Evolutionary Algorithms: Perspectives on the Evolution of Parallel Models

F. Fernández de Vega

Abstract This chapter discusses the inherent parallel nature of evolutionary algorithms, and the role this parallelism can take when implementing them on different hardware architectures. We show the interest in studying ephemeral behaviors that distributed computing resources may feature and some EA's self-properties of interest, such as the fault-tolerant nature that helps to fight the *churn* phenomenon. Moreover, interactive versions of EAs, which require distributed computing systems, allow to incorporate human based knowledge within the algorithm at different levels, providing new means for improving their computing capabilities while also requiring a proper analysis of human behavior under an EA framework. A proper understanding of ephemeral properties of hardware resources, human behavior in interactive applications and intrinsic parallel behaviors of population based algorithms will lead to significant improvements.

1 Introduction

Although evolutionary algorithms [1], and other population based algorithms, have been successfully applied to solving a wide number of optimization problems, researchers typically apply sequential version of the algorithms. Several reasons explain this traditional approach to software development, including the learning curve required to properly apply parallel models and libraries, and the wide number of available software tools that were developed in the traditional sequential approach. Although things are slightly changing, the literature is still dominated by sequential EAs.

Nevertheless, parallelism was soon recognized as an intrinsic property of EAs that works in the background even when a sequential version of the algorithm is run. The schema theorem, proposed by Holland in the seventies, was in charge of

F. Fernández de Vega (✉)
Centro Universitario de Mérida, Universidad de Extremadura,
Sta. Teresa Jornet, 38, 06800 Mérida (Badajoz), Spain
e-mail: fcofdez@unex.es

P. Novais et al. (eds.), *Intelligent Distributed Computing IX*,
Studies in Computational Intelligence 616,
DOI 10.1007/978-3-319-25017-5_2

explaining this inherent parallel property [2]. Although the reasoning is of interest for understanding how EAs can build solutions to problems, it doesn't allow to speed up the behavior of the algorithm: researchers have happily relied on this explanation for its intrinsic parallel behavior until hard real life problems have been faced. Only then, researchers have resorted to parallelism, when days, weeks or even months are consumed until a proper solution is found [3].

This chapter reviews different parallel models that have been proposed, how they can be deployed on different hardware architectures, and focuses in new properties that have been studied in the last few years, involving non reliable hardware resources as well as human interaction with the algorithms, showing that work ahead may provide new means for improving the performance of the algorithm.

This chapter is organized as follows: In Sect. 2, an overview of available parallel models is provided; Sect. 3 discusses the role distributed users may have on the algorithm; Finally, Sect. 4 describe our conclusions and paths for future improvement of parallel EAs.

2 Parallel Models Have Evolved

Embarrassingly parallel models were firstly propose as a way to quickly embody parallelism within EAs [4]. The easier incarnation of parallelism allows to simultaneously evaluate a number of individuals when hardware resources are available. We must remind that the standard evolutionary loop includes the computationally expensive evaluation of a number of individuals from the population, candidate solution to the problem at hand, followed by the crossover process that give rise to the new generation of individuals. Thus, the master-slave based model doesn't change the main algorithm, in charge of selecting parents for the next generation and applying genetic operators, being the fitness function the only one requiring a change. Given that fitness evaluation is typically the most time consuming part of the algorithm, and how easily a sequential implementation of an EA can be parallelized using this model, it quickly attracted attention. Thus the simplest parallel EA has been deployed and run on networks of transputers [5], clusters and grids [6], and more recently on GPUs [7] and clouds [8], and has probably been the most frequently used version of a parallel EA.

Nevertheless, researchers soon devised new ways for improving convergence properties, adding new functions to parallel models that in the end implied a deep change in the underlying algorithm and produced a change in the process of searching for solutions. Instead of evaluating single individuals in parallel, researchers decided to run the main algorithm over a number of them -a subpopulation- within each of the processors available, thus resulting in the *Island Model* [9].

Each of the subpopulation run the standard algorithm in the island model, and a new step, the migration, allows selected individuals to be exchanged among subpopulations -islands- with a given frequency. Thus the researcher must set up the value of some new parameters: island size, frequency of migration, number of migrating individuals, selection operator in charge of selecting migrants, discharging policy allowing to maintain the size of islands once new individuals arrive, etc. All these new parameters have already been widely studied, and its influence on the convergence process exposed for different flavors of EAs, including Genetic Algorithms and Genetic Programming [10]. The conclusions points out the benefit of migrating individuals, which helps to improve diversity in the subpopulations, thus helping to find better solutions, regardless of the time saved thanks to the parallel hardware infrastructure employed.

Yet, the island model is not the only one available for improving convergence properties of EAs as well as speeding up the finding of solutions. The cellular model is another possibility [11]. In this case, individuals from the population are distributed on a grid, so that interaction required when genetic operators must be applied only occur within a previously established neighborhood. This means that one individual can only interact with surrounding ones, which changes the way chromosome information spreads along the population [9]. Several authors have applied successfully this model borrowed from the cellular automata literature, although the implementation details make it more difficult to be adopted by researchers.

An interesting difference among the available parallel models, regarding hardware resources to be used and their properties, can be noticed: While for the island model, each of the subpopulations can run semi-isolated within each of the processors employed, and only a migrating step is required after a number of generations, the embarrassingly parallel model requires fitness values computed to be returned to the master in charge of applying each of the genetic operators, and this implies the sending of fitness values at least once per generation, which may be of importance or not depending on the time required to compute fitness values: shorter time to compute fitness value means worse performance of the algorithm, given that the latency of communications has a larger impact. Similarly, this is also something that must be taken into account when using the cellular model, which requires communication among adjacent individuals from the topological point of view that are run on different processors every time a crossover operation must be applied. Summarizing, high latencies will significantly deteriorates the speedup of both cellular and embarrassingly parallel model, even preventing them to compete in certain situations with the sequential version, while it will not hinder island model to properly finding solutions in shorter times.

In any case, communication libraries had to be adopted by researchers when implementing parallel EAs, such as classic PVM or MPI [12], when using clusters of computers; other different approaches can be considered when resorting to internet and grid frameworks. Even in this latest hardware infrastructures, interesting software packages allow to quickly deploy any algorithm on an easy to build desktop grid system, such as that based in the BOINC framework [13], which has opened up a world of possibilities for EAs. As we will see below, a proper study of the dynamics of this model has allowed to develop in the last decade new proposals for parallel EAs that benefit from the properties of the underlying communication model: in the area of Grid computing, the well known desktop grid model has been employed to run massively parallel evolutionary algorithms applied to real-world problems; On the other hand, P2P models have allowed to implement new agent based EAs that change the standard dynamics of the algorithm. Both models have changed the way we understand the algorithm, and have shed light into some of the properties that the new parallel models have unveiled.

2.1 Desktop Grids and Shrinking Population

When referring to Desktop Grid Computing, we consider a particular case of Grid technology where all of the computing resources are homogeneous: desktop personal computers. Given that all of the computers are based on the same hardware architecture, and basically the same operating system, the grid system significantly simplifies the way parallel algorithms can be deployed on the network of computers: a single version of the algorithm must be implemented (linux—i386, for instance), instead of considering all of the potential hardware architectures and operating systems combinations that are present in a more traditional Grid infrastructure. Moreover, available software tools, such as BOINC [13], allow to easily manage the desktop grid infrastructure, allowing researchers to only focus on the Evolutionary Algorithm to be deployed. The basic desktop-grid model follows the master-slave approach, and is well suited to embarrassingly parallel EAs: typically desktop grids are deployed within institutions, and communication latencies are thus under control.

The simplicity of desktop grids, has allowed researchers to face hard real life problems: packages of individuals are distributed every generation along the available computing nodes, allowing researchers to manage large population sizes for real life problems requiring long fitness evaluation time [3]. Thus, the model was shown to perfectly work on desktop grids provided by the researchers. The surprise came when the model was applied using computing resources provided by volunteers under the well known volunteer computing model [14].

Volunteer computing is based on the desktop grid model, and desktop computers are provided by a number of volunteers connected to internet that are willing to contribute to a scientific project. Thus, the scientist is typically in charge of setting up the master node, where all of the computing tasks are established, and then, the volunteers connect to the server and agree to provide computing resources for each of the tasks. The model has worked fine for decades, being the *Seti@home* project one of the best known with several million volunteers providing resources [13]. Nevertheless, and given that resources are switched on and off according to volunteers' needs, nobody can assure the time a computing node will be alive, and whether a specific task submitted will be thus completed on time. The dynamic of the volunteer computing infrastructure is thus characterize by this well known *churn* phenomena, and scientists interested in profiting from volunteers must encode a number of fault-tolerant techniques if they want their project to finish properly [15]. But this inherent property, churn, was recently considered from a different perspective in the context of EAs, specifically from the point of view of Genetic Programming (GP) with interesting results.

2.1.1 Distributed GP and the Churn Phenomena

One of the main flavors of Evolutionary Algorithms is GP, popularized by Koza in the nineties [16] as a mean for automatic programming. One of the main features of GP, considered as a problem, is the *bloat* phenomenon: given that variable size chromosomes are employed in GP, the evolution dynamics make chromosomes to grow out of control, which implies an increase in memory consumption and usually time required to evaluate longer individuals. This behavior in GP has lead researchers to focus on chromosome growth [17], an although a number of techniques have already been propose to fight it, we think future research on the topic will show how this behavior may find a strong connection to improvements on the way GP is run in parallel systems: a natural load balancing technique could make use of individual differences to run them on different computing elements, as well as applying genetic operations as soon as individuals have been evaluated, thus favoring shorter computing-times, which typically implies smaller sizes. Thus parallel systems could naturally fight bloat. We must also bear in mind, the difficulty for properly running GP on GPUs, which has been an issue in latest years. Although some proposals have already been published, we still feel there is room for improvement, considering main differences among GP and other EAs.

Among the different techniques introduced in the literature for the last decades, the *plague* operator was proposed to remove progressively individuals from the population as a countermeasure for the bloat phenomenon, thus maintaining the amount of memory required to manage the population: individuals' growth is fought with a shrinking population [18]. Since then, different studies have shown the interest of considering variable size populations for GP and other EAs, which require a self-analyzing capability of the algorithm to know when the size must be changed. But a deeper analysis allowed to recently see the connection between this idea and the churn phenomenon in volunteer computing infrastructures: if instead of removing selected individuals, we consider churn phenomenon as the component in charge of randomly discarding individuals along the run of a GP experiment in a volunteer computing environment, we have a quite similar experiment, the only difference being the way individuals are selected.

In the last few years, a number of experiments have tested this approach showing that not only Genetic Programming, but also Genetic Algorithms are fault tolerant, and can cope with up to 30 % of population decrease without applying any particular fault tolerance technique. This has opened up the possibility of running distributed versions of the algorithm in non reliable distributed computing resources with results whose quality does not significantly deteriorates [19], boosting a line of research that focuses on self-properties of EAs in the context of parallel and distributed systems. The experiments have thus shown that other network topologies and communication models can also be employed within this context, such as Peer to Peer networks.

2.2 *To Peer or Not to Peer*

Peer to Peer models (P2P), have been recently studied in the context of EAs by Laredo et al. [20]. One of the main features of the model, is the lack of a central node, both in terms of hardware resources and in the main algorithm to be run. The model instead relies in a number of software agents with capability for establishing connections with surrounding agents, being them run on the same or different computing element.

P2P models require specific communication protocols, that allow agents to know where other agents are located, and from the algorithmic point of view, also requires changes when a task must be performed. If we consider EAs in a P2P context, we will see each of the individuals in the population as an agent. No central storage location for the population exist anymore, nor a single algorithm applying genetic operations to the individuals. Instead, each of the agents must include the capability to interact with other agents, individuals, so that they can crossover and create offspring. New software tools allow to deploy EAs using P2P models, and some of them rely on web browsers to run the genetic operations, including fitness evaluations [21]. The fault tolerance nature of these agent based models have also been studied reaching similar conclusions as with its volunteer model counterpart [19]. But one of the main interests now, is the possibility of using web browsers, and also user interaction, as the underlying system where the algorithms are run. The possibility of allowing users to interact with the algorithm through a web browser, in the context of P2P EAs but also when using the master-slave approach, and the churn properties featured are allowing to explore new properties for distributed EAs.

3 Interactive EAs, Ephemeral Properties and the Role of Users

Although the possibility for allowing users to interact with EAs was soon recognized as a means for fitness evaluation, similarly as how volunteer computing based projects invite users to collaborate by performing visual analysis of images [21], in the context of EAs, the interaction has been exclusively used as a way for aesthetic assessment in Evolutionary Art. Thus interactive EAs are directly related to Evolutionary Art and Design, and typically the interaction has been facilitated through web based applications in charge of displaying each of the *individuals* in the population, that are then rated by the users, so that fitness values provided are employed to apply selection, crossover, etc. Users are thus contributing not only with fitness values, but also with hardware resources to run the user interface, one of the main part of the algorithm, and are therefore prone to the same kind of problems that were previously described in the context of volunteer computing and P2P environments.

Only recently, new software tools have been developed trying to generalize the model allowing users to both run and/or interact with EAs through the web browser, such as Evospace and Evospace-i agent based software models that connect through web browsers and allow to face any kind of problem by means of Evolutionary Algorithms [8]. The dynamics of the underlying model feature some of the ephemeral properties that naturally arise in an agent based model, and have already been studied with satisfactory results [15]. We will focus now in one component of these latest distributed models that are increasingly attracting attention: the user.

3.1 Distributed Users

The fact that users collaborating with interactive EAs, deployed through the web, are part of the algorithm changes the way we understand distributed EAs. On the first hand, users may visit a website but their collaboration is not guaranteed: in order for the evolutionary algorithm to progress, users must get involved in the experiment. Similarly as with volunteer computing experiments, the scientists must properly provide information of interest that attract users attention. On the other hand, and given that usually web browser must remain open with the application running while the user is devoting attention to other tasks, the experiment must keep user interest to collaborate and donate both computing resources and their time. Finally, when repetitive actions are required by the user, some kind of reward may be necessary if we want to fight the problem of users fatigue. These are some open problems in the area, and although efforts are applied trying to model users interaction that may in the future reduce the number of times an action is required from the user, we still lack a general solution to that problem [22].

Also, the number of users to be involved in a given experiment and also the way they interact should be adjusted: although typically users are simply in charge of rating individuals, different possibilities could be also adopted, such as asking users to select the parents for a crossover operation, so that indirectly a fitness evaluation is performed every time new children is generated. As we will show below, this later approach has been recently adopted in the context of *unplugged evolutionary algorithms* [23], but it is not still a common method.

Therefore, the actual influence of users in interactive EA experiments still allow for deeper studies, and a number of questions remain to be explored: is it possible to allow users to perform other operations different from fitness evaluations? What are the main reasons that lead a user to apply a specific rate to a given individual? Is it useful to allow different users to rate the same individual? What is the situation in the context of evolutionary art? These and other questions are leading efforts in the area, and one of the most recently proposed approaches is the Unplugged Evolutionary Algorithm.

3.2 Unplugged EAs

The idea behind the model arise from the interactive version of the EA: human beings are in charge of performing fitness evaluations. The question is: is it possible to delegate *all* of the operations to human beings? In the context of Evolutionary Art, the idea of making artists to perform the whole evolutionary algorithms tries to analyze the creative model when applied by a team of artists: they apply every operation required for an evolutionary algorithm so that no computer is required in the experiment.

Thus, a team of five artists developed an artistic experiment: each of the artist was in charge of producing an artwork every week by applying any kind of crossover and mutation over two works of their colleagues produced the week before. This way, instead of explicitly asking for fitness values -rates- for each of the paintings, the artists introduce an indirect fitness evaluation: only the two preferred works are given best rates and selected as an inspiration source when producing offspring. After ten weeks of work, a collective art work was produced and interesting information on the operations applied were described within forms provided to artists [24].

The analysis of the work gives us some clues for a better understanding of the creative processes developed by human artists, such as information on the key elements when applying crossover or mutation. For instance, artists always perceive a story within each figurative work, that may lead mutation operations towards a new work. Figurative work is typically preferred, instead of abstract works that are usually the output of evolutionary art experiments [23]. Yet, is not easy to foresee how some of the concepts learnt can in the future be incorporated into software tools in charge of producing human-like art.

On the other hand, if we want to fully emulate the creative process developed by human artists, a possible way to future improvement should include studying audience response to the work, including audience understanding of the genetic operations developed. Given the need of an audience when an art work is exhibited, audiences should be somehow included in the Evolutionary Art loop, being part of experimental research, and artworks should be exhibited in art museums and galleries.

4 Conclusions

This chapter presents an overview of latest attempts to parallelize Evolutionary Algorithm considering different points of view. On the one hand we have reviewed the models that have arisen in the last decade, such as those based on agents making use of Desktop grids and P2P infrastructures; on the other hand we have seen new paths that are being explored when distributed users are included as part of the parallel versions of the algorithm, particularly when art and creativity are pursued.

This review has led us to a number of questions that show paths towards future improvements on the way we understand and apply parallel and distributed versions of the Evolutionary Algorithms, that may be summarized as follows: (i) a proper understanding of the dynamics of algorithms employing variable size chromosomes, such as GP, as well as employing self-properties that allow to be aware of individual-size dynamics may make it easier to profit from parallel infrastructures, including GPUs as well as those characterized by ephimeral properties, such as desktop grids. (ii) the proper understanding of users interaction dynamics in the context of unplugged evolutionary algorithms may provide clues to improving how distributed interactive evolutionary algorithms are applied when facing evolutionary art project. It can make it easier for scientists to atract users and also avoid users' fatigue, as well as provide a better understanding of creative process that helps in the future to improve computer assited creativity.

Acknowledgments This work is supported by EU Merie Curie actions, FP7-PEOPLE-2013-IRSES, Grant 612689 ACoBSEC; MINECO project EphemeCH (TIN2014-56494-C4-P) and Gobierno de Extremadura,Consejería de Economía-Comercio e Innovación y FEDER, proyect GRU10029.

References

1. Back, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, New York (1996)
2. Bertoni, A., Dorigo, M.: Implicit parallelism in genetic algorithms. Artif. Intell. **61**(2), 307–314 (1993)
3. González, D.L., de Vega, F.F., Trujillo, L., Olague, G., Araujo, L., Castillo, P., Sharman, K.: Increasing gp computing power for free via desktop grid computing and virtualization. In 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, IEEE, pp. 419–423 (2009)
4. Cantu-Paz, E.: Efficient and accurate parallel genetic algorithms. Springer (2000)
5. Andre, D., Koza, J.R.: Parallel genetic programming: a scalable implementation using the transputer network architecture. Advances in Genetic Programming, pp. 317–337. MIT Press, Cambridge (1996)
6. Lim, D., Ong, Y.S., Jin, Y., Sendhoff, B., Lee, B.S.: Efficient hierarchical parallel genetic algorithms using grid computing. Future Gener. Comput. Syst. **23**(4), 658–670 (2007)
7. Wong, M.L., Wong, T.T., Fok, K.L.: Parallel evolutionary algorithms on graphics processing unit. In The 2005 IEEE Congress on Evolutionary Computation, IEEE, vol. 3, pp. 2286–2293 September 2005
8. García-Valdez, M., Trujillo, L., Merelo, J.J., de Vega, F.F., Olague, G.: The EvoSpace model for pool-based evolutionary algorithms. J. Grid Comput., 1–21 (2014)
9. Tomassini, M.: Spatially Structured Evolutionary Algorithms. Springer, Berlin (2005)
10. Fernández, F., Tomassini, M., Vanneschi, L.: An empirical study of multipopulation genetic programming. Genet. Program. Evolvable Mach. **4**(1), 21–51 (2003)
11. Folino, G., Pizzuti, C., Spezzano, G.: A Cellular Genetic Programming Approach to Classification. In GECCO, pp. 1015–1020, July 1999
12. Fernández, F., Tomassini, M., Vanneschi, L., Bucher, L.: A distributed computing environment for genetic programming using MPI. Recent Advances in Parallel Virtual Machine and Message Passing Interface, pp. 322–329. Springer, Berlin (2000)

13. Anderson, D. P. Boinc: A system for public-resource computing and storage. In Grid Computing. 2004. Proceedings. Fifth IEEE/ACM International Workshop on (pp. 4–10). IEEE
14. Cole, N., Desell, T., González, D.L., de Vega, F.F., Magdon-Ismail, M., Newberg, H., Varela, C.: Evolutionary algorithms on volunteer computing platforms: the milkyway@ home project. Parallel and Distributed Computational Intelligence, pp. 63–90. Springer, Berlin (2010)
15. González, D.L., Laredo, J.L.J., de Vega, F.F., Guervós, J.J.M.: Characterizing fault-tolerance of genetic algorithms in desktop grid systems. Evolutionary Computation in Combinatorial Optimization, pp. 131–142. Springer, Berlin (2010)
16. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1. MIT press, Cambridge (1992)
17. Alfaro-Cid, E., Merelo, J.J., de Vega, F.F., Esparcia-Alcázar, A.I., Sharman, K.: Bloat control operators and diversity in genetic programming: a comparative study. Evol. Comput. **18**(2), 305–332 (2010)
18. Fernandez, F., Vanneschi, L., Tomassini, M.: The effect of plagues in genetic programming: a study of variable-size populations. Genetic Programming, pp. 317–326. Springer, Berlin (2003)
19. Laredo, J.J., Bouvry, P., González, D.L., de Vega, F.F., Arenas, M.G., Merelo, J.J., Fernandes, C.M.: Designing robust volunteer-based evolutionary algorithms. Genet. Program. Evolvable Mach. **15**(3), 221–244 (2014)
20. Laredo, J.L.J., Eiben, A.E., van Steen, M., Castillo, P.A., Mora, A.M., Merelo, J.J.: P2P evolutionary algorithms: A suitable approach for tackling large instances in hard optimization problems. Euro-Par 2008-Parallel Processing, pp. 622–631. Springer, Berlin (2008)
21. Secretan, J., Beato, N., D Ambrosio, D.B., Rodriguez, A., Campbell, A., Stanley, K.O.: Picbreeder: evolving pictures collaboratively online. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, pp. 1759–1768 (2008)
22. Frade, M., Fernandez de Vega, F., Cotta, C. (2012). Automatic evolution of programs for procedural generation of terrains for video games: accessibility and edge length constraints
23. de Fernandez Vega, F., Cruz, C., Navarro, L., Hernández, P., Gallego, T., Espada, L.: Unplugging evolutionary algorithms: an experiment on human-algorithmic creativity. Genet. Program. Evolvable Mach. **15**(4), 379–402 (2014)
24. Fernendez de Vega, F., Navarro, L., Cruz, C., Chavez, F., Espada, L., Hernandez, P., Gallego, T.: Unplugging evolutionary algorithms: on the sources of novelty and creativity. In IEEE Congress on Evolutionary Computation (CEC), pp. 2856–2863 (2013)