

Cloud-based Evolutionary Parallel Computation using low cost storage services

M.G. Arenas, J.J. Merelo, P. Castillo,
G. Romero, J.L.J. Laredo, A. Mora
School of Computer Science and Telecommunications
University of Granada
Granada, 18071 - Spain
Email: maribel@atc.ugr.es

Abstract—In this paper we will check several different, and difficult, problems, and present what effects the automatic load-balancing and asynchrony have on the speed of resolution of problems. We conclude that the effects of parallelization are noted with difficult problems, although there is a complicated interplay between the number of nodes and migration rate that makes difficult to say exactly what is the best combination of them. This paper uses Dropbox as free storage service to connect several heterogeneous computers to run an evolutionary algorithm. The algorithmic performance of the system has been proved in previous papers.

I. INTRODUCTION

The main objective of this research is to find easily available means to either use or connect computing nodes in a distributed evolutionary computation experiment, and this often means resorting to free and readily available services. DropboxTM is one of these services: it is commercialized as a *cloud storage* service, which is free up to a certain level of use (measured in traffic and usage). There are many other services like this one; however Dropbox was chosen due to its popularity, which also implies having many more potential volunteer users of a massive evolutionary computation experiment. There are also other features that make it the right tool for these experiments. Some other cloud storage services, like Wuala, provide a client program on which one must add explicitly the files that will be stored, which does not allow a seamless integration with the filesystem; others, like ZumoDrive, use remotely-mounted filesystems whose access is not so fast. Dropbox monitors local filesystems, and uploads them asynchronously, which makes it faster from the local point of view¹.

In the experiments we are performing, we are interested in its use as a file *synchronization* service. When one file in one of the folders that is monitored by Dropbox is changed, it is uploaded to Dropbox servers and then distributed to all the clients that share the same folder. It is interesting, however, to note that from the programming point of view, all folders are written and read as local one, which makes its use quite easy, and also seamless.

¹The characteristics of these and others online backup services can be seen in http://en.wikipedia.org/wiki/Comparison_of_online_backup_services

In previous experiments [1] we measured whether adding several computers to an experiment of this kind resulted in an increase in the number of simultaneous evaluations. In this paper we will measure whether besides an increase in speed, the algorithm profits from the distribution and asynchrony of the particular instance we have implemented, or on the contrary it suffers from it. In order to do that, we chose two optimization problems with a different degree of difficulty, and measured the time needed to find the solution, along with the number of evaluations.

The rest of the paper is organized as follows: after a brief section presenting the state of the art in voluntary and pool-based evolutionary computation, we describe the algorithm, the experimental setup and the implementation in Section III; results of these experiments will be briefly presented in Section IV, to be followed by the conclusion, discussion and future lines of work in Section V.

II. STATE OF THE ART

Cloud computing [2], [3] is an emergent technology, and as such research related to it is just recently emerging. Research addressing cloud storage is mainly related to content delivery [4] or designing data redundancy schemes to ensure information integrity [5]. However, its use in distributed computing has not been addressed in such depth. Even if it is related to data grids [6], in this paper we address the use of free cloud storage as a medium for doing distributed evolutionary computation, in a more or less parasitic way [7], since we use the infrastructure laid by the provider as part of an immigration scheme in an island-based evolutionary algorithm [8].

Thus we will have to look at pool-based distributed evolutionary algorithms for the closest methods to the one presented here. In these methods, several nodes or *islands* share a *pool* where the common information is written and read. To work against a single pool of solutions is an idea that has been considered almost from the beginning of research in distributed evolutionary algorithms. Asynchronous Teams or A-Teams [9]–[11] were proposed in the early nineties as a cooperative scheme for autonomous agents. The basic idea is to create a work-flow on a set of solutions and apply several heuristic techniques to improve them, possibly including humans working on them. This technique is not

constrained to evolutionary algorithms, since it can be applied to any population based technique, but in the context of EAs, it would mean creating different single-generation algorithms, with possibly several techniques, that would create a new generation from the existing pool.

The A-Team method does not rely on a single implementation, focusing on the algorithmic and data-flow aspects, in the same way as the Meandre [12] system, which creates a data flow framework, with its own language (called ZigZag), which can be applied, in particular, to evolutionary algorithms.

While algorithm design is extremely important, implementation issues always matter, and some recent papers have concentrated on dealing with pool architectures in a single environment: G. Roy et al. [13] propose a shared memory multi-threaded architecture, in which several threads work independently on a single shared memory, having read access to the whole pool, but write access to just a part of it. That way, interlock problems can be avoided, and, taking advantage of the multiple thread-optimized architecture of today's processors, they can obtain very efficient, running time-wise, solutions, with the added algorithmic advantage of working on a distributed environment. Although they do not publish scaling results, they discuss the trade off of working with a pool whose size will have a bigger effect on performance than the population size on single-processor or distributed EAs. The same issues are considered by Bollini and Piastra in [14], who present a design pattern for persistent and distributed evolutionary algorithms; although their emphasis is on persistence, and not performance, they try to present several alternatives to decouple population storage from evolution itself (*traditional* evolutionary algorithms are applied directly on storage) and achieve that kind of persistence, for which they propose an object-oriented database management system accessed from a Java client. In this sense, our former take on browser-based evolutionary computation [15] is also similar, using for persistence a small database accessed through a web interface, but only for the purpose of interchanging individuals among the different nodes, not as storage for the whole population.

In fact, the efforts mentioned above have not had much continuity, probably due to the fact that there have been, until now, few (if any) publicly accessible online databases. However, given the rise of cloud computing platforms over the last few years, interest in this kind of algorithms has bounced back, with implementations using the public FluidDB platform [16] having been recently published.

III. THE ALGORITHM

A pool based evolutionary algorithm can be described as an island model [17] without topology; in fact, it is closer to the *island* metaphor since migrants are sent to the *sea* (pool), and come also from it, that is, the evolutionary algorithm is a canonical one with binary codification, except for two steps within the cycle that (conditionally) emit or receive immigrants. A minimum number of evaluations for the whole

algorithm is set from the beginning; we will see later on how to control when this minimum number of evaluations is reached.

During the evolutionary loop, new individuals are selected using 3-tournament selection and generated using bit-flip mutation and uniform crossover. Migration is introduced in the algorithm as follows: after the population has been evaluated, migration might take place if the number of generations selected to do it is reached. The best individual is sent to the pool, and the best individual in the pool (chosen among those emitted by the other nodes) is incorporated into the population; if there has been no change in the best individual since the last migration, a random individual is added to the pool, which adds diversity to the population even if the individual fitness is not the highest. For the time being this has no influence on the result, but will have it later on when algorithmic tests are run.

Migrants, if any, are incorporated into the population substituting the worst individual, along with the offspring of the previous generation using generational replacement with a 1-elite. Population was set to 1000 individuals for all problems, and the minimum number of evaluations has been four million. Several migration rates were tested to assess its impact on performance. Besides, we introduced a 1-second delay after migration so that workload was reduced and the Dropbox daemon had enough time to propagate files to the rest of the computers. This delay makes 1-computer experiments faster when less migration is made, and will probably have to be fine-tuned in the future. The results are updated at the end of the loop to check if the algorithm has finished, that is, found the (single) solution to the problem.

One of the advantages of this topology-less arrangement is the independence from the number of computers participating in the experiment, and also the lack of need from a *central* server, although it can be arranged so that one of the computers starts first, and the others start running when some file is present. Adding a new computer, then, does not imply to arrange new connections to the current set of computers; the only thing that needs to be done is to locate the directory with the migrated individuals that is shared.

Two representative functions have been chosen to perform the tests; the main idea is that they took a long enough time to make sense in a distributed environment, but at the same time a short enough time that experiments did not take a long time. One of them is **P-Peaks**, a multimodal problem generator proposed by De Jong in [18]; a **P-Peaks** instance is created by generating P random N -bit strings where the fitness value of a string \vec{x} is the number of bits that \vec{x} has in common with the nearest peak divided by N .

$$f_{P-PEAKS}(\vec{x}) = \frac{1}{N} \max_{1 \leq i \leq P} \{N - H(\vec{x}, Peak_i)\} \quad (1)$$

where $H(\vec{x}, \vec{y})$ is the Hamming distance between binary strings \vec{x} and \vec{y} . In the experiments made in this paper we will consider $P = 300$ and $N = 600$. Note that the optimum fitness is 1.0.

The second function is **MMDP** [19], which is a deceptive

problem composed of k subproblems of 6 bits each one (s_i). Depending of the number of ones (unitation) s_i takes the values depicted next:

$$\begin{aligned} fitness_{s_i}(0) &= 1.0 & fitness_{s_i}(1) &= 0.0 \\ fitness_{s_i}(2) &= 0.360384 & fitness_{s_i}(3) &= 0.640576 \\ fitness_{s_i}(4) &= 0.360384 & fitness_{s_i}(5) &= 0.0 \\ fitness_{s_i}(6) &= 1.0 \end{aligned}$$

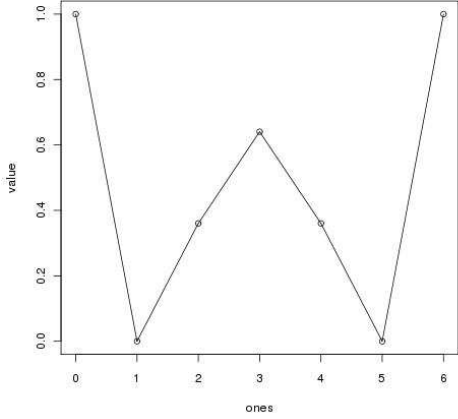


Fig. 1. Graph of the values for a single variable in the MMDP problem (bottom)

The fitness value is defined as the sum of the s_i subproblems with an optimum of k (equation 2). Figure 1 represents one of the variables of the function. The number of local optima is quite large (22^k), while there are only 2^k global solutions².

$$f_{MMDP}(\vec{s}) = \sum_{i=1}^k fitness_{s_i} \quad (2)$$

In this paper we have used $k = 80$, in order to make it difficult enough to need a parallel solution.

IV. EXPERIMENTS AND RESULTS

In this occasion the experiments were done in several different computers connected also in different ways; however, computers were added to the experiment in the same order; the problems were solved first in a single computer, then on two and finally with four computers. Total time, as well as the number of evaluations, were measured. Since the end of the experiment is propagated also via Dropbox, the number of evaluations is not exactly the one reached when the solution is found. This number also increases with the number of nodes.

The only parameter that was changed during experiments was migration rate. We were interested in doing this, since network performance will be impacted negatively with migration rate: bandwidth usage (and maybe latency) increases with the

²The local optima occur when there are 3 ones; off all the 64 possible combinations of six zeros and ones, there are 22 with exactly three ones

TABLE I
SUCCESS RATE FOR THE MMDP PROBLEM WITH DIFFERENT NUMBER OF NODES AND MIGRATION RATES.

Nodes	Generations between migration	Success rate
1 2 4	100	0.83 0.95 1
1 2 4	200	0.70 0.88 1
1 2 4	400	0.80 0.90 1

inverse of the migration rate. On the other hand, evolutionary performance will increase in the opposite direction: the bigger the migration rate, the more similar to a panmictic population will be, which might make finding the solution easier; on the other hand, it will also decrease diversity, making the relationship between migration rate and evolutionary/runtime performance quite complex and worth studying.

To keep (the rest of) the conditions uniform for one and two machines, all parameters were fixed but for the population, which was distributed among the machines in equal proportions: all computers maintained a population of 1000, so that initial diversity was roughly the same. Further experiments will have to be made keeping population constant, but this is left for further study.

Finally, Dropbox itself was used to check for termination conditions: a file was written on the folder indicating the experiment had finished; when the other node read that file, it finished too; all nodes were kept running until the solution was found or until a maximum number of generations were reached. That is why, in some cases, solution is not found; the number of generations was computed so that it was possible in a high number of cases to find out the solution.

The computers used in this experiment were laptops connected via the University of Granada WiFi, they were different models, and were running different operating systems and versions of them. The most powerful computer was the first one; then #2 was the second-best, and finally numbers 3 and 4 were the least powerful ones. Since computers run independently without synchronization checkpoints, load balancing is automatic, with more powerful computers contributing more evaluations to the mix, and less powerful ones contributing fewer.

The first thing that was checked with the two problems examined (P-Peaks and MMDP) was whether adding more computers affected the solution rate. For P-Peaks there was no difference, independently of migration rate and number of computers, all experiments found the solution. However, there was a difference for MMDP, shown in Table I.

The evolution with the migration rate can also be observed in figure 2; as was advanced in the introduction, the relationship is quite complex and decrease or increase do not lead to a monotonic change of the success rate. In fact, the best success rate corresponds to the highest migration

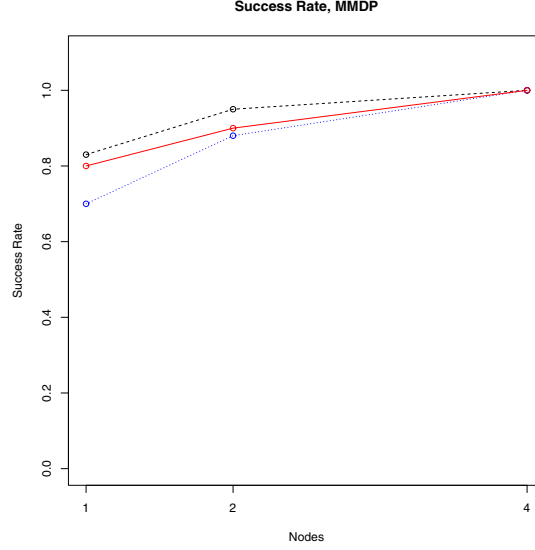


Fig. 2. Variation of the success rate in the MMDP problem with the number of nodes (x axis) and migration rate: red-solid corresponds to migration rate = 400; blue-dotted to 200, and black-dashed to 100.

rate (migration after 100 generations), but the second best corresponds to the lowest one (migration after 400), which is almost akin to no migration, since taking into account that generations run asynchronously, this might mean that in fact on migrant from other nodes is incorporated into the population. This result is in accordance with the *intermediate disturbance hypothesis*, proved by us previously [20]. However, it is not clear in this case that migration in 100 generations can be actually considered intermediate and in 200 too high, so more experiments will have to be performed to ascertain the optimum migration rate.

Thus, having proved that success rate increases with the number of nodes, we will have to study how performance varies with it. Does the algorithm really find the solution faster when more nodes are added? We have computed time only for the experiments that actually found the solution, and plotted the results in figures 3 and 4.

As seen above in the case of MMDP, there is not a straightforward relationship between the migration rate and the time to solution; in this case, the relationship between the number of computers and time solution is also complex. If we look first at the P-Peaks experiment in 3 we see that we obtain little time improvement when adding more nodes to the mix. Since success rate is already 100% with a single computer, and the solution takes around two minutes, the delay imposed by Dropbox implies that it is not very likely that the migrated solutions are transmitted to the other nodes. In this case it is the intermediate migration rate (every 40 generations) the only one that obtains a steady decrease of time to solution. The best time is obtained for a single node and a migration gap of 60; in general, the best times are for the highest migration gap since the total delay induced by migration is also the least. These

results probably imply that there must be a certain degree of complexity in the problems to take advantage of the features in this environment. For relatively *easy* problems, which need few generations, there is nothing to gain.

The situation varies substantially for the MMDP, as seen in figure 4. In this case, the best result is obtained for four nodes and the smallest mutation gap (every 100 generations, dashed black line). However, it is interesting to observe that trend change for two nodes in all cases, either the solution takes more time than for a single node or it takes less than for four nodes; the conclusion is, anyways, that the increased number of simultaneous evaluations brought by the number of nodes eventually makes solution faster. However, a fine-tuning of the migration gap is needed in order to take full advantage of the parallel evaluation in the Dropbox-based system.

V. CONCLUSIONS AND FUTURE WORK

In general, and for complex problems like the MMDP, a Dropbox-based system can be configured to take advantage of the parallelization of the evolutionary algorithm and obtain reliably (in a 100% of the cases) solutions in less time than a single computer would. Besides, it has been proved that it does not matter whether the new computers added to the set are more or less powerful than the first one. In general, however, adding more computers to a set synchronized via Dropbox has more influence in the success rate than in the time needed to find the solution, which seems roughly linked to the population size, although this hypothesis will have to be tested experimentally. On the other hand, using relatively simple problems like P-Peaks yields no sensible improvement, due to the delay in migration imposed by Dropbox, which implies that this kind of technique would be better left only

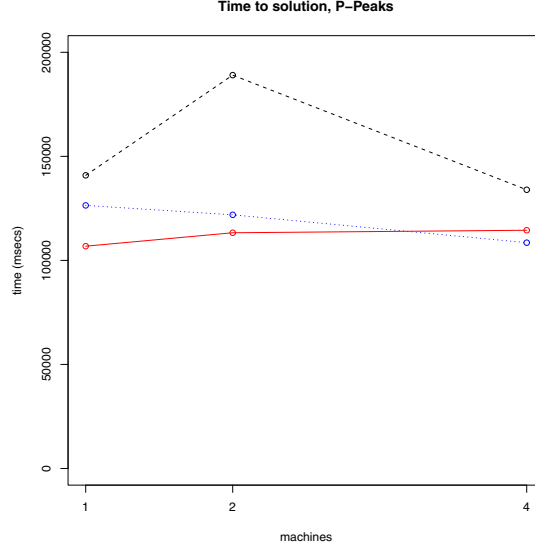


Fig. 3. Variation of the time needed to find the solution in the P-Peaks problem with the number of nodes (x axis) and migration rate: red-solid corresponds to migration rate = 60; blue-dotted to 40, and black-dashed to 20.

for problems that are at the same time difficult from the evolutionary point of view and also slow to evaluate.

However, several issues remain to be studied. First, more accurate performance measures must be taken to measure how the time needed to find the solution in all occasions scales when new machines are added. We will have to investigate how parameter settings such as population size and migration gap (time passed between two migrations) influence these measures. This paper proves that this influence is important, but it is not clear what is the influence on the final result. It would be also interesting to test different migration policies affect final result, as done in [21], where it was found out that migrating the best one might not be the best policy.

An important issue too is how to interact with Dropbox so that information is distributed optimally and with a minimal latency. In this case we had to stop each node for a certain time (which was heuristically found to be 1 second) to leave time for the Dropbox daemon to distribute files. In an experiment that lasts for less than two minutes, this can take up 25% of the total time (per node), resulting in an obvious drag in performance that can take many additional nodes to compensate. A deeper examination of the Dropbox API and a fine-tuning of these parameters will be done in order to fix that.

Finally, this framework opens many new possibilities for distributed evolutionary computation: meta-evolutionary computation, artificial life simulations, and big-scale simulation using hundreds or even thousands of clients. The type of problems suitable for this, as well as the design and implementation issues, will have to be explored. Other cloud storage solutions, preferably including open source implementations, will be also tested. Since they have different models (synchronization

daemon or user-mounted filesystems, mainly) latency and other features will be completely different, so we expect that performance will be affected by this.

ACKNOWLEDGEMENTS

This work has been supported in part by the CEI BioTIC GENIL (CEB09-0010) MICINN CEI Program (PYR-2010-13 and PYR-2010-29) project, the TIN2011-28627-C04-02 project, the UGR 2349-2011 project and the Andalusian Regional Government P08-TIC-03903 and P08-TIC-03928 projects. The authors are very grateful to the anonymous referees whose comments and suggestions have contributed to improve this paper.

REFERENCES

- [1] M. Garca-Arenas, J.-J. Merelo, A. M. Mora, P. Castillo, G. Romero, and J. Laredo, "Assessing speed-ups in commodity cloud storage services for distributed evolutionary algorithms," in *Proceedings CEC 2011*, 2011, to be published.
- [2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, April 2010. [Online]. Available: <http://doi.acm.org/10.1145/1721654.1721672>
- [3] H. Jin, S. Ibrahim, T. Bell, W. Gao, D. Huang, and S. Wu, "Cloud types and services," in *Handbook of Cloud Computing*, B. Furht and A. Escalante, Eds. Springer US, 2010, pp. 335–355. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-6524-0_14
- [4] J. Broberg, R. Buyya, and Z. Tari, "Metacdn: Harnessing [']storage clouds' for high performance content delivery," *Journal of Network and Computer Applications*, vol. 32, no. 5, pp. 1012 – 1022, 2009, next Generation Content Networks. [Online]. Available: <http://www.sciencedirect.com/science/article/B6WKB-4W0R0PY-2/2/3b86507c3e3c2d6d41f58cd27f6914eb>
- [5] L. Pamies-Juarez, P. Garca-Lpez, M. Snchez-Artigas, and B. Herrera, "Towards the design of optimal data redundancy schemes for heterogeneous cloud storage infrastructures," *Computer Networks*, vol. In Press, Corrected Proof, pp. –, 2010. [Online]. Available: <http://www.sciencedirect.com/science/article/B6VRG-51H70BC-1/2/7d311ced40a874e4cf540a373eeefc430>

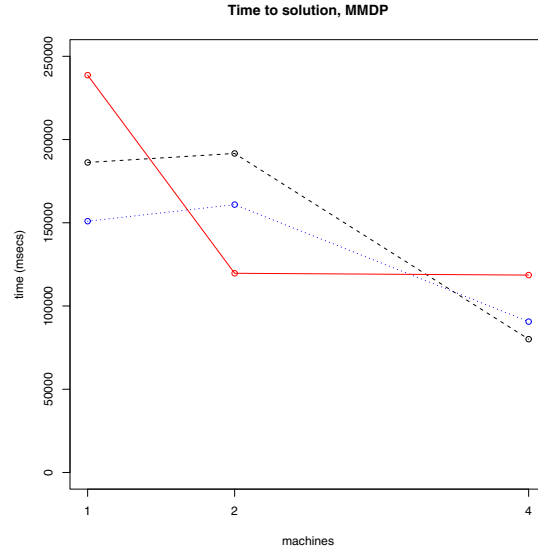


Fig. 4. Variation of the time needed to find the solution in MMDP with the number of nodes (x axis) and migration rate: red-solid corresponds to migration rate = 400; blue-dotted to 200, and black-dashed to 100.

- [6] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 187–200, 2000.
- [7] A.-L. Barabási, V. W. Freeh, H. Jeong, and J. B. Brockman, "Parasitic computing," *Nature*, vol. 412, no. 6850, pp. 894–897, August 2001. [Online]. Available: http://www.nature.com/cgi-taf/DynaPage.taf?file=/nature/journal/v412/n6850/abs/412894a0_fs.html
- [8] E. Cant-Paz, "Topologies, migration rates, and multi-population parallel genetic algorithms," in *Genetic and Evolutionary Computation Conference, GECCO-99*, 1999, pp. 13–17.
- [9] P. de Souza and S. Talukdar, "Genetic algorithms in asynchronous teams," in *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 1991, pp. 392–399.
- [10] S. Talukdar, L. Baerentzen, A. Gove, and P. De Souza, "Asynchronous teams: Cooperation schemes for autonomous agents," *Journal of Heuristics*, vol. 4, no. 4, pp. 295–321, 1998.
- [11] S. Talukdar, S. Murthy, and R. Akkiraju, "Asynchronous teams," *INTERNATIONAL SERIES IN OPERATIONS RESEARCH AND MANAGEMENT SCIENCE*, pp. 537–556, 2003.
- [12] X. Llorà, B. Ács, L. Auvil, B. Capitanu, M. Welge, and D. Goldberg, "Meandre: Semantic-driven data-intensive flows in the clouds," Illinois Genetic Algorithms Laboratory, Tech. Rep. 2008103, 2008.
- [13] G. Roy, H. Lee, J. Welch, Y. Zhao, V. Pandey, and D. Thurston, "A distributed pool architecture for genetic algorithms," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, May 2009, pp. 1177–1184.
- [14] A. Bollini and M. Piastra, "Distributed and persistent evolutionary algorithms: a design pattern," in *Genetic Programming, Proceedings EuroGP '99*, ser. Lecture notes in computer science, no. 1598. Springer, 1999, pp. 173–183.
- [15] J. Merelo, P. Castillo, J. Laredo, A. Mora, and A. Prieto, "Asynchronous distributed genetic algorithms with Javascript and JSON," in *WCCI 2008 Proceedings*. IEEE Press, 2008, pp. 1372–1379. [Online]. Available: http://atc.ugr.es/I+D+i/congresos/2008/CEC_2008_1372.pdf
- [16] J. Merelo, "Fluid evolutionary algorithms," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, pp. 1–8.
- [17] D. Whitley, S. Rana, and R. Heckendorn, "Island model genetic algorithms and linearly separable problems," *Evolutionary Computing*, pp. 109–125, 1997.
- [18] K. A. D. Jong, M. A. Potter, and W. M. Spears, "Using problem generators to explore the effects of epistasis," in *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, T. Bäck, Ed. San Francisco, CA: Morgan Kaufmann, 1997. [Online]. Available: citeseer.ist.psu.edu/dejong97using.html
- [19] D. E. Goldberg, K. Deb, and J. Horn, "Massive multimodality, deception, and genetic algorithms," in *Parallel Problem Solving from Nature*, 2, R. Männer and B. Manderick, Eds. Amsterdam: Elsevier Science Publishers, B. V., 1992, pp. 37–48. [Online]. Available: citeseer.ist.psu.edu/goldberg92massive.html
- [20] J. J. Merelo, A. M. Mora, P. A. Castillo, J. L. J. Laredo, L. Araujo, K. C. Sharman, A. I. Esparcia-Alcázar, E. Alfaro-Cid, and C. Cotta, "Testing the intermediate disturbance hypothesis: Effect of asynchronous population incorporation on multi-deme evolutionary algorithms," in *Parallel Problem Solving from Nature - PPSN X*, ser. LNCS, G. Rudolph, T. Jansen, S. Lucas, C. Poloni, and N. Beume, Eds., vol. 5199. Dortmund: Springer, 13-17 Sep. 2008, pp. 266–275.
- [21] J. Araujo, L.; Merelo, "Diversity through multiculturalism: Assessing migrant choice policies in an island model," *IEEE Transactions on Evolutionary Computation*, 2010. [Online]. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-78650474606&partnerID=40&md5=7615134b1c2b04077050397cf4ea2ad7>