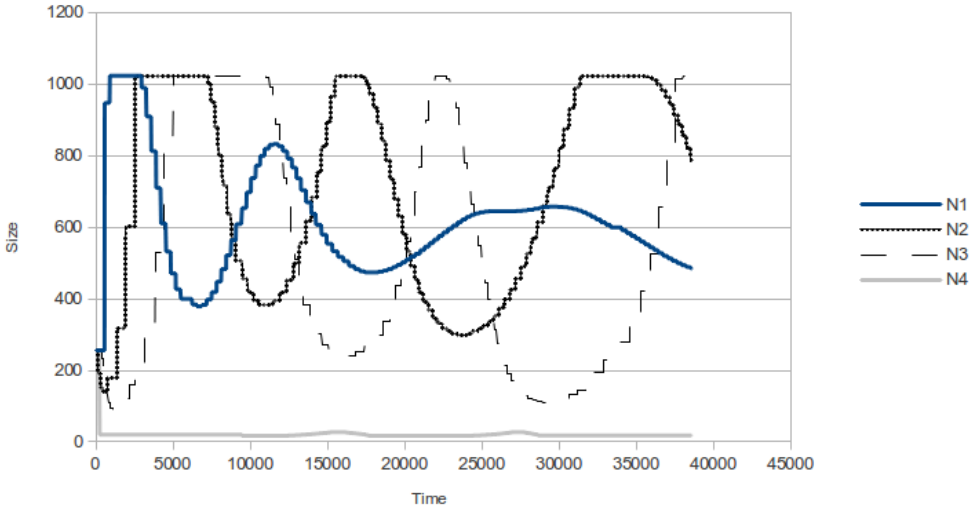
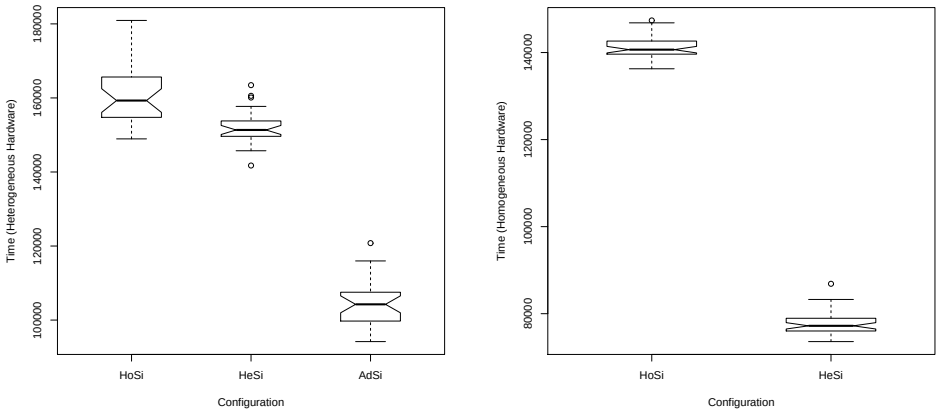
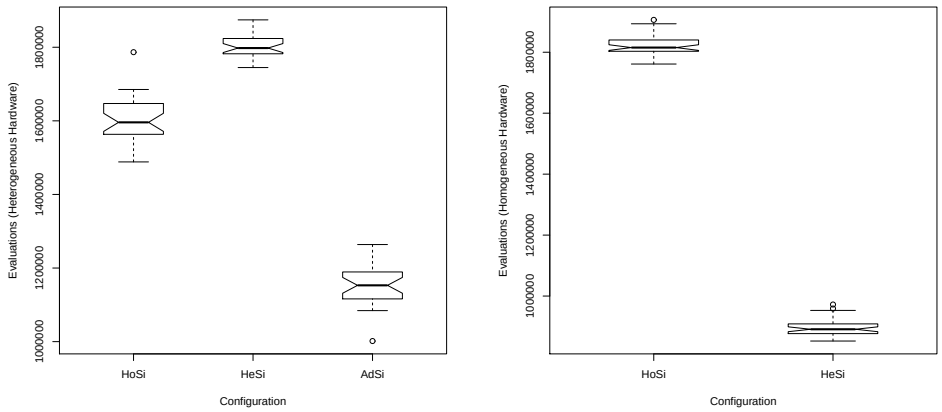


Figure 8: Population size in each node during one execution to solve the MMDP problem.



(a) Heterogeneous cluster (b) Homogeneous cluster

Figure 9: Time to obtain the optimum in the OneMax problem (milliseconds).



(a) Heterogeneous cluster (b) Homogeneous cluster

Figure 10: Number of evaluations for OneMax problem.

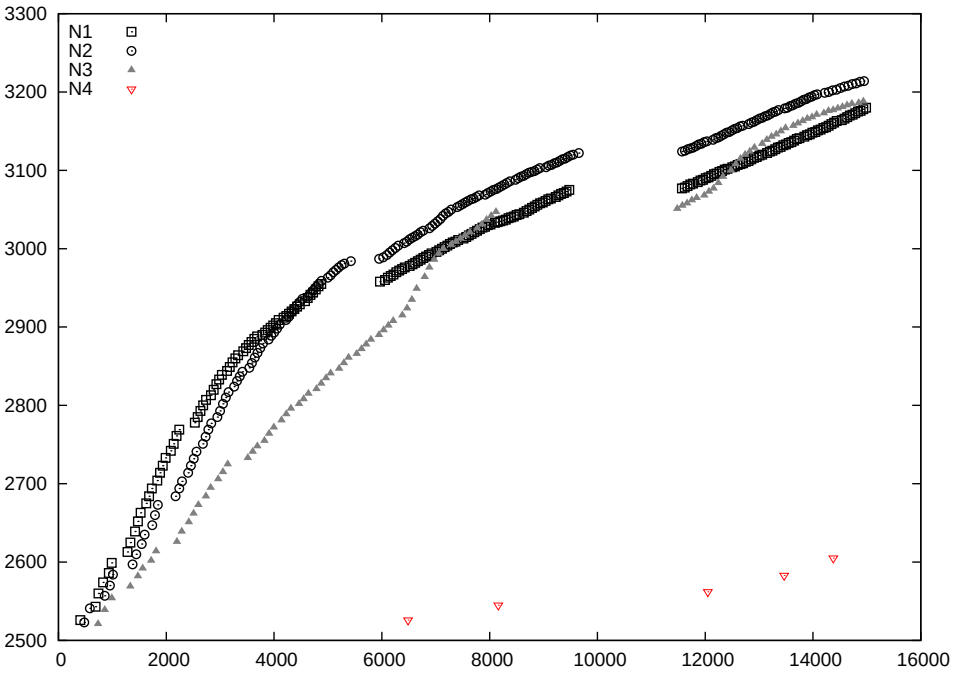


Figure 11: Average fitness in the first 15000 milliseconds of execution of the four nodes of the heterogeneous cluster with the same population sizes (HoSi/HeHa) for the OneMax problem.

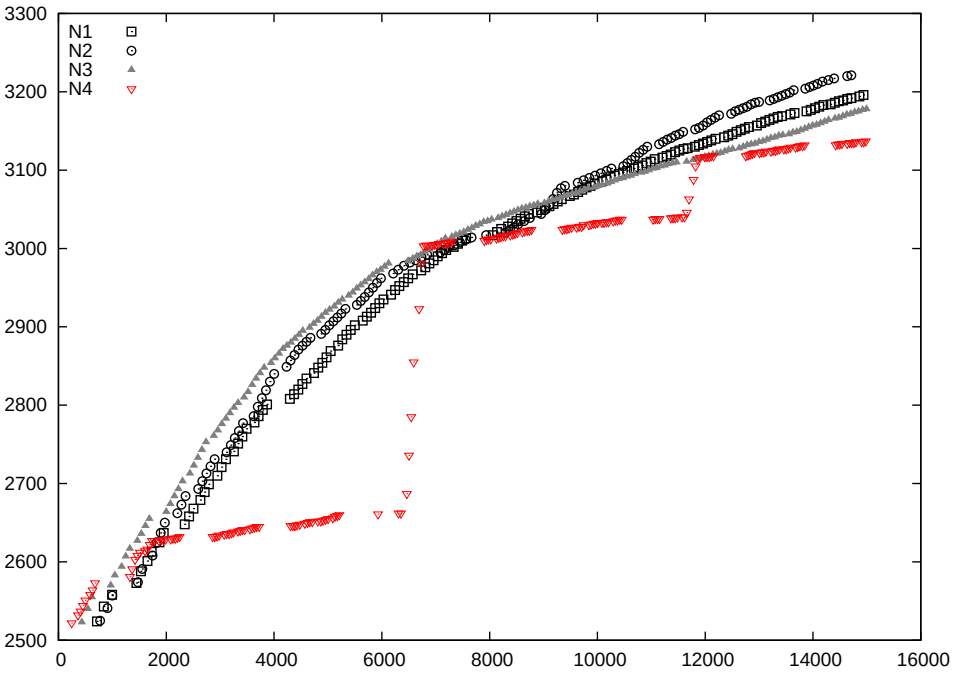


Figure 12: Average fitness in the first 15000 milliseconds of execution of the four nodes of the heterogeneous cluster with different population sizes (HeSi/HeHa) for the OneMax problem.

Table 11: Times of the stages of the algorithm for the MMDP problem (in ms).

Heterogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HeN1	0.077 ± 0.170	0.788 ± 0.779	1.004 ± 0.187	1.648 ± 20.185	82.458 ± 143.266
HoSi HeN2	0.088 ± 0.190	0.907 ± 0.932	1.145 ± 0.425	1.579 ± 17.907	76.725 ± 126.360
HoSi HeN3	0.105 ± 0.163	1.207 ± 0.927	1.374 ± 0.301	2.108 ± 21.848	108.605 ± 142.633
HoSi HeN4	1.165 ± 1.526	30.445± 59.553	12.221 ± 7.412	10.978 ± 57.135	84.936 ± 0.000
HeSi HeN1	0.067 ± 0.065	0.973 ± 0.403	1.411 ± 0.166	0.790 ± 6.266	28.081 ± 42.169
HeSi HeN2	0.062 ± 0.075	0.973 ± 0.470	1.433 ± 0.265	0.811 ± 7.056	29.667 ± 48.702
HeSi HeN3	0.066 ± 0.108	1.104 ± 0.346	1.435 ± 0.296	0.937 ± 7.072	40.964 ± 40.027
HeSi HeN4	0.109 ± 0.257	1.895 ± 5.611	0.913 ± 0.834	2.085 ± 5.626	43.880 ± 7.535
Homogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HoN1	0.163 ± 0.223	1.884 ± 2.386	1.591 ± 0.479	2.254 ± 5.513	40.256 ± 8.726
HoSi HoN2	0.151 ± 0.212	1.952 ± 2.876	1.597 ± 0.574	2.178 ± 4.922	37.110 ± 6.999
HoSi HoN3	0.154 ± 0.206	1.990 ± 3.010	1.591 ± 0.577	2.215 ± 4.743	36.413 ± 5.266
HoSi HoN4	0.146 ± 0.196	1.913 ± 2.697	1.651 ± 1.167	2.194 ± 5.124	38.429 ± 6.192
HeSi HoN1	0.214 ± 0.288	2.800 ± 3.793	2.359 ± 0.691	2.516 ± 4.706	36.972 ± 4.214
HeSi HoN2	0.190 ± 0.252	2.672 ± 3.902	2.277 ± 0.649	2.261 ± 4.546	41.171 ± 9.672
HeSi HoN3	0.148 ± 0.208	2.030 ± 3.161	1.623 ± 0.500	2.164 ± 4.512	35.551 ± 6.132
HeSi HoN4	0.045 ± 0.052	0.345 ± 1.121	0.217 ± 0.142	1.531 ± 4.856	38.106 ± 9.251

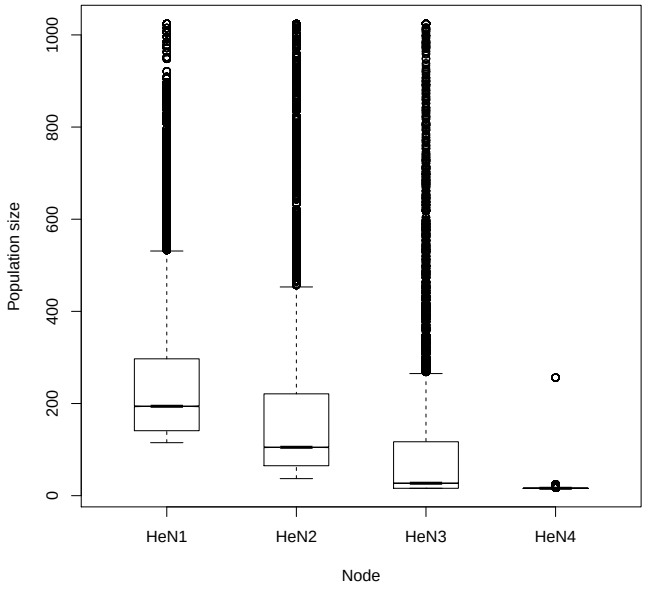


Figure 13: Boxplots of the population sizes in each node during all the runs for the OneMax problem.

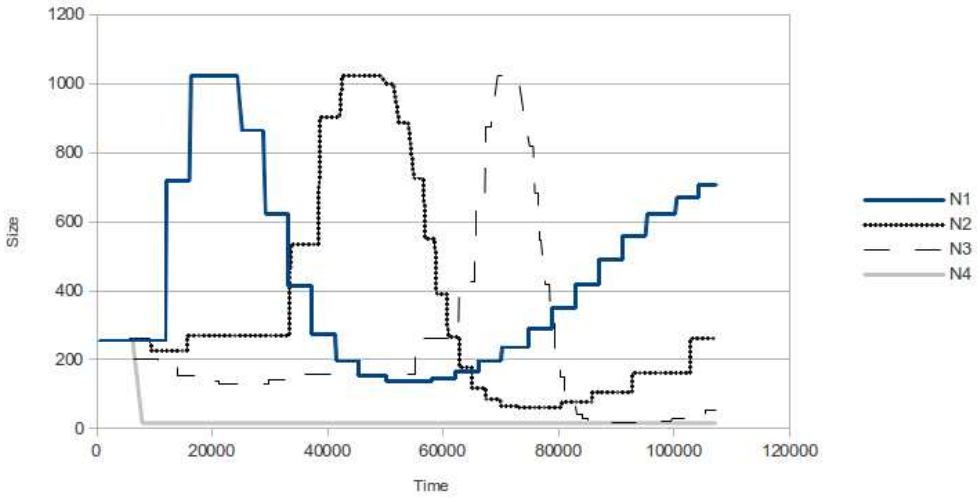


Figure 14: Population size in each node during one execution to solve the OneMax problem.

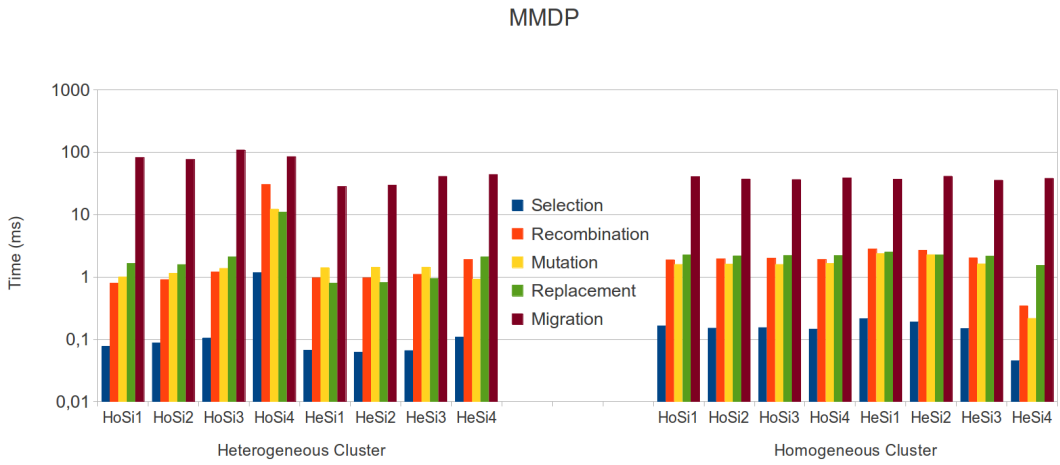


Figure 15: Average running time in each stage of the algorithm for the MMDP problem.

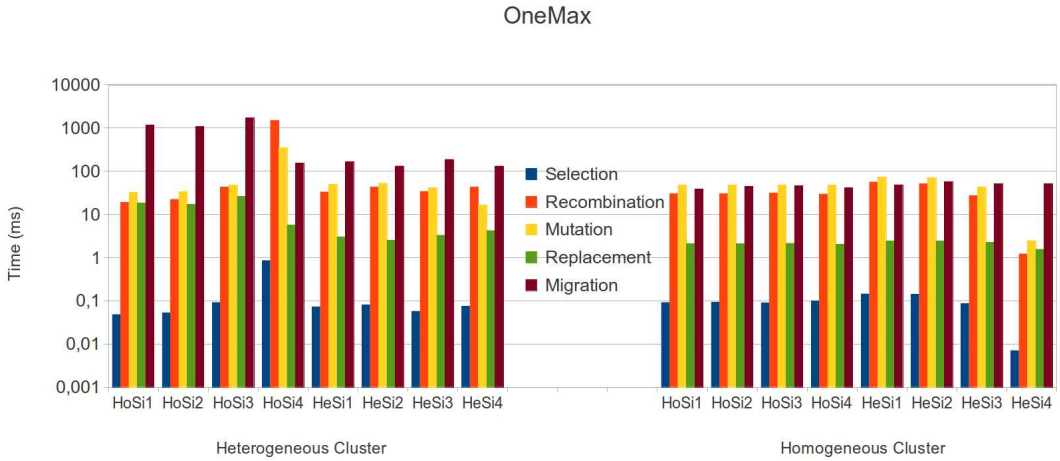


Figure 16: Average running time in each stage of the algorithm for the ONEMAX problem.

Table 12: Times of the stages of the algorithm for the OneMax problem (in ms).

Heterogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HeN1	0.048 ± 0.043	18.713 ± 13.454	31.984 ± 2.104	18.375 ± 197.676	1172.986 ± 1108.388
HoSi HeN2	0.052 ± 0.051	22.266 ± 22.716	33.553 ± 4.931	17.176 ± 180.580	1085.508 ± 995.382
HoSi HeN3	0.091 ± 1.005	42.634 ± 21.621	47.674 ± 0.546	26.094 ± 252.667	1708.402 ± 1207.925
HoSi HeN4	0.851 ± 0.435	1491.568 ± 1185.723	344.872± 6.634	5.655 ± 16.175	154.019 ±0.000
HeSi HeN1	0.072 ± 0.063	32.917 ± 26.792	49.103 ± 2.655	3.023 ± 27.647	163.479 ±157.172
HeSi HeN2	0.080 ± 0.092	43.001 ± 51.680	52.288 ± 13.210	2.527 ± 21.861	131.063 ±124.404
HeSi HeN3	0.057 ± 0.052	33.951 ± 15.063	41.375 ± 1.707	3.284 ± 30.170	186.467 ±163.906
HeSi HeN4	0.075 ± 0.107	42.443 ± 88.536	16.236 ± 12.028	4.194 ± 33.119	131.135 ±144.359
Homogeneous Cluster					
Node	Selection	Recombination	Mutation	Replacement	Migration
HoSi HoN1	0.091 ± 0.078	29.969 ± 21.459	47.445 ± 2.194	2.073 ± 6.970	38.782 ± 40.369
HoSi HoN2	0.093 ± 0.082	30.119 ± 22.029	47.247 ± 2.146	2.108 ± 7.440	44.303 ± 42.759
HoSi HoN3	0.089 ± 0.080	30.951 ± 21.904	47.103 ± 2.031	2.138 ± 8.006	46.107 ± 47.351
HoSi HoN4	0.098 ± 0.075	29.468 ± 20.876	47.086 ± 1.856	2.043 ± 7.491	41.458 ± 44.970
HeSi HoN1	0.144 ± 0.151	56.124 ± 48.229	72.811 ± 5.177	2.424 ± 9.056	48.165 ±57.798
HeSi HoN2	0.141 ± 0.152	51.226 ± 41.016	70.047 ± 4.152	2.427 ± 10.890	57.152 ±74.177
HeSi HoN3	0.086 ± 0.088	26.932 ± 20.460	42.963 ± 3.935	2.239 ± 8.658	51.014 ±49.648
HeSi HoN4	0.007 ± 0.008	1.215 ± 1.133	2.470 ± 0.098	1.553 ± 10.078	50.498 ± 63.983