Randomized Parameter Settings for Heterogeneous Workers in a Pool-Based Evolutionary Algorithm

Mario García-Valdez[1], Leonardo Trujillo[1],
Juan Julián Merelo-Guérvos[2], and Francisco Fernández-de-Vega[3]

[1] Instituto Tecnológico de Tijuana, Tijuana B.C., México
[2] Universidad de Granada, Granada, Spain
[3] Grupo de Evolución Artificial, Universidad de Extremadura, Mérida, Spain
{mario,leonardo.trujillo}@tectijuana.edu.mx
jmerelo@geneura.ugr.es, fcofdez@unex.es

Abstract. Recently, several Pool-based Evolutionary Algorithms (PEAs) have been proposed, that asynchronously distribute an evolutionary search among heterogeneous devices, using controlled nodes and nodes outside the local network, through web browsers or cloud services. In PEAs, the population is stored in a shared pool, while distributed processes called workers execute the actual evolutionary search. This approach allows researchers to use low cost computational power that might not be available otherwise. On the other hand, it introduces the challenge of leveraging the computing power of heterogeneous and unreliable resources. The heterogeneity of the system suggests that using a heterogeneous parametrization might be a better option, so the goal of this work is to test such a scheme. In particular, this paper evaluates the strategy proposed by Gong and Fukunaga for the Island-Model, which assigns random control parameter values to each worker. Experiments were conducted to assess the viability of this strategy on pool-based EAs using benchmark problems and the EvoSpace framework. The results suggest that the approach can yield results which are competitive with other parametrization approaches, without requiring any form of experimental tuning.

Keywords: Pool-based Evolutionary Algorithms, Distributed Evolutionary Algorithms, Algorithm Parametrization.

1 Introduction

Evolutionary computation (EC) research has allowed scientists and engineers from many fields to understand the power of the natural search process described by the biological theory of Neo-Darwinian evolution [13]. Inspired in biological evolution, EC researchers have developed a variety of search and optimization algorithms [6]. While EAs are inspired by evolution, they mostly follow an abstract model of the natural process. For instance, one aspect that is omitted from most EAs is the open-ended nature of evolution, in practice EAs are used to solve problems with well defined objectives, while natural evolution is an adaptive process without an a priori goal or purpose. Biological evolution is also an intrinsically parallel, distributed and asynchronous process,

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 702–710, 2014.

undoubtedly important features that have allowed evolution to produce impressive results throughout nature. However, some of these features are not trivially included into standard EAs [1], which are mostly coded as sequential and synchronous algorithms [6]. For instance, a large body of work exists in EA parallelization, a comprehensive introduction can be found in [1]. However, distributed and asynchronous EAs have started to become common only recently. In particular, recent trends in information technology have opened new lines of future development for EC research.

Today, computing resources range from personal computers and smart-devices to massive data centers. These resources are easily accessible through Internet technologies, such as cloud computing, peer-to-peer networks and web environments. Several EAs have been proposed that distribute the evolutionary process among heterogeneous devices, not only among controlled nodes within an in-house cluster or grid, but also to others outside the data center, in web browsers, smart phones or cloud services. This reach out approach allows researchers to use low cost computational power that would not be available otherwise, but on the other hand, have the challenge to manage heterogeneous and mostly unreliable computing resources. In particular, we are interested in systems that follow a pool-based approach, where the search process is conducted by a collection, of possibly heterogeneous, collaborating processes using a shared repository or population pool. We will refer to such algorithms as pool-based EAs or PEAs, which are intrinsically parallel, distributed and asynchronous.

Despite promising results, PEAs present several challenges. From a technological perspective, lost connections, low bandwidth, abandoned work, security and privacy are all important issues. This work, however, focuses on algorithm parametrization, a common issue with most EAs that is amplified in a PEA. In general, EAs are sometimes criticized by the large number of parameters they posses, that for real world problems need to be tuned empirically or require additional heuristic processes to be included into the search to adjust the parameters automatically [15, 16]. In the case of a PEAs, this issue is magnified since the underlying system architecture adds several degrees of freedom to the search process, with unknown interactions.

This work studies the recently proposed EvoSpace system, a framework to develop PEAs using an heterogeneous collection of possibly unreliable computing resources. Despite promising initial results [9–11, 21], research devoted to EvoSpace has not addressed the parametrization issue. Therefore, in this paper the recent approach called Randomized Parameter Setting Strategy (RPSS) [12, 20] is tested with EvoSpace. The idea behind RPSS is that in a distributed EA, algorithm parametrization may be completely skipped and still conduct a successful search. Results suggest that when the number of distributed process is large enough, algorithm parameters can be set randomly and still achieve good results. However, work on RPSS has only focused on the well-known Island Model for EAs, a distributed but synchronous system. On the other hand, the goal of this work is to evaluate RPSS on a PEA implemented through EvoSpace, that does not have a fixed population structure.

The remainder of the paper proceeds as follows. Section 2 reviews related work. Section 3 briefly describes the EvoSpace framework and provides implementation details. The problem statement and experimental work are presented in Section 4. Finally, concluding remarks are outlined in Section 5.

2 Related Work

In terms of parallelizing EAs, a large body of work has been developed, covering many different EAs, with important practical and theoretical results [1]. However, only recently has the topic of distributed systems been explored. For instance, Fernández et al. [8] use the Berkeley Open Infrastructure for Network Computing (BOINC) to distribute EA runs across an heterogeneous network of volunteer computers using virtual machines. However, such a system, as well as most distributed and parallel systems, does not follow the PEA approach studied in the current paper. In general, a pool-based system employs a central repository where the evolving population is stored. Distributed clients interact with the pool, performing some or all of the basic EA processes (selection, genetic operators, survival). For example, Smaoui Feki et al. [19] uses BOINC redundancy to deal with the volatility of computing nodes. In that work each BOINC work unit consisted of a fitness evaluation task and multiple replicas were produced and sent to different clients. Merelo et al. [17] developed a Javascript PEA that distributes the evolutionary process over web browsers, this provides the added advantage of not requiring additional software installations on client machines. Other similar cloud-based solutions are based on a global queue of tasks and a Map-Reduce implementation [5, 7, 18]. The current work, however, focuses on the EvoSpace presented in [9–11, 21] and summarized in Section 3.

One of the main problems with implementing successful EAs is parameter tuning [16], of particular importance in real-world scenarios, where usually there is little prior insights regarding what might be the best configuration for an EA, especially if the intent is to use it as a black-box optimizer; a comprehensive survey on this topic is given in [16]. Indeed, this problem has received a growing level of interest in recent years, as evidenced by the Self-Search track at GECCO for instance, where the aim is to develop self-adaptive or auto-tuning systems that reduce the amount of human intervention that might be required before performing an EA-based search or optimization.

A noteworthy contribution in [16] is the chapter by Cantú Paz that tackles the problem of deriving theoretical models of the effects of key EA parameters, such as population size and migration schemes in Island-Model EAs (IMEA) [2]. However, it does not cover the effects of all possible parameters, or the specific intricacies of a PEA algorithm. Here, we would stress some important differences between PEAs and IMEAs. First, an IMEA presents a fixed topological structure, with a predefined interaction protocol among each evolving population, this leads to a coordinated, or even synchronized, interaction between the islands. On the other hand, a PEA does not include such constraints, which means that the interactions between workers is much less structured or controlled. Second, in an IMEA each island represents an individual evolutionary process, sharing some of the same dynamics as standard EAs. In a PEA, however, only a single centralized population exists, samples of which are distributed across workers, but ultimately combined once again in the centralized pool. Therefore, some of the well-known insights derived from IMEA research (regarding, for example, migration policies) are not necessarily relevant in the PEA framework. Therefore, new parametrization approaches must be explored.

Several parameter tuning methodologies for EAs are presented in [16], that can substantially reduce the computational cost when compared with an exhaustive search in

parameter space. However, these methods focus on standard EAs, based on serial and synchronous searchers. On the other hand, recent works [12, 20] have shown a promising simpler alternative to tune EAs that employ multiple evolving populations. In particular, the RPSS approach proposed in [12] which quite intriguing given its simplicity. First, consider the original configuration studied in [12, 20], an IMEA where a set of N separate populations, or demes, run semi-isolated evolutionary processes, organized using a particular neighborhood structure, such as ring or a random graph. Each deme is not totally isolated, since after a certain amount of time (generations or function evaluation) a set of individuals is exchanged between neighboring demes, a process known as migration. Obviously, drastically increasing the number of system parameters makes sweeping parameter space computationally unfeasible, since it is not possible to assume that the best configuration is an homogeneous system where all demes share the same parametrization. Moreover, the additional complexity of the Island Model incorporates additional degrees of freedom that must be tuned before performing a run. Such a tuning task can become overwhelming, particularly if the number of islands is large. Therefore, the proposal in [12] is to set the parameter values randomly, without a tuning or self-adaptive process whatsoever. The RPSS approach is to set the parameters of each deme randomly at the beginning of the run, a very simple and apparently naive approach. Nevertheless, results reported in [12, 20] show promise, achieving competitive results while substantially reducing the amount of effort required to tune the system (the approach only requires the user to specify a range of valid values for each parameter).

3 EvoSpace

EvoSpace is based on the tuple space model [11], consisting of two main components (see figure 1): (i) the EvoSpace container that stores the population and (ii) EvoWorkers, which execute the actual evolutionary process, while EvoSpace acts only as a population repository. In a basic configuration, EvoWorkers pull a small random subset of the population, and use it as the initial population for a local EA executed on the client machine. Afterward, the evolved population from each EvoWorker is returned to the EvoSpace container. When individuals are pulled from the container they remain in a phantom state, they cannot be pulled again but they are not deleted. Only if the EvoWorker returns new individuals, are the phantom individuals deleted. If the EvoSpace container is at risk of starvation or when a time-out occurs, phantom individuals are reinserted and made available again. This can be done because a copy of each sample is stored in a priority queue, used to re-insert the sample to the central population; similar to games where characters are re-spawned. In the experiments conducted in this work re-insertion occurs when the population size is below a certain threshold. Figure 1 illustrates the main EvoSpace components.

The population of an EA is stored in-memory using the key-value database Redis, chosen because it provides a hash based implementation of sets and queues which are natural data structures for a PEA model. EvoSpace is implemented as a python module and exposed as a web service using Cherrypy. The EvoSpace modules are freely available with a Simplified BSD License at `https://github.com/mariosky/EvoSpace`. The EvoSpace system is deployed using Heroku, a multi-language Platform-as-a-Service

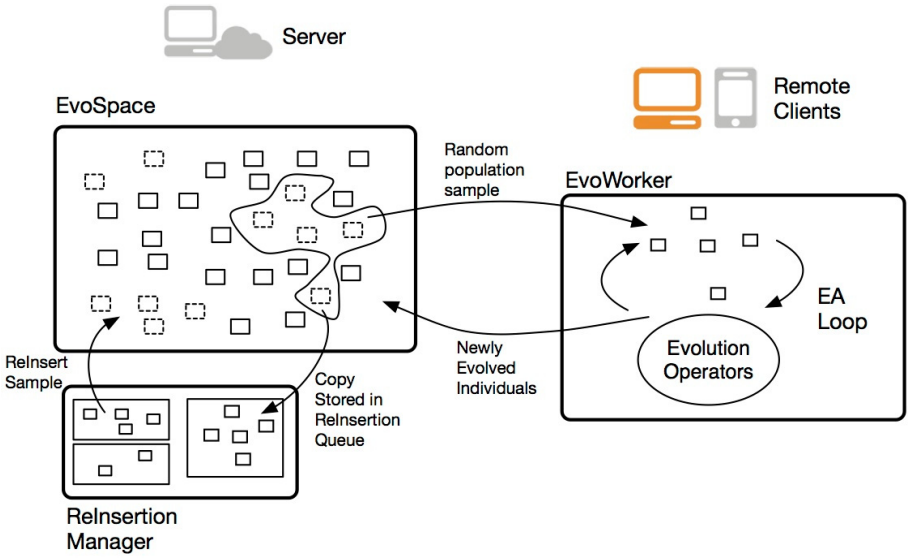


Fig. 1. Main components and dataflow within EvoSpace

(PaaS). The basic unit of composition on Heroku is a lightweight container running a single user-specified process. These containers, which they call *dynos*, can include web (only these can receive `http` requests) and worker processes (including systems used for database and queuing, for instance). Once deployed the web process can be scaled up by assigning more dynos; in our case and in the more demanding configurations of our experiments, the web process was scaled to 20 dynos. Instructions and code for deployment is available at `https://github.com/mariosky/EvoSpace`. For the experiments carried out for this paper, EvoSpace workers are distributed using the PiCloud PaaS.

4 Problem Statement and Experimental Work

As stated before, one of the main practical issues with EAs is parameter tuning. Following [12], the proposal of the current work is to apply the RPSS approach to a PEA developed with EvoSpace. However, before turning to the experimental work, lets highlight the main differences between a PEA and the Island Model studied in [12, 20]. First, the Island Model is a synchronous EA, while it implements a higher level of parallelization than a normal EA, and is amenable to distributed implementations, it still relies on a synchronized system to perform migration events. Second, the PEA approach based on EvoWorkers can be implemented as a more heterogeneous system than the Island Model, since new EvoWorkers can be added or removed dynamically. In particular, the sample (population) size and number of generations executed by each EvoWorker can be different, since synchronized migrations do not take place. Notice that in EvoSpace, there is no explicit migration process, on the other hand EvoWorkers exchange population members through the centralized pool. These differences could be important

Table 1. Ranges for each EvoWorker parameter

Parameter	Range
Crossover probability	[0,1]
Mutation probability	[0,1]
Sample Size	[12,24]
Generations	[5,30]

regarding the applicability of RPSS on an EvoSpace PEA, which is evaluated in the experimental work.

Therefore, the goal of this paper is to evaluate RPSS on a PEA developed over EvoSpace, in particular a genetic algorithm (GA). In other words, to determine if a random configuration for each of the n EvoWorkers that collaborate on a given run can achieve competitive results. The parameters considered are: 1) crossover probability; 2) mutation probability; 3) sample size; and 4) number of generations (executed locally in each EvoWorker). The valid ranges established for each parameter are summarized in Table 1.

To gauge the effectiveness of RPSS on a PEA, it is compared with three different parametrization strategies, similar to what is done in [12, 20]. All methods are compared based on average performance over a set of runs. First, the simplest approach consists on setting all of the EvoWorker parameters homogeneously. To do this, 200 random parametrizations are created, based on the ranges established in Table 1. The average performance of these runs characterizes the random-homogeneous parametrization, denoted Average-Homogeneous. From these runs, the best configuration is chosen, the one that achieved the best results, and then 20 independent runs are carried out, this method is called Best-Homogeneous [1]. Finally, the random-heterogeneous-parametrization is considered, where the parameters of each worker are set independently at random at the beginning of each run; 20 independent runs are performed, the method is denoted as Average-Heterogeneous.

4.1 Benchmark

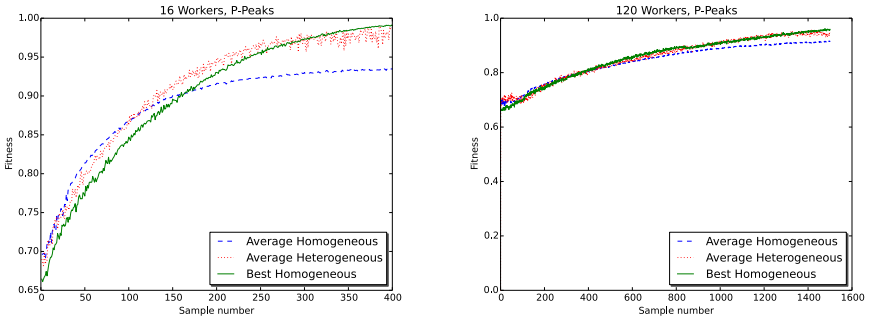
The algorithms are evaluated using the P-Peaks generator of multimodal problems proposed by De Jong et al. [3]. A P-Peaks instance is created by generating a set of P random N-bit strings, which represent the location of the P peaks in the space. To evaluate an arbitrary bit string $\mathbf{x}$ first locate the nearest peak (in Hamming space). Then the fitness of the bit string is the number of bits the string has in common with that nearest peak, divided by N. The optimum fitness for an individual is 1. This particular problem generator is a generalization of the P-peak problems introduced in [4], defined by

$$f_{P-PEAKS}(\mathbf{x}) = \frac{1}{N} \max_{i=1}^{P} \{N - hamming(\mathbf{x}, Peak_i)\} \ . \tag{1}$$

[1] This is a very naive approach to choose the best possible configuration, with much more comprehensive approaches reviewed in [16]. However, here we use the Best-Homogeneous approach for direct comparison with [12, 20].

Table 2. GA configuration for each benchmark problem

Feature	P-Peaks
Crossover (probability)	Two Points (0.7)
Mutation (probability)	Flip Bit (0.2)
Selection	Tournament (size=4)
Variable range	$\{0, 1\}$
Survival	Elitist (Keep-Best)
Individuals in the Pool	300,1000 (16,120 workers)



(a) 16 EvoWorkers (b) 120 EvoWorkers

Fig. 2. Convergence plots for the P-Peaks with 16 (a) and 120 (b) EvoWorkers

A large number of peaks induces a time-consuming search, which is convenient since in order to justify a distributed EA implementation, the cost of computing fitness has to be significantly larger than the implicit communication costs over the network or Cloud. However, according to Kennedy and Spears [14] the length of the string being optimized has a greater effect on the difficulty of the search.

4.2 Experimental Set-up and Results

Experiments are carried out using a different number N of EvoWorkers to solve the benchmark problem. The first group of runs are done with $N = 16$ EvoWorkers, and the second with $N = 120$. Based on [12, 20], it is assumed that with an increased number of workers the RPSS approach should achieve relatively better results, much closer to the Best-Homogeneous configuration. This is particularly important, since increasing the number of EvoWorkers greatly magnifies the dimensionality of the tuning problem. Results are summarized by tracking how the best solution varies with respect to the total number of samples taken from the EvoSpace pool of individuals. These results are presented in Figure 2, where the average performance for each of the three methods evaluated here.

First, for the P-Peaks problem with 16 EvoWorkers we can see a clear trend, the random Heterogeneous configuration is very similar with the best homogeneous configuration, depicted in Figure 2(a). This is a promising initial observation, since the

heterogeneous configuration did not require any parameter tuning, while the best homogeneous configuration is chosen from a set of 200 runs. Moreover, we see that using an homogeneous configuration with random values achieves noticeably inferior performance. When the number of EvoWorkers is increased, shown in Figure 2(b), a similar trend appears, however the differences among the algorithms is reduced. Nevertheless, it is obvious that using a random heterogeneous parametrization can be used as an off-the shelf approach on this problem.

5 Conclusions and Further Work

This paper presents an evaluation of the RPSS parametrization approach on a pool-based EA developed over the EvoSpace system. The basic idea, which is quite simple, is to randomly set the parameter values of each EvoWorker, that connect to the central population pool and perform an independent evolutionary search on a sampled set of individuals. While PEAs developed over EvoSpace have been studied before with good initial results, they suffer from the fact that they have a large number of degrees-of-freedom, requiring extensive parameter tuning. However, using the RPPS approach, it seems that a PEA can be executed successfully without any form of parameter tuning, achieving comparable results to standard homogeneous parametrizations. Future work will focus on exploring the limits of the approach using a more diverse set of benchmark problems, as well as other EA techniques, such as genetic programming or particle swarm optimization.

Acknowledgements. Funding provided by CONACYT (Mexico) Project No. 29537 from the Programa de Estimulo a la Innovación, CONACYT Basic Science Research Project No. 178323, DGEST (Mexico) Research Projects No.5149.13-P and TIJ-ING-2012-110, and IRSES project ACoBSEC from the European Commission. Additional funding provided by projects P08-TIC-03903 (Andalusian Regional Government), TIN2011-28627-C04-02 (Spanish Ministry of Science and Innovation), project 83 (CANUBE) awarded by the CEI-BioTIC UGR. Regional Government Junta de Extremadura, Consejería de Economía, Comercio e Innovación and FEDER, project GRU10029.

References

1. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. John Wiley & Sons (2005)
2. Cantú-Paz, E.: Parameter setting in parallel genetic algorithms. In: Lobo, F.G., Lima, C.F., Michalewicz, Z. (eds.) Parameter Setting in Evolutionary Algorithms. SCI, vol. 54, pp. 259–276. Springer, Heidelberg (2007)
3. De Jong, K.A., Potter, M.A., Spears, W.M.: Using problem generators to explore the effects of epistasis. In: Bäck, T. (ed.) ICGA, pp. 338–345. Morgan Kaufmann (1997)
4. De Jong, K.A., Spears, W.M.: An analysis of the interacting roles of population size and crossover in genetic algorithms. In: Schwefel, H.-P., Männer, R. (eds.) PPSN 1990. LNCS, vol. 496, pp. 38–47. Springer, Heidelberg (1991)

5. Di Martino, S., Ferrucci, F., Maggio, V., Sarro, F.: Towards migrating genetic algorithms for test data generation to the cloud. In: Software Testing in the Cloud: Perspectives on an Emerging Discipline, pp. 113–135. IGI Global (2013)
6. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer (2003)
7. Fazenda, P., McDermott, J., O'Reilly, U.-M.: A library to run evolutionary algorithms in the cloud using mapReduce. In: Di Chio, C., et al. (eds.) EvoApplications 2012. LNCS, vol. 7248, pp. 416–425. Springer, Heidelberg (2012)
8. Fernández De Vega, F., Olague, G., Trujillo, L., Lombraña González, D.: Customizable execution environments for evolutionary computation using boinc + virtualization. Natural Computing 12(2), 163–177 (2013)
9. Garcia-Valdez, M., Mancilla, A., Trujillo, L., Merelo, J.-J., Fernandez-de Vega, F.: Is there a free lunch for cloud-based evolutionary algorithms? In: 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 1255–1262 (2013)
10. García-Valdez, M., Trujillo, L., de Vega, F.F., Merelo Guervós, J.J., Olague, G.: Evospace-interactive: A framework to develop distributed collaborative-interactive evolutionary algorithms for artistic design. In: Machado, P., McDermott, J., Carballal, A. (eds.) EvoMUSART 2013. LNCS, vol. 7834, pp. 121–132. Springer, Heidelberg (2013)
11. García-Valdez, M., Trujillo, L., Fernández de Vega, F., Merelo Guervós, J.J., Olague, G.: EvoSpace: A Distributed Evolutionary Platform Based on the Tuple Space Model. In: Esparcia-Alcázar, A.I. (ed.) EvoApplications 2013. LNCS, vol. 7835, pp. 499–508. Springer, Heidelberg (2013)
12. Gong, Y., Fukunaga, A.: Distributed island-model genetic algorithms using heterogeneous parameter settings. In: IEEE Congress on Evolutionary Computation, pp. 820–827. IEEE (2011)
13. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)
14. Kennedy, J., Spears, W.: Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In: The 1998 IEEE International Conference on Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence 1998, pp. 78–83 (May 1998)
15. Kramer, O.: Self-Adaptive Heuristics for Evolutionary Computation. SCI, vol. 147. Springer, Heidelberg (2008)
16. Lobo, F.G., Lima, C.F., Michalewicz, Z.: Parameter Setting in Evolutionary Algorithms. Springer Publishing Company, Incorporated (2007)
17. Merelo-Guervos, J., Castillo, P., Laredo, J.L.J., Mora Garcia, A., Prieto, A.: Asynchronous distributed genetic algorithms with Javascript and JSON. In: 2008 IEEE Congress on Evolutionary Computation (CEC), pp. 1372–1379 (June 2008)
18. Sherry, D., Veeramachaneni, K., McDermott, J., O'Reilly, U.-M.: Flex-GP: Genetic programming on the cloud. In: Di Chio, C., et al. (eds.) EvoApplications 2012. LNCS, vol. 7248, pp. 477–486. Springer, Heidelberg (2012)
19. Smaoui Feki, M., Nguyen, H.V., Garbey, M.: Parallel genetic algorithm implementation for boinc. In: PARCO, pp. 212–219 (2009)
20. Tanabe, R., Fukunaga, A.: Evaluation of a randomized parameter setting strategy for island-model evolutionary algorithms. In: IEEE Congress on Evolutionary Computation, pp. 1263–1270. IEEE (2013)
21. Trujillo, L., Valdez, M.G., de Vega, F.F., Guervós, J.J.M.: Fireworks: Evolutionary art project based on evospace-interactive. In: IEEE Congress on Evolutionary Computation, pp. 2871–2878. IEEE (2013)