HPCLatAm 2013, pp. 25-36 (full paper)
Session: Evolutionary Computation & Scheduling
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

Two Models for Parallel Differential Evolution*

María Laura Tardivo[1,2,3], Paola Caymes-Scutari[2,3], Miguel Méndez-Garabetti[2,3] and Germán Bianchini[2]

[1] Departamento de Computación, Universidad Nacional de Río Cuarto.
(X5804BYA) Río Cuarto, Córdoba, Argentina
[2] Laboratorio de Investigación en Cómputo Paralelo/Distribuido (LICPaD)
Departamento de Ingeniería en Sistemas de Información, Facultad Regional Mendoza
Universidad Tecnológica Nacional. (M5502AJE) Mendoza, Argentina
[3] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

Abstract. In the area of scientific research there are countless optimization problems that can not be exactly solved by a computer in a reasonable time. Advances in computing science have addressed with these problems developing different techniques that attempt to approximate the exact solutions. Among them, the Differential Evolution (DE) algorithm is a method of common choice. Numerous applications have demonstrated the potential of the method in problem solving, naming efficiency, convergence and robustness. Moreover, by the nature of the algorithm, there are several approaches for transforming its sequential processing scheme into a parallel one, so as to increase the computational speed without neglecting the solutions quality.
This paper presents two parallel alternatives to the classical Differential Evolution algorithm. Both proposals are based on an island model, a ring interconnection topology and a population migration strategy, whose advantages and drawbacks are presented. They have been proved with a set of benchmark functions considering different configurations for the parameters of DE, and they have also been analyzed according to explicit performance measurements.

1 Introduction

One of the research areas in computer science is the study of strategies for solving certain problems that can not be solved by deterministic methods in a reasonable computation time. Within this group we find combinatorial optimization problems, present in different scientific and industrial areas.

An optimization problem is defined in [11] by a couple (S, f), where S represents the set of possible solutions and $f : S \to \mathbb{R}$ is the objetive function to optimize. The objective function assigns to every solution $s \in S$ of the search space a real number indicating its worth. Hence, the main goal in solving an optimization problem is to find a solution $s* \in S$, called *global optimum*, which

* This work has been supported by UTN under projects PICT2010/12 and UTN1585, and by ANPCyT under project PRH PICT-2008-00242.

has the best objective function value of all solutions of the search space. However, complete methods might need exponential computation time to explore the search space in the worst case. Thus, the use of approximated optimization techniques that provide good solutions to complex problems computed in a reasonable time has gained popularity.

In general, metaheuristics family is divided into two categories: single solution based metaheuristics (also known as trajectory metaheuristics, or S-metaheuristics) and population based metaheuristics (P-metaheuristics). Each metaheuristic in the first class starts with a single solution that is perturbed to explore the search space of the problem addressed. Metaheuristics in the latter class is characterized by working with a set of solutions, called population, represented by individuals who interact with each other to carry out the search process. Well-known single solution methods are Local Search, Simulated Annealing and Tabu Search. Within population metaheuristics we found the Evolutionary Algorithms like Genetic Algorithms or Differential Evolution; and some inspired from the collective behavior of species such as Ant Colonies, among others [11].

Originally, metaheuristics were developed following a sequential processing scheme. Later, and consequently to technological advances in the area of hardware and the increasing demand for computing power, several alternatives were proposed in order to maximize the use of resources, without jeopardising the quality of the solutions or even enhancing the solutions obtained by the sequential method.

In this paper we focus on Differential Evolution (DE) as a population metaheuristic applicable to provide solutions for optimization problems, and widely used over several benchmark functions and real-world problems [15]. We present two different alternatives for parallelizing DE, its main characteristics, advantages and drawbacks of using each one. Also, we describe the experiments carried out in order to test the performance of both versions, with regard to solutions quality versus computing time. Existing works related to the theme, used small test cases compared to real combinatorial optimization problems. In our tests, we considered large enough cases to deal with such problems.

The paper is organized as follows: section 2 describes this particular metaheuristic, including its main characteristics and a description of its processing model. Section 3 shows the advantages of using parallel computing when the problem is addressed by population-based metaheuristics, in order to obtain better quality results and/or to reduce the computation time. Two parallelization versions for DE are also included in this section. Section 4 describes the experiments carried out. Finally, we present the conclusions.

2 Differential Evolution

The Differential Evolution (DE) algorithm was proposed by Price, Storn and Lampinen in 1995 [10]. It is a population based optimizer or metaheuristic that starts generating a population of D-dimensional vectors whose initial values are

randomly obtained based on the limits defined by the inputs of the algorithm. The total of individuals in the population is a known constant value.

Each individual belongs to a generation g, i.e., let $X_{i,g} = (x^1_{i,g}, ..., x^D_{i,g})$ an individual of the population, with $i = 1, ...,$N where the index i denotes *i-th* population individual, g determines the generation to which the individual belongs and N is the total number of individuals in the population.

The main idea of the method is to use vector difference in order to modify the population vector. This idea has been integrated into a recombination operator of two or more solutions, consisting of two phases (mutation and crossover), with the aim of guiding the search towards "good" solutions.

In the following, we explain the three classical main operators of DE. In section 3 we also introduce the *migration* operator, which is typically used in parallel versions of metaheuristics.

Mutation: After initialization, DE mutates and recombines the current population to produce another one constituted by N individuals. The mutation process begins in each generation selecting random individuals $X_{r_1,g}, X_{r_2,g}$. The *i-th* individual is perturbed using the strategy of the formula (1), where the indexes i, r_1 and r_2 are integers numbers different from each other, randomly generated in the range [1, N].

$$\text{“}DE/best/1\text{”} : V_{i,g+1} = X_{best,g} + (X_{r_1,g} – X_{r_2,g})F \tag{1}$$

The constant F represents a scaling factor and controls the difference amplification between individuals r_1 and r_2, and it is used to avoid stagnation in the search process. $X_{best,g}$ is the best individual, i.e., it has the best value of the objective function evaluation among all individuals of current generation g. The notation "$DE/best/1$" represents that the base vector chosen is the best individual, and "1" vector difference is added to it.

The formula (1) is the most general of the mutation strategies. Additionally, the original DE version proposes five alternative strategies [10].

***Crossover*:** After the mutation phase, each perturbed individual $V_{i,g+1} = (v^1_{i,g+1}, ..., v^D_{i,g+1})$ and the individual $X_{i,g} = (x^1_{i,g}, ..., x^D_{i,g})$ are involved in the crossover operation, generating a new vector $U_{i,g+1} = (u^1_{i,g+1}, ..., u^D_{i,g+1})$, denominated "trial vector", and obtained using the expression (2).

$$U^j_{i,g+1} = \begin{cases} v^j_{i,g+1} & \text{if } rand_j \leq Cr \text{ or } j = k \\ x^j_{i,g} & \text{in other case} \end{cases} \tag{2}$$

where $j = 1, ..., D$, and $k \in \{1, ..., D\}$. The constant $Cr \in [0, 1]$, denominated *crossover factor*, is a parameter of the algorithm defined by the user. Cr is used to control the values fraction that are copied from the mutant vector V. $rand_j$ is the output of a uniformly distributed random number generator, and is generated in each comparison made on the vector components. The value k is a randomly generated index chosen for each individual. The vector component for that index is taken from the mutated vector to ensure that the trial vector is not exactly equal to its source vector $X_{i,g}$.

There are two crossing operators that can be applied: binomial or exponential. Both types use the expression (2), but differ in the way it is applied. The binomial crossover operator copies the jth parameter value from the mutant vector $V_{i,g+1}$ to the corresponding element in the trial vector $U_{i,g+1}$ if $rand_j \leq Cr$ or $j = k$. Otherwise, it is copied from the corresponding target (or parent) vector $X_{i,g}$. Instead, the exponential crossover operator inherits the parameters of trial vector $U_{i,g+1}$ from the corresponding mutant vector $V_{i,g}$ starting from a randomly chosen parameter index. Then, it continues copying the parameter values form the mutant vector $V_{i,g}$ till the jth parameter value satisfying $rand_j > Cr$. The remaining parameters of the trial vector $U_{i,g+1}$ are copied from the corresponding target vector $X_{i,g}$. To complete the notation, when the crossover applied is binomial the method is named "$DE/best/1/bin$". If a exponential crossover is used, it is referred as "$DE/best/1/exp$".

***Selection*:** This phase determines which element will be part of the next generation. The objective function of each trial vector $U_{i,g+1}$ is evaluated and compared with the objective function value for its counterpart $X_{i,g}$ in the current population. If the trial vector has less or equal objective function target value (for minimization problems) it will replace the vector $X_{i,g}$ in the next generation population. The scheme followed is presented in the expression (3).

$$X_{i,g+1} = \begin{cases} U_{i,g+1} & \text{if } f(U_{i,g+1}) \leq f(X_{i,g}) \\ X_{i,g} & \text{in other case} \end{cases} \quad (3)$$

The three stages mentioned above are repeated from generation to generation until the specified termination criterion is satisfied. This criterion could be finding a predefined minimal error or reaching a certain number of iterations.

Due to the potentialities of DE for solving optimization problems, in recent years, numerous variations and methods have been proposed with the aim of improving the performance of the classic technique. Among them are those trying to adapt DE parameters such as self-adjusting [16], [3]; others using different mechanisms to optimize the individuals selection for the mutation and selection phases [4], and some combining both methods [17]. In several studies DE have been used in combination with other metaheuristics, obtaining hybrid approaches [2], [9], [12], in order to improve the solutions quality, or to accelerate the speed of convergence. In addition, there are several studies that incorporate parallelism in order to improve the quality of the solutions obtained and/or diminish the execution time of this metaheuristic. The following section describes some of them and presents two parallel models used in this study.

3 Parallel Differential Evolution proposals

By their nature, most metaheuristics are prone to parallelism, since most variation operations can be undertaken in parallel [1]. But beyond the natural possibilities for parallelization, the difficulty lies in opting for parallel versions that, empowering the spirit of the metaheuristic, does not radically change the original method, nor be penalized by the portions of code that are not parallelizable.

There are different approaches to parallelize population-based metaheuristics, and consequently DE, depending on the purpose to be achieved. On the one hand, it is desirable to improve the fitness value of the solution found with the sequential version. On the other hand, the execution time may be reduced, trying not to affect the quality of the solutions. The ideal case would be achieving both goals at the same time.

Even though there are several classifications of parallel metaheuristics models [11, 10], in general they constitute different sorts of combinations for the *Master/Worker* model [6] (with focus on the decomposition, mapping and assignment between tasks, data and processors) and a *migration* operator. Usually, each worker is located into a separate computing node, and depending on the model, some authors named the worker units as *islands*. The ***migration*** constitutes a very important operator when there are various populations or when the population is subdivided between different processing nodes. This operator allows for the information interchange among worker processes. Migration is applied at a certain rate, and involves each local population in the selection of a set of individuals to emigrate, and a set of individuals to be replaced. The main objective of moving individuals along the communication topology, is to periodically introduce new members (information) in each population. Under the assumption that each worker is evolving to approximate the solution, the exchanged individuals may spread good characteristics locally found, opening new searching spaces. However, the migration rate can have a positive or negative influence on the quality of the solutions obtained, (it affects the heterogeneity of the population), or on the computing time (due to the communication overhead).

In recent years there have been developed different algorithmic versions to parallelize DE. They are framed within shared or distributed memory models. Among the first, we encounter those who follows a threaded implementation [8] in order to reduce the computing time, but obtaining similar quality results. Among the latter, most designs follows the standard model with migration [10], in which a master process creates a population, subdivides it and assigns the subpopulations to the worker processes, who are in charge of applying the evolutionary process at local subpopulation level and also applying migration at a certain rate. In [18] is presented a proposal for solving the Pareto front problem. An individual in the population can be migrated with a certain probability to a random position in a random subpopulation. In [13], the standard model uses a ring interconnection topology and random migration rate controlled by a parameter of the algorithm. The functions used for the tests are similar to those used in our experiments, but with much more limited dimensions and less generations than the ones used here. The aim of that work is to study the implications of a controlled migration constant. In [5], a parallel DE version is proposed and applied to solve biological systems. It also follows a ring interconnection topology, where the replacement strategy in the migration phase is to substitute the oldest member in the target subpopulation (one who has been longer in that population without being replaced) by the best individual in the source subpopulation. The population size is smaller than those used in this work, and there are employed

only four processing nodes. The analysis was done with different migration rates and they conclude by identifying the best of them.

The last three approaches mentioned are based on the standard model (similar to the model presented in section 3.1). A critical issue of it, is the consideration of the population size. The ability of the algorithm to find a solution depends on the tasks size and is related to the amount of individuals per node, making significant local and global evolution. Moreover, in occasions combinatorial optimization problems have an important number of variables, and in consequence the search space is too big. Then, it is relevant to develop techniques that encompass these scenarios. For all these reasons arises the need to perform a comparative study to test with large enough cases, and to present different alternatives that take into account scalability measures. Next subsections describe two parallel versions for DE. One follows what we call the *subpopulation-based model*, with ring interconnection topology and a semi-elitist replacement. The other, consist of an *island model*, having the entire population in each island, and a separate master process coordinating the system.

3.1 First algorithm: Subpopulation-based Model

The first model described is an "Standard Model with migration" version [10]. It follows a master-worker scheme. The figure 1 describes this model. The master process initializes and divides the population (Pop. in the figure 1) into as many subpopulations (sp 1,..., sp n) as workers in the system, and sends one subpopulation per worker. Each worker receives its subpopulation, and starts with the evolutionary actions like a classic DE. The boxes in figure 1 named "SDE" represent worker processes, running a Subpopulation-based DE. The boxes with doted lines represent the processing node in which the worker is located.

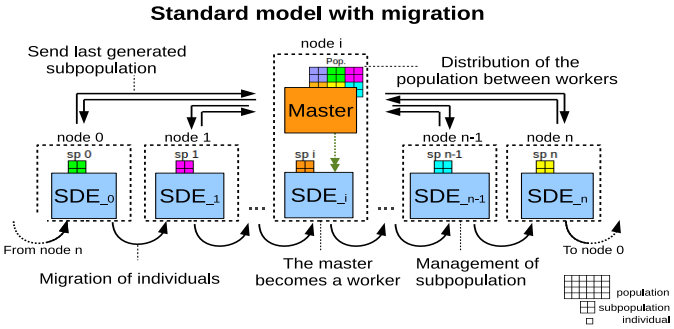

Fig. 1: **Subpopulation-based model:** Population distribution between workers.

Every certain number of generations, and considering a certain topology, begins a *migration phase*. The amount of individuals that migrate is a certain percentage of the whole population, and this value is calculated from the total of individuals involved in the process, i.e. from the initial population size; the resulting number is proportionally divided among the workers. In each node, the set of

emigrant individuals is constituted by the best individual in the subpopulation (i.e. the one who has the best fitness value) and the rest is completed by means of a random selection. While selected individuals are migrating, the worker organizes its subpopulation to determine which will be the candidates to be replaced by the newly received ones. The replacement would be completely elitist, if the worst individuals were substituted by the bests of the source subpopulation. In our proposal, the set of emigrants is composed by some individuals randomly selected and the best member, thus balancing the diversity of the population.

With this scheme, the master process remains idle while workers perform the evolutionary stages, and will have no further intervention until the merge of the results. To avoid idle computing time, in our proposal the master process becomes another worker, and it applies DE with its corresponding portion of the initial population, like any of the other workers. As can be seen in figure 1, the master process shares the same processing node with a worker process, so as to maximize the use of the resources.

Once the workers finish their generational evolutionary process, they communicate with the master in order to send the individuals of the last obtained generation. The master evaluates them, and stores the best individual found from all the subpopulations, then the process ends. In our proposal, the finalization condition consists in reaching a certain number of generations.

3.2 Second algorithm: Island Model

The second model described is framed within the classification "Algorithmic Level with cooperation" [11]. The figure 2 represents this model, whose processing scheme is described in the following. Multiple instances of DE are executed in parallel on different compute nodes, each one considering a different population of individuals (Pop. 1, ..., Pop. n), and a different random initial seed. We call each computing node "an island". A master process is in charge of monitoring the system as a whole, and each worker process is dedicated to compute all the generations in that island. As can be seen, the master process is located in an exclusive computing node, so as to coordinate the system and to avoid delaying the response to the workers.

Every certain migration rate, the worker communicates with a neighbor to send some individuals. The amount of individuals that migrate, in the same manner as was detailed before, is a certain percentage of the population. This percentage is a global value, calculated from the number of individuals in the island. Like the subpopulation model, the individuals to be migrated are the best member of the island plus other individuals randomly selected, and the received individuals will replace the worst individuals of the target population.

After a migration phase and replacement process, the workers inform to the master which is the best individual found so far. The master receives this information and temporarily stores the best individual of all those who have been sent by the workers. Then, if the termination condition is met, the master sends a message to workers indicating the end of the process. Otherwise, the master informs to continue with their evolutionary process. In our proposal, the

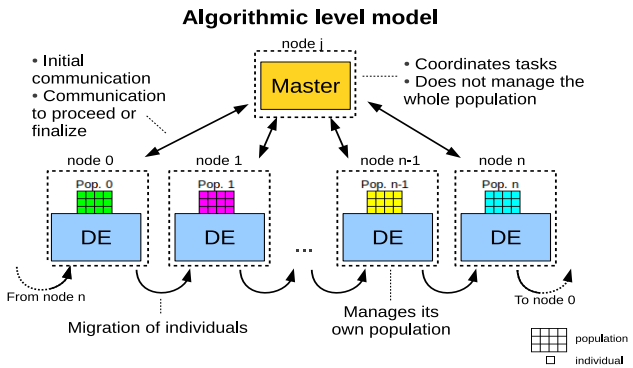

Fig. 2: **Island Model:** independent or cooperating self-contained metaheuristics.

termination condition was established, like in the previous mentioned model, to reach a certain number of generations.

This model has greater computational cost than the subpopulation based model, since the number of individuals of each island does not vary as the amount of workers increases. However, the island model can achieve an increment in the variability of the population when the number of individuals in each island is a considerable number. This can significantly promote the exploration of a larger search space, which may lead to better solutions quality.

4 Test cases and results analysis

The performance for the algorithms was tested with a set of scalable functions, obtained from [14]. For each of them, 30 executions were carried out with different seeds. The sizes of the problems considered were 100, 500 and 1000 with a population made up with 100 and 400 individuals. The function used for the test where Shifted Sphere (unimodal, search range in [-100,100] and a bias value of -450), and Shifted Rosenbrock (multimodal, search range in [-100,100] and a bias value of 390). The method used for DE is $DE/best/1/bin$. Some tests were performed with different values for F (scaling factor) and Cr (crossover factor), using functions with similar features to those used in this work. This lead to the determination of $F = 0.5$ and $Cr = 0.3$ as the values for that constant factors, this values where the same for all islands.

In both models, the exchange rate is a parameter of the algorithm. Several experiments were performed so as to establish the appropriate value for this parameter. As a consequence, in all the tests the individuals were exchanged among the islands or workers at a migration rate of 15% every 500 iterations. The value considered for the maximum number of generation was 6000 iterations.

The intercommunication topology used in both models is a ring topology, so that each worker or island receives individuals from its predecessor in the topological order, and sends their own individuals to its successor in that order.

The average error is defined as the difference between the current value of the global optimum and the value obtained by the algorithm. If the error is zero indicates that it has been found the global optimum. For the problems considered, the best results are those that are closer to zero error.

The graphs of the figures 3, 4 and 5 show the mean errors obtained in the different experiments performed with the two proposals. On the one hand, the results were computed for the model of subpopulations with a total of 400 individuals. This size was considered such that, after the distribution of the population, workers have a significant number of individuals to do their job. On the other hand, the results for the island model with 100 and 400 individuals were obtained. The comparative analysis of these last two tests can give us a pattern of the influence of population size with respect to solution quality and runtime. As hypothesis, we know that the larger the population for the island model, the greater the computing time. However, this diversity can lead to better quality solutions.

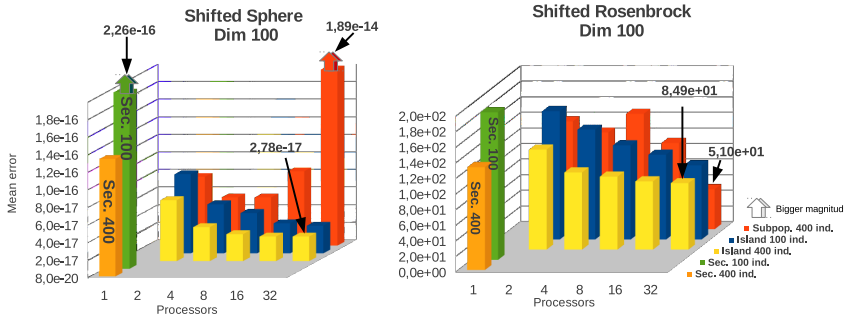

Fig. 3: Mean error of Shifted Sphere and Shifted Rosenbrock functions. Dimension 100.

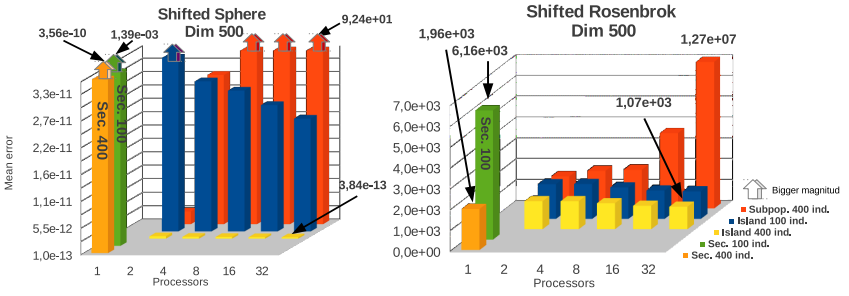

Fig. 4: Mean error of Shifted Sphere and Shifted Rosenbrock functions. Dimension 500.

In the graphs, each color represents one of the experiments mentioned above. In order to contrast with the parallel experiments, the graphs also include two columns that represents the mean error for the sequential version, with population sizes of 100 and 400 individuals. Some bar columns of the graphics have a colored arrow at top, representing that the column bar has a bigger magnitude

that the maximum scale in the graphic. Moreover, we include some small labeled black arrows with two possible purposes: they may explicitly indicate the value of those big columns or they may highlight some interesting value.

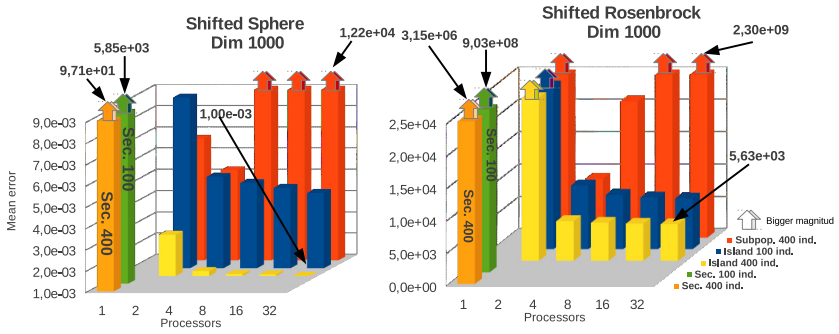

Fig. 5: Mean error of Shifted Sphere and Shifted Rosenbrock functions. Dimension 1000.

In order to test scalability, all experiments included 2, 4, 8, 16 and 32 processors dedicated to the workers processes, and a separate processor for the master process for the island model. All tests were made on a cluster with 36 CPUs distributed between 9 nodes. They have 64 bits with Intel Q9550 Quad Core 2.83GHz processors and RAM memory of 4GB DDR3 1333Mz. All the nodes are connected together by Ethernet segments and switch Linksys SLM2048 of 1Gb. Base software on the cluster includes a 64 bits Debian 5 Lenny Operating System. In the codification we use the MPICH library [7] for message passing communication between participating nodes.

Table 1 shows the average computing time, discriminating the tests according to the dimension and model analyzed. The results show that the island model with 400 individuals gives better results in terms of quality of the solution, in most cases. In those instances in which the results are better using the subpopulation based model, the differences with the island version are not significant. Moreover, in almost all test made with the subpopulation based model, the quality of the results gets worse as the number of workers increase. Although with this model the scalability in terms of computation time is achieved, the quality of the solutions is not optimal, considering the results obtained with the other model. But in certain cases, it may be desired the reduction of computational time, relegating a certain percentage of the solutions quality. For example (see table 1), the computation time for Shifted Rosenbrock function, dimension 1000 with island model, eight workers and 400 individuals is 186.64 seconds, and the error is in the order of E+03. With the subpopulation based model, having the same quantity of workers and individuals, the computing time for the function was 18.22 sec. with an error of E+04. Here we see that time is reduced about nine times compared with the island parallel model, and the obtained solution may be considered of similar quality.

Then, the subpopulation based model, compared with the island model, may be useful when what is sought is to reduce the computing time, penalizing in certain orders of magnitude the quality of the solutions.

Table 1: Average computing time (in seconds), obtained with the algorithms.

Shifted Sphere						Shifted Rosenbrock					
Par. model	2	4	8	16	32	Par. model	2	4	8	16	32
Dim 100. Sequential time: 100 ind=2.08; 400 ind=10.91						**Dim 100. Sequential time:** 100 ind=2.54; 400 ind=12.82					
Subp. 400	5,32	3,06	1,69	2,81	4,01	Subp. 400	6,29	3,48	1,95	2,93	4,13
Island 100	2,68	2,71	2,83	2,91	4,27	Island 100	3,17	3,21	3,28	3,45	5,05
Island 400	10,73	10,95	11,33	11,71	16,05	Island 400	12,71	13,00	13,36	13,77	19,03
Dim 500 Sequential time: 100 ind=11.06; 400 ind=53.26						**Dim 500 Sequential time:** 100 ind=13.65; 400 ind=63.24					
Subp. 400	26,82	13,94	7,33	5,42	3,19	Subp. 400	31,09	16,08	8,69	5,62	3,61
Island 100	12,98	13,23	13,60	14,66	19,30	Island 100	15,40	15,72	16,37	17,36	23,13
Island 400	54,92	69,95	72,32	78,04	84,74	Island 400	64,39	82,11	84,42	91,83	109,12
Dim 1000 Sequential time: 100 ind=22.57; 400 ind=110.14						**Dim 1000 Sequential time:** 100 ind=27.80; 400 ind=128.64					
Subp. 400	67,72	33,47	15,39	8,13	4,80	Subp. 400	87,25	38,14	18,22	8,53	5,49
Island 100	26,03	26,85	28,18	29,37	40,10	Island 100	30,78	32,10	32,93	34,88	46,42
Island 400	113,26	161,99	156,51	169,72	181,27	Island 400	132,89	181,16	186,64	200,62	207,92

However, the subpopulation based model, gives good quality results compared with the sequential version, and at lower computing time costs. Then, if we contrast this proposal versus the sequential algorithm, having few computational resources, we achieve a significant reduction in execution time. Although the solutions are not the best that can be achieved, for certain applications they may be considered of good quality. Also, we can notice that the hypothesis of population heterogeneity associated to solutions quality was corroborated. The experiments carried out over the island model with 400 individuals gives better results than those made with 100 individuals. Then, the island model is suitable in those cases when what is desired is an improvement in the results quality.

5 Conclusions

In this paper we describe two different models to parallelize the classical Differential Evolution metaheuristic. Through the results analysis it was found that the subpopulation model reduces significantly the computing time, but the quality of the solutions is not the optimal that can be achieved. With the island model, the computing time is not reduced, because of the model characteristics, but the solutions quality is improved significantly with the increment of the number of workers involved in the process. This feature reflects the fact that the model explores a greater search space, since each island is configured with a different initial seed. The tendency indicates that better quality solutions would be obtained as the number of workers increases. As future works we plan to employ these parallelization models to replace the sequential DE scheme in a hybrid metaheuristic that combines DE with Local Search, presented on an early work [12].

References

1. Alba, E., Tomassini M.: Parallelism and Evolutionary Algorithms. In: Proc. of the IEEE Trans. on Evol. Comp., vol. 6, num. 5, pp. 443-462 (2002)
2. Ali, M., Pant, M., Nagar, A.: Two local Search Strategies for Differential Evolution. In: 5th IEEE Inter. Conf. on Bio-Inspired Computing: Theories and Appl., vol. 2, pp 1429-1435 (2010)
3. Brest, J., Zamuda, A., Bokovie, B., Maucec M., Zumer, V.: High-Dimensional Real-Parameter Optimization using Self-Adaptive Differential Evolution Algorithm with Population Size Reduction. In: Proc. of the IEEE Congr. on Evol. Comp., pp 2032-2039 (2008)
4. Martínez, C., Rodriguez, F., Lozano, M.: Role differentiation and malleable mating for differential evolution: an analysis on large-scale optimisation. In: Soft Computing, vol. 15, issue 11, pp. 2109-2126 (2011)
5. Kozlov, K., Samsonov A.: New Migration Scheme for Parallel Differential Evolution. In: Proc. Int. Conf. on Bioinf. of Genome Reg. and Structure, pp 141-144 (2006)
6. Mattson T., Sanders B., Massingill B.: Patterns for Parallel Programming. Addison-Wesley, chapter 5, pp 143-152 (2004)
7. MPICH Message Passing Interface, http://www.mpich.org/
8. Nebro A., Durillo J.: A Study of the Parallelization of the Multi-Objective Metaheuristic MOEA/D. In: Proc. of the 4th Int. Conf. on Learning and Intelligent Optimization, vol. 6073, pp. 303-317 (2010).
9. Noman, N., Iba, H.: Accelerating Differential Evolution Using an Adaptive Local Search. In: Proc. of the IEEE Trans. on Evol. Comp., vol. 12, pp 107-125 (2008)
10. Price, K., Stron R., Lampinen J.: Differential Evolution: A Practical Approach to Global Optimization. Springer. New York (2005)
11. Talbi, E.: Metaheuristics: From Design to Implementation. John Wiley & Sons, Hoboken, New Jersey (2009)
12. Tardivo, L., Cagnina, L., Leguizamón, G.: A Hybrid Metaheuristic Based on Differential Evolution and Local Search with Quadratic Interpolation. In: XVIII Congreso Argentino de Ciencias de la Computación (2012)
13. Tasoulis, D., Pavlidis, N., Plagianakos, V., Vrahatis, M.: Parallel Differential Evolution. In: Proc. of the Congr. Evol. Comp., vol. 2, pp. 2023-2029 (2004)
14. Tang, K., Yao, X., Suganthan, P. N., MacNish, C., Chen, Y. P., Chen, C. M., Yang, Z.: Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization. Technical Report. In: Nature Inspired Computation and Applications Laboratory. USTC. China. pp. 4-31 (2007)
15. Vesterstrom, J., Thomsen, R.: A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In: Proc. of IEEE Congr. on Evol. Comp., pp. 1980-1987 (2004)
16. Yang, Z., Tang, K., Yao, X.: Self-adaptive Differential Evolution with Neighborhood Search. In: Proc. of the IEEE Congr. on Evol. Comp., pp. 1110-1116 (2008)
17. Yang, Z., Tang, K., Yao, X.: Scalability of generalized adaptive differential evolution for large-scale continuous optimization. In: Soft Computing, vol. 15, issue 11, pp. 2141-2155 (2011)
18. Zaharie, D., Petcu, D.: Adaptive Pareto Differential Evolution and Its Parallelization. In: Proc. of the 5th Int. Conf. Parallel Processing and Applied Mathematics, vol. 3019, pp. 261-268 (2004)