Validating a Peer-to-Peer Evolutionary Algorithm

Juan Luis Jiménez Laredo[1], Pascal Bouvry[1],
Sanaz Mostaghim[2], and Juan Julián Merelo Guervós[3]

[1] Faculty of Sciences, Technology and Communication,
University of Luxembourg, Luxembourg City L-1359, Luxembourg.
e-mail: {juan.jimenez,pascal.bouvry}@uni.lu
[2] Karlsruhe Institute of Technologie
Kaiserstrasse 89, Karlsruhe D-76133, Germany.
e-mail: sanaz.mostaghim@kit.edu
[3] University of Granada. ATC-ETSIIT
Periodista Daniel Saucedo Aranda s/n 18071, Granada, Spain.
e-mail: jmerelo@geneura.ugr.es

Abstract. This paper proposes a simple experiment for validating a Peer-to-Peer Evolutionary Algorithm in a real computing infrastructure in order to verify that results meet those obtained by simulations. The validation method consists of conducting a well-characterized experiment in a large computer cluster of up to a number of processors equal to the population estimated by the simulator. We argue that the validation stage is usually missing in the design of large-scale distributed meta-heuristics given the difficulty of harnessing a large number of computing resources. That way, most of the approaches in the literature focus on studying the model viability throughout a simulation-driven experimentation. However, simulations assume idealistic conditions that can influence the algorithmic performance and bias results when conducted in a real platform. Therefore, we aim at validating simulations by running a real version of the algorithm. Results show that the algorithmic performance is rather accurate to the predicted one whilst times-to-solutions can be drastically decreased when compared to the estimation of a sequential run.

1 Introduction

Given that most computer devices nowadays are connected to the Internet continuously, volunteer computing systems [2] have arisen as an alternative to supercomputers or wide-area grid systems. Volunteer computing systems usually behave in a centralized fashion which might be a problem when huge numbers of clients are simultaneously connected, posing a challenge to the server, or converting it into a bottleneck in the case the systems become especially large. Peer-to-Peer (P2P) systems, where no node has any special role, do take advantage of the nature of the Internet and take more advantage of the bandwidth each node is connected with [14].

These systems have received much attention from the scientific community within the last decade. In this context and under the term of P2P optimization, many optimization heuristics such as Evolutionary Algorithms (EAs), Particle Swarm (PSO) or Branch-and-bound have been re-designed in order to take advantage of such computing platforms [16, 13, 5, 3]. The key issue here is that gathering a large amount of computational devices pose a whole set of practical problems. Therefore, and to the best of our knowledge, most of the approaches to P2P optimization –if not all– have been analyzed in simulators rather than in real environments. That way, this paper aims to go an step further and validate a P2P EA in a real large-scale infrastructure running up to 3008 parallel individuals.

To that aim, we consider the results published in [8] on the scalability of the Evolvable Agent model (i.e. a P2P EA model) in a simulated based environment. Such results are validated using an equally parametrized parallel version of the algorithm in a real environment[4]. In order to simplify the experimentation, the real platform consists of a cluster of homogeneous nodes. This allows the trace of computer failures and the minimization of asynchronous effects on the performance so that the characterization of the real environment mirrors the simulator settings.

In a first set of experiments, the algorithmic accuracy of the simulations is tested by running a medium size instance of trap functions [1]. We focus then on the fine-tuning of the population size and adjust both, simulation-based and parallel versions to their optimal sizes. Given that both approaches are equally parametrized, our validation proof relies on showing that the simulations require the same population size than the parallel version to induce the same progress in fitness. On the basis of previous results, a second set of experiments is conducted in order to prove the massive scalability of the approach. In this case, the computational performance of the model is assessed by tackling a larger problem instance for which the simulator points out large population size requirements. Here, results show that the parallel version is able to find the problem optimum in two and half hours in contrast with the estimation of hundred days of the sequential run.

The rest of the paper is organized as follow. Section 2 provides an overall description of the Evolvable Agent model. Section 3 explains the setup of the experiments. Results are analyzed in Section 4. Finally, we reach some conclusions and propose some future lines of work in Section 5.

2 Description of the Model

The Evolvable Agent (EvAg) model (proposed by Laredo et al. in [7]) is a fine-grained spatially structured EA in which every agent schedules the evolution

[4] In order to reproduce experiments, all the source-code –either the simulator or the parallel version of the algorithm– is available from our Subversion repository at http://forja.rediris.es/svn/geneura published under GNU public license.

process of a single individual and self-organizes its neighborhood via the newscast protocol. As explained by Jelasity and van Steen in [6], newscast runs on every node and defines the self-organizing graph that dynamically maintains some constant graphs properties at a virtual level such as a low average path length or a high clustering coefficient from which a small-world behavior emerges [15]. This makes the algorithm inherently suited for parallel execution in a P2P system which, in turn, offers great advantages when dealing with computationally expensive problems at the expected speedup of the algorithm.

Every agent acts at two different levels; the evolutionary level for carrying out the main steps of evolutionary computation (selection, variation and evaluation of individuals [4]) and the network level which defines P2P population structure.

The evolutionary level is depicted in Algorithm 1. It shows the pseudo-code of an $EvAg_i \in [EvAg_1 \dots EvAg_n]$ where $i \in [1 \dots n]$ and n is the population size. Despite the model not having a population in the canonical sense, neighbors EvAgs provide each other with the genetic material that individuals require to evolve.

Algorithm 1 Pseudo-code of an Evolvable Agent ($EvAg_i$)

```
Evolutionary level

$Ind_{current_i} \Leftarrow$ Initialize Agent
while not termination condition do
    $Pool_i \Leftarrow$ Local Selection($Neighbors_{EvAg_i}$)
    $Ind_{new_i} \Leftarrow$ Recombination($Pool_i$,$P_c$)
    Evaluate($Ind_{new_i}$)
    if $Ind_{new_i}$ better than $Ind_{current_i}$ then
        $Ind_{current_i} \Leftarrow Ind_{new_i}$
    end if
end while

Local Selection($Neighbors_{EvAg_i}$)
[$Ind_{current_h} \in EvAg_h, Ind_{current_k} \in EvAg_k$] $\Leftarrow$ Random selected nodes from the newscast
neighborhood
```

The key element at this level is the locally executable selection. Crossover and mutation never involve many individuals, but selection in EAs usually requires a comparison among all individuals in the population. In the EvAg model, the mate selection takes place locally within a given neighborhood where each agent selects the current individuals from other agents (e.g. $Ind_{current_h}$ and $Ind_{current_k}$ in Algorithm 1).

Selected individuals are stored in $Pool_i$ ready to be used by the recombination (and eventually mutation) operator. Within this process a new individual Ind_{new_i} is generated.

In the current implementation, the replacement policy adopts a replace if worst scheme, that is, if the newly generated individual Ind_{new_i} is better than the current one $Ind_{current_i}$, $Ind_{current_i}$ becomes Ind_{new_i}, otherwise, $Ind_{current_i}$ remains the same for the next generation. Finally, every EvAg iterates until a termination condition is met.

As previously mentioned, newscast is the canonical underlying P2P protocol in the EvAg model. It represents the network level of the model that conforms the population structure. Algorithm 2 shows the newscast protocol in an agent $EvAg_i$. There are two different tasks that the algorithm carries out within each node. The active thread which pro-actively initiates a cache exchange once every cycle and the passive thread that waits for data-exchange requests (the cache consists in a routing table pointing to neighbor nodes of $EvAg_i$).

Algorithm 2 Newscast protocol in $EvAg_i$

```
Active Thread
loop
    wait one cycle
    $EvAg_j \Leftarrow$ Random selected node from $Cache_i$
    send $Cache_i$ to $EvAg_j$
    receive $Cache_j$ from $EvAg_j$
    $Cache_i \Leftarrow$ Aggregate ($Cache_i$,$Cache_j$)
end loop

Passive Thread
loop
    wait $Cache_j$ from $EvAg_j$
    send $Cache_i$ to $EvAg_j$
    $Cache_i \Leftarrow$ Aggregate ($Cache_i$,$Cache_j$)
end loop
```

Every cycle each $EvAg_i$ initiates a cache exchange. It uniformly selects at random a neighbor $EvAg_j$ from its $Cache_i$. Then $EvAg_i$ and $EvAg_j$ exchange their caches and merge them following an aggregation function. In this case, the aggregation consists of picking the freshest c items (i.e. c is the maximum degree of a node. In this paper $c = 40$) from $Cache_i \cup Cache_j$ and merging them into a single cache that will be replicated in $EvAg_i$ and $EvAg_j$.

Within this process, every EvAg behaves as a virtual node whose neighborhood is self-organized at a virtual level with independence of the physical network. In this paper, we conduct experiments following the ideal case in which every computing core hosts a single EvAg.

3 Experimental Setup

The experimental setup in this paper is based on the simulations performed in [8] on the scalability of the Evolvable Agent model when tackling trap functions [1]. In order to validate the model, we will try to reproduce such results in a parallel infrastructure.

3.1 Simulation settings

A trap function is a piecewise-linear function defined on unitation (the number of ones in a binary string). There are two distinct regions in search space, one leading to a global optimum and the other leading to the local optimum (see Figure 1). In general, a trap function is defined by the following equation:

$$trap(u(\overrightarrow{x})) = \begin{cases} \frac{a}{z}(z - u(\overrightarrow{x})), if \quad u(\overrightarrow{x}) \leq z \\ \\ \frac{b}{l-z}(u(\overrightarrow{x}) - z), \quad otherwise \end{cases} \tag{1}$$

where $u(\overrightarrow{x})$ is the unitation function, a is the local optimum, b is the global optimum, l is the problem size and z is a slope-change location separating the attraction basin of the two optima.

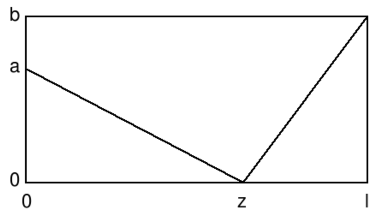

Fig. 1. Generalized *l-trap* function.

For the following experiments, a 3-trap function was designed with the following parameter values: $a = l - 1$; $b = l$; $z = l - 1$. With these settings, 3-trap lies in the region between deception and non-deception. Scalability tests were then performed by juxtaposing m trap functions and summing the fitness of each sub-function to obtain the total fitness.

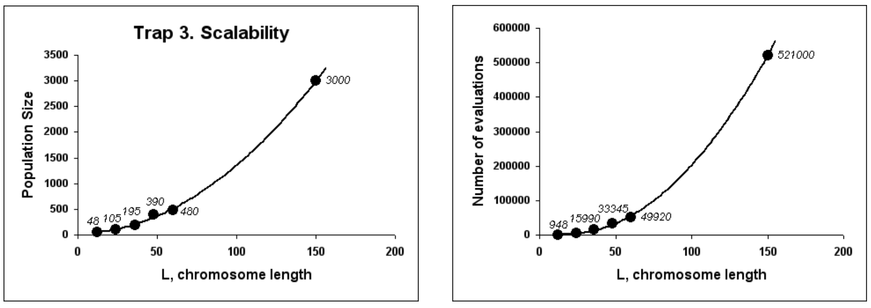

Fig. 2. Simulator estimated scalability of the Peer-to-Peer Evolutionary Algorithm [8] tackling different instances of the 3-trap problem [1]. On the left the estimated population sizes and the number of evaluations to solution on the right. Results are obtained for a selectorecombinative version of the algorithm (i.e. no mutation) and depicted as a function of the length of the chromosome, L.

The bisection method [12] was used for each size m to determine the optimal population size P, that is, the lowest P for which 98% of the runs solve the traps functions. To find it, mutation rate is set to 0, so as to search a minimum population size such that using random initialization it is able to provide enough building blocks to converge to the optimum without other mechanism besides recombination and selection.

Figure 2 depicts the simulation-based results for increasing problem instances of the 3-trap problem (lengths of the chromosomes are $L = 12, 24, 36, 48, 60, 150$). As the problem scales, the P2P EA requires of both, a larger population size and a larger number of evaluations, to guarantee that the optimal solution is found with a probability of 0.98.

3.2 Parallel version settings

In order to run parallel experiments, we are going to consider two of the problem instances from previous simulations. The first instance of a medium size, i.e. $L = 48$ bits, and the second with $L = 150$ bits. Table 1 provides the simulator-based results for both instances that will be used as parameter inputs in the parallel runs. They characterize the settings of the algorithm to find the optimum 98 out of 100 times, e.g. in order to find the optimum in the $L = 48$ instance, the algorithm requires a population size of 390 individuals and a maximum of 140 generations.

Table 1. Simulator-based results for the population size and the respective number of generations in order to find the problem optimum.

Instance	Population Size	Avg. n. of generations	Max. n. of generation
$L = 48$	390	85	140
$L = 150$	3000	173	250

The rest of parameter settings are summarized in Table 2.

Table 2. Parameters of the experiments

Trap instances	
Size of sub-function (k)	3
Individual length (L)	$48, 150$
GA settings	
Selection of Parents	Binary Tournament
Recombination	Uniform crossover, $p_c = 1.0$
Mutation	No mutation, $p_m = 0.0$

All experiments in this paper were conducted in the NEC Nehalem cluster at the HPC center of the University of Stuttgart (see `http://www.hlrs.de/`

`systems/platforms/nec-nehalem-cluster` for further details on the architecture). Here, it should be noted that P2P overlay networks behave independently of the underlying infrastructure they are running in, therefore, a cluster of homogeneous nodes can be consider a P2P system whenever it runs a P2P engine in every node. In that sense, using a cluster of homogeneous nodes has the advantage of simplifying the validation process. First, the side effects of asynchrony are minimized since every agent is scheduled by a single processor core running at the same frequency than the rest, and second, the lifetime and load of computers can be monitored so that we can ensure that there are no failures. Both effects, asynchrony and fault-tolerance to computer failures are left, therefore, as a future line of research.

4 Analysis of Results

In this section, we conduct two different sets of experiments. The first one focuses on verifying the results of the simulator in a real parallel platform. To that aim, a medium size instance of length $L = 48$ is considered. In a second experiment, we try to prove the massive scalability of the approach in terms of time speedups. Given that the goal of parallel Evolutionary Algorithms is to reduce times-to-solutions of expensive optimization problems, experiments were conducted on the largest problem instance of length $L = 150$.

4.1 Test-Case 1: Validating Results of the Simulator

In order to validate the results obtained by the simulator for the $L = 48$ instance, the P2P Evolutionary Algorithm was distributed in the NEC Nehalem cluster using a fine-grained parallelization in which every agent was scheduled in an independent thread, each running in its own processor. As described in [9], such settings stand for a worst-case scenario in which the fitness evaluations are computationally heavy. In advance, there are no restrictions limiting the number of agents per processor. Both algorithms (i.e. simulator-based algorithm and the parallel version) were equally parametrized and only differ on the population sizes P tested for the parallel approach.

Figure 3 depicts the average progress of the fitness convergence of the parallel runs for different population sizes $P = 50, 100, 200, 300, 400$. Given that the simulator predicts an optimal population size of $P = 390$, we aim at investigating the improvements on the fitness as the population size increases from $P = 50$ to $P = 400$. Results show that the smaller population sizes are not able to find the problem optimum (set to 48). However, for $P = 400$ the algorithm is able to track the optimal solution 8 times out of 10 as roughly estimated by the simulator (i.e. simulator actually predicts a success rate of 0.98 for $P = 390$). Taking into account the side effects of asynchrony and communications in the parallel version, we can conclude that the simulator-based results can be considered as a good estimate of the parallel performance of the algorithm.

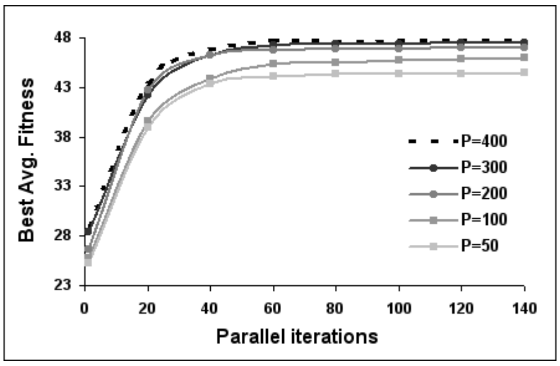

Fig. 3. Test-case 1. Best fitness convergence for the $L = 48$ instance using different population sizes (P). Results are averaged from 10 independent runs.

4.2 Test-Case 2: Testing the Massive Scalability of the Approach

In this experiment, we tackle the largest problem instance with a length of $L = 150$. The complexity of the instance is so high that the simulator estimates a population size of $P = 3000$ for the problem to be solved. However, and despite trap functions being algorithmically-complex problems (i.e. NP-hard), they are computationally lightweight. To emulate realistic time-consuming problems (e.g. the simulation guided optimization proposed by Ruiz et al. in [11] where the fitness function takes 6.5 seconds), we add a delay routine in every fitness evaluation that takes 16 seconds. Adding a delay routine aims reproducing heavy-loaded scenarios in which the ratio between communications and computation decreases. With these settings and according to the results in the simulator, the algorithm will require that 3000 individuals evolve during an average number of 173 generations to reach the optimum. In terms of time, that would translates into 100 days of sequential computation.

With the aim of reducing the time of convergence, the algorithm was parallelized using 3008 agents, each one running in a thread. The entire population was deployed in 188 computers, having 8 cores each and implementing hyper-threading with 2 threads per core. Note that the 8 extra individuals to the estimated 3000 are due to the composition of the architecture in which the parallelism extends to the microprocessor level –as McNairy and Bhatia describe in [10]– by adding several cores per processor and through hyper-threading technology.

Figure 4 depicts the convergence of the algorithm as a function of time. It shows how the algorithm is able to find optimality –the problem optimum is set to 150– after 2.5 hours of parallel processing which demonstrates that way the massive scalability of the approach.

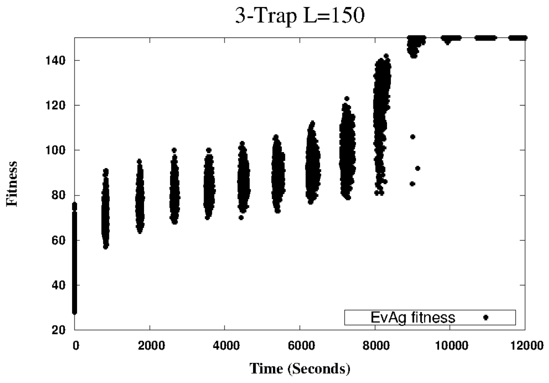

Fig. 4. Test-case 2. Fitness convergence for 3008 agents ($P = 3008$) in the $L = 150$ instance. The problem optimum is set to 150 and is found after 8900 seconds.

5 Conclusions

In this paper, we have conducted experiments for the validation of a Peer-to-Peer Evolutionary Algorithm in a cluster of homogeneous nodes. A common approach for designing such a kind of models is to use a simulation-driven experimentation given the difficulties of accessing a large amount of computers for testing. Therefore, model characterizations remain valid only under certain set of assumptions and the viability of the approaches is subject to the scope of the simulator.

In order to validate a model, we propose to reproduce results from simulations in a real-world system so that they can approach the predicted values. In that context, two simple experiments were conducted in a real infrastructure using a parallel version of the Peer-to-Peer Evolutionary Algorithm. The first experiment shows that the parallel version performs roughly the same than an equally-parametrized simulator-based run from which the validation of the model can be drawn. Specifically, the population size of the parallel version is adjusted to the same values of the simulations having an equivalent performance in fitness convergence. The second experiment focuses on determining the scalability of the parallel approach for large problem instances which, additionally, require of large population sizes. In this case, a massively parallel run is conducted in 3008 computing cores, each hosting an individual. The problem optimum is found after two and half hours of parallel execution in contrast to the estimate of one hundred days run if computed sequentially.

As future lines of work, we aim at investigating the parallel approach in a bigger set of scenarios taking into account the effects of heterogeneous computers on the algorithmic performance. We find that asynchrony, communication latencies and computer failures are the main issues to circumvent in order to deploy the algorithm in ad-hoc networks as they are Internet-based volunteer systems.

Acknowledgments

This work was supported by the Luxembourg FNR GreenIT Project (C09/IS/05), Spanish Ministry of Science Projects TIN2008-05941 and TIN2011-28627-C04 and Andalusian Regional Project P08-TIC-03903.

References

1. David H. Ackley. *A connectionist machine for genetic hillclimbing.* Kluwer Academic Publishers, Norwell, MA, USA, 1987.
2. David P. Anderson. Boinc: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, 2004.
3. M. Biazzini and A. Montresor. Gossiping de: A decentralized heuristic for function optimization in p2p networks. In *ICPADS'10*, pages 468 –475, 2010.
4. Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* SpringerVerlag, 2003.
5. Y. Guo, J. Cheng, Y. Cao, and Y. Lin. A novel multi-population cultural algorithm adopting knowledge migration. *Soft Comput.*, 15(5):897–905, 2011.
6. Márk Jelasity and Maarten van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, October 2002.
7. J.L.J. Laredo, P.A. Castillo, A.M. Mora, and J.J. Merelo. Exploring population structures for locally concurrent and massively parallel evolutionary algorithms. In *IEEE Congress on Evolutionary Computation (CEC2008), WCCI2008 Proceedings*, pages 2610–2617. IEEE Press, Hong Kong, June 2008.
8. J.L.J. Laredo, A. E. Eiben, Maarten van Steen, and Juan Julián Merelo Guervós. Evag: a scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2):227–246, 2010.
9. J.L.J. Laredo, D. Lombraña, F. Fernández de Vega, M.G. Arenas, and J.J. Merelo. A peer-to-peer approach to genetic programming. In *EuroGP*, volume 6621 of *Lecture Notes in Computer Science*, pages 108–117. Springer, 2011.
10. C. McNairy and R. Bhatia. Montecito: a dual-core, dual-thread itanium processor. *IEEE Micro*, 25(2):10–20, 2005.
11. P. Ruiz, B. Dorronsoro, G. Valentini, F. Pinel, and P. Bouvry. Optimisation of the enhanced distance based broadcasting protocol for manets. *Journal of Supercomputing*, 2011. To appear.
12. K. Sastry. Evaluation-relaxation schemes for genetic and evolutionary algorithms. Technical Report 2002004, University of Illinois at Urbana-Champaign, Urbana, IL., 2001.
13. I. Scriven, D. Ireland, A. Lewis, S. Mostaghim, and J. Branke. Asynchronous multiple objective particle swarm optimisation in unreliable distributed environments. In *IEEE Congress on Evolutionary Computation (CEC2008)*, 2008.
14. R. Steinmetz and K Wehrle. *Peer-to-Peer Systems and Applications*, chapter What is this Peer-to-Peer about?, pages 9–16. Springer, 2005.
15. D.J. Watts and S.H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.
16. W. R. M. U. K. Wickramasinghe, M. van Steen, and A. E. Eiben. Peer-to-peer evolutionary algorithms with adaptive autonomous selection. In *GECCO '07*, pages 1460–1467, New York, NY, USA, 2007. ACM Press.