Parallel Processing for Multi-objective Optimization in Dynamic Environments

Mario Cámara[1], Julio Ortega[1], Francisco J. Toro[2]

[1]University of Granada
Dept. of Computer Architecture and Technology,
ETSIIT, 18071, Granada, Spain
{mcamara, julio}@atc.ugr.es

[2]University of Granada
Dept. of Signal Theory, Telematics, and Communications,
ETSIIT, 18071, Granada, Spain
ftoro@ugr.es

Abstract

This paper deals with the use of parallel processing for multi-objective optimization in applications in which the objective functions, the restrictions, and hence also the solutions can change over time. These dynamic optimization problems appear in quite different real-world applications with relevant socio-economic impact. The procedure here presented is based on PSFGA, a parallel evolutionary procedure for multi-objective optimization. It uses a master process that distributes the population among the processors in the system (that evolve their corresponding solutions according to an island model), and collects and adjusts the set of local Pareto fronts found by each processor (this way, the master also allows an implicit communication among islands). Moreover, the procedure exclusively uses non-dominated individuals for the selection and variation, and maintains the diversity of the approximation to the Pareto front by using a strategy based on a crowding distance.

1. Introduction

Many real-world optimization problems are dynamic because there are changes in the conditions on which the cost functions depend, in the restrictions that the solutions must meet, etc. For example, in a scheduling problem the characteristics of the resources and the number of tasks to be allocated could vary over time [1]. In the optimal control of an industrial plant the conditions change due to the ageing of the plant, to random intrinsic effects, etc [2].

On the other hand, many of the optimization problems must optimize more than one objective at a time, which frequently are in conflict. In this context, the concept of optimum must be redefined, because instead of providing only one optimal solution, the procedures applied to these multi-objective optimization problems should obtain a set of non-dominated solutions, known as Pareto optimal solutions [3], from which a decision agent (be human or not) will choose the most convenient in the current circumstances. These solutions are optimal in the sense that there is not any other better solution when all the objectives are taken into account. The Pareto optimal solutions form a hypervolume known as Pareto front. Thus, in the dynamic multi-objective optimization problems the objective functions and the set of variables which define the solution space may change over time.

Evolutionary algorithms [3] have been successfully applied to static multi-objective problems and have contributed to change the perspective on how these problems were tackled by using classical procedures [2]. As evolutionary algorithms steer a population of solutions in a concurrent way by making use of cooperative searching techniques, it could be relatively easy to adapt these algorithms to obtain sets of Pareto optimal solutions.

Our goal in this paper is not only to tackle problems with more than one objective, but also to consider the problems where the objective functions and even the spaces in which the solutions lie (the restrictions they must meet) may change. It is reasonable to think that evolutionary algorithms may also prove to be useful in dynamic optimization problems because they are inspired in the natural evolution and this is a continuous process of adaptation. In [7], a summary can be found about the use of evolutionary algorithms in dynamic optimization problems (not necessarily multi-objective), together with other optimization problems in environments with uncertainty.

1-4244-0910-1/07/$20.00 c2007 IEEE.

A possible way to act whenever a change occurs in the conditions is trying to solve the problem as if it were a new problem. However, instead of starting the search from a random solution set, the process towards the new solutions could be accelerated if it takes advantage of the already known solutions, depending on the characteristics of the change that has happened in the problem.

Anyway, the speed of the reaction to changes is a quite important topic in the context of dynamic optimization. Therefore, the use of high performance computers may turn out very useful for these kinds of problems. Hence, in section 2 the dynamic multi-objective optimization problem is defined along with the measures that can be used to evaluate the performance of the corresponding procedures. Section 3 considers the use of evolutionary procedures and the different approaches to reach convergence in the dynamic optimization problems. In section 4 the benefits that parallel processing can provide are reviewed, and a parallel algorithm for dynamic multi-objective optimization is described. The performance of this parallel procedure is analyzed in section 5, and finally, the conclusions of this paper are given in section 6.

2. Dynamic Multi-objective Optimization

A dynamic multi-objective optimization problem (DMO) can be defined as the problem of finding a vector of decision variables $\boldsymbol{x}(t) \in R^n$, that satisfies a restriction set and optimizes a function vector whose scalar values represent objectives that change over time. Thus, expressed mathematically, it has to be found a decision variable vector $\boldsymbol{x}^*(t) = [x_1^*(t), x_2^*(t), ..., x_n^*(t)]$ that satisfies a given restriction set $\boldsymbol{g}(\boldsymbol{x},t) \leq 0$, $\boldsymbol{h}(\boldsymbol{x},t) = 0$ and optimizes the function vector:

$$\boldsymbol{f}(\boldsymbol{x},t) = \{f_1(\boldsymbol{x},t), f_2(\boldsymbol{x},t), ..., f_m(\boldsymbol{x},t)\}.$$

As the objectives are usually in conflict, the optimization of one of them is carried out at the expense of the other ones. Thus, a trade-off, that implies the concept of Pareto optimality, must be reached. In a dynamic multi-objective optimization problem, a decision vector $\boldsymbol{x}^*(t)$ is said to be a Pareto optimal solution if there is *not any other feasible decision vector*, $\boldsymbol{x}(t)$, that improves one objective without worsening at least one of the other objectives. The set of all non-dominated solutions determines the Pareto front in the objective space. We can define $S_p(t)$ and $F_p(t)$ as the sets of Pareto optimal solutions at time t, respectively in the decision and objective spaces. In [2], it is presented a classification of the dynamic multi-objective optimization problems onto four groups depending on whether the sets $S_p(t)$ and $F_p(t)$ change over time or not.

One of the main questions in the research on dynamic optimization procedures is the evaluation of the performance of these procedures [8,21]. Three characteristics can be considered to carry out that evaluation. They are the *accuracy*, the *stability* and the *reaction capability* of the optimization procedure [8]. The accuracy, *acc*, measures the quality of the solution found; the stability, *stb*, gives us an idea about the effect of the changes in the problem on the algorithm accuracy; and the reaction capacity, *reac*, measures the capability of the optimization algorithm to adapt itself to changes. In [8], measures for the stability and reaction capacity are proposed. They are based on the measure of accuracy, *acc*:

$$stb(t) = max\{0, acc(t-1) - acc(t)\} \quad (1)$$

$$reac(t,\varepsilon) = min\{t'-t \,/\, \left\{ \begin{array}{l} t < t' \leq Mgen \\ \dfrac{acc(t')}{acc(t)} \geq (1-\varepsilon) \end{array} \right\} \cup \{Mgen - t\} \quad (2)$$

where *t'* is a natural number, ε is a fixed real number less than one; *Mgen* is the number of generations, and *acc(t)* is a measure of the solution accuracy at time *t*, that ranges from 0, the worst, to 1, the best. The stability, *stb(t),* also takes values from 0 to 1, but in this case the maximum stability is given by 0. Moreover, as smaller is *reac(t),* the bigger is the reaction capacity of the procedure.

The definition of accuracy, *acc(t),* is not trivial, because the cost functions change over time, and it is even more difficult if the optimal values are unknown. In the case of multi-objective optimization problems, as we want to reach the Pareto front, it should be considered as the reference point in order to evaluate the accuracy of the solutions already found. Moreover, it should be also considered that the location of the Pareto front is usually unknown. Thus, to estimate the procedure accuracy, we use the hypervolume of the non-dominated (in a minimizing problem) or dominated (a maximizing problem) space which is given by the set of solutions at time t, *V(t)* [9]. From *V(t)*, the measure of *acc(t)* can be defined as:

$$acc(t) = \frac{V(t)}{V^{\max}(t)}, \quad (3)$$

for minimization problems, and as:

$$acc(t) = \frac{V^{\min}(t)}{V(t)}, \quad (4)$$

for maximization problems. In the expressions, $V^{max}(t)$ is the maximum hypervolume in the objective space that has been reached through *t* iterations, and $V^{min}(t)$ is the minimum hypervolume that has been reached through *t* iterations. It can be easily seen that *acc(t)* takes values from 0 to 1.

3. Evolutionary Computation for Dynamic Multi-objective Optimization

Evolutionary algorithms have been widely applied to multi-objective optimization, bringing a different point of view on the solution of these problems with regard to the classic methods previously proposed. They can give a very good approach to the Pareto front and to reveal the properties of the optimal solutions [5, 6]. In an evolutionary algorithm, a trade-off is required between exploration and exploitation. Thus, the characteristics of the transformations (mutation, crossover, etc) must be set in order to find a trade-off between the search for solutions in new areas of the space and the convergence towards better solutions in the surroundings of the already found ones.

Thus, diversity and uniform distribution are required in the found solutions in order to provide an accurate description of the Pareto front. Moreover, in dynamic optimization problems, the population of the evolutionary algorithm must react to changes as fast as possible. Some of the main topics that should be addressed are the following ones [4]:

1. *Diversity after the changes.* As soon as a change is detected, diversity should be increased in order to make it ease the evolution towards a new optimum. If the mutation probability is too high, the situation is similar to a re-start of the algorithm and no advantage is obtained from the already found solutions. There are some alternatives, as *hypermutation* [13], a sudden increment in the mutation probability after the change of the conditions, and variable local search [14], where mutation probability is gradually increased.
2. *Diversity along the runtime.* It tries to avoid convergence through the execution of the algorithm so that the population could adapt itself better to changes. There are some alternatives: to insert random migrant solutions in the population in each generation; the thermodynamic genetic algorithms [15]; the use of niching techniques [16] for preserving diversity like sharing or crowding. Of course, the bigger the diversity, the slower the convergence.
3. *Memory based techniques.* The evolutionary algorithm uses a memory that keeps information about what has happened in previous generations [17,18]. This approach is mainly useful when the problem shows conditions that have appeared before.
4. *Multi-population techniques.* The population is divided in subpopulations that hold information about different regions of the search space [19,20]. The idea behind this is to evolve different optimal solutions in each population.

In [7], there is a selection of bibliography on dynamic optimization where these alternatives can be consulted.

4. Parallel Processing for Dynamic Optimization and PSFGA

Parallel processing can be useful to efficiently solve dynamic optimization problems with evolutionary algorithms, not only by improving the quality of the solutions found but also by speeding up the execution times. Two decomposition alternatives are usually implemented in parallel evolutionary algorithms: functional decomposition and data decomposition. The functional decomposition techniques identify tasks that may be run separately in a concurrent way. The data decomposition techniques divide the sequential algorithm in different tasks that are run on different data (i.e. the individuals of the population). Moreover, hybrids methods are also possible.

In this paper, data decomposition has been applied as we consider this alternative more attractive. In an evolutionary algorithm, the evaluation of the objective function and the application of operators to the individuals of the population can be independently done. This allows data parallelization without modifying the convergence behaviour of the sequential algorithm. The fitness evaluation for each individual in the population is frequently the part of the algorithm with the highest computational cost. This is particularly true in non-trivial optimization problems, with large sized populations and/or individuals codified by complex data structures that require big computation times. As a consequence, a suitable parallelization scheme is to concurrently evaluate the fitness of the individuals, by using a *master-worker* structure in which every worker process evaluates a different and unique group of individuals, returning the fitness values to the master process which completes the rest of the algorithm steps. If the individuals are distributed in a balanced way, acceptable speedups could be obtained whenever the evaluation of the solutions require high computation times and the communication costs associated with the distribution of data structures and results between processors are kept low enough.

An alternative to improve the achieved speedup (and to get other benefits as we describe below) is to allow the different processors not only the evaluation of the fitness, but also the application of operators to the individuals of the subpopulation allocated to them. This alternative corresponds to the so called, *island model*. So, the initial population is divided into subpopulations (that could be also associated to different search subspaces) which are evolved separately. Sometimes, individuals can be exchanged between the different subpopulations (*migration*). This kind of parallelization can also improve the diversity of the population during the algorithm convergence and leads to algorithms with better performance than the sequential versions. Thus, together with the advantages derived from the availability of more memory and the use of several CPUs (it is possible to approach more complex problems and/or to speedup the programs execution), the evidences of bigger efficiency and diversity in the population (very useful elements in dynamic multi-objective optimization problems, as it has been indicated in Section 3) justify the use of parallelism in the field of evolutionary algorithms.

Nevertheless, the selection of individuals and the operations required to maintain diversity need comparisons that imply the whole population or a big part of it. This means that data parallelization at this level, especially in the case where there is not any mechanism to share information about the fitness of the individuals between the processes, modifies the behaviour of the algorithm with regard to the sequential version. Thus, it is difficult to predict the behaviour of this kind of parallelization and must be evaluated for each particular implementation.

Moreover, in multi-objective optimization several objectives and the Pareto dominance relationships have to be evaluated at the same time [10-12]. The calculation of the Pareto dominance relationships requires, most of the time, statistics of the whole population. Besides, the computational bottleneck in most of the applications is the evaluation of the objective functions, which may be parallelized by means of distributing the functions between processors, or with a hybrid approach in which each processor evaluates a subset of functions for a subset of the population. After the evaluation of the objective functions, the algorithms with Pareto front-based selection usually calculate dominance measures and the corresponding distances as part of the mechanism for keeping up the diversity. This mechanism is implemented in each case, as a previous step to assign the fitness value to each individual and to select the parents. The parallelization of these tasks is not easy. For example, problems appear in algorithms that usually work with small populations (PAES), in algorithms where the calculation of distances must be done sequentially after the determination of dominance relationships (SFGA) [11], or in those algorithms where the calculation of dominance relationships, distance, and selection takes place at the same time (NPGA) [22].

In principle, the benefits that can be obtained from parallel processing of dynamic multi-objective optimization problems are the same that with *static* multi-objective optimization, but also the possibility of speeding up the capacity of the algorithm reaction, which in turn reduces the needed processing time, which leads to reach a set of non-dominated solutions near to the Pareto front earlier. Thus, dynamic optimization problems, where the change rate is faster, could also be tackled.

The algorithm here described is shown in Figure 1. It is based on PSFGA [11], a parallel algorithm for multi-objective optimization that uses an island model where the processors that execute the islands (workers) implicitly communicate themselves through a master process that divides the population and send the corresponding subpopulations of the same size to the workers. By using the SFGA algorithm, every worker looks for the optimal solutions in the search space that has been assigned to it and keeps only those solutions that are not dominated by the others. After a fixed number of iterations (*genpar*), the workers send the solutions found to the master, who after joining all the solutions into a new population, rule out the dominated solutions. At the same time, the master runs an instance of the SFGA algorithm (along *genser* iterations) over the whole population before sending new subpopulations again to the worker processes. In the master, there is a crowding mechanism for keeping the diversity and the distribution of the solutions on the Pareto front founded. So, after reaching a number of solutions, equal or above to a given percent of the population size, only the non-dominated solutions that are far enough of the other ones are chosen.

```
Master Process
01  Initialize algorithm and random population
02  for i:=1 to n_workers
03      Create subpopulation SP[i] of size subpop_size
04      Send subpopulation SP[i] to i-worker process
05  end
06  t = 1
07  while true
08      for i:=1 to n_workers
09          Receive subpopulation SP[i] from i-worker process
10          Insert SP[i] into P
        end
11      Execute SFGA on P for genser generations
12      Rank individuals in P according to objective function $f_{obj}(t)$
13      if ((t mod $\tau_t$)=0) then gather statistics of the time span
14      Perform Crowding on P if criteria are met
15      Partition of P among subpopulations SP[i] (i=1,..,n_workers) of subpop_size
         elements
16      for i:=1 to n_workers
17          Send subpopulation SP[i] to i-worker process
18      end
19      t=t+1;
20  end

Worker Process
1   while true
2       Receive subpopulation SP[i] from master process
3       Execute SFGA on SP[i] for genpar generations
4       Send SP[i] to master process
5   end
```

Figure 1. PSFGA version for dynamic multi-objective optimization

5. Experimental Results

The algorithm has been evaluated by using two test functions for dynamic multi-objective that have been taken from the problems presented in [2]. In the first test function, FDA1 (5), the Pareto front, $F_p(t)$, is equal to $f_2 = 1 - \sqrt{f_1}$, and does not change, but only the values of the solutions to the corresponding front, $S_p(t)$. In the second function, FDA2 (6), besides to the values of the solutions, $S_p(t)$, also the corresponding Pareto front, $F_p(t)$, changes. In (5) and (6), n_t and τ_t are intended, respectively, to control the speed at which the problem changes and the time interval in which the changes are being considered. In our tests, and as it is suggested in [2], it has been taken $n_t = 10$ and $\tau_t = 5$. The solution sets are $|X_I| = 1$ and $|X_{II}| = 19$ for FDA1, and $|X_I| = 1$ and $|X_{II}| = |X_{III}| = 15$ for FDA2.

The experiments were carried out on an 8-node cluster with two 2 GHz AMD Athlon processors and 2 Gbytes RAM by node, connected via Gigabit Ethernet.

$$\begin{cases} f_1(x_I) = x_1 \\ g(x_{II}) = 1 + \sum_{x_i \in X_{II}} (x_i - G(t))^2 \\ h(f_1, g) = 1 - \sqrt{\dfrac{f_1}{g}} \\ G(t) = \sin(0.5\pi t), \quad t = \dfrac{1}{n_t}\left\lfloor \dfrac{\tau}{\tau_t} \right\rfloor \\ x_I = (x_1) \in [0,1], \\ x_{II} = (x_2, \dots, x_n) \in [-1,1] \end{cases}$$
$$\text{with} \quad n = 20,\ n_t = 10,\ \tau_t = 5 \qquad (5)$$

$$\begin{cases} f_1(x_I) = x_1 \\ g(x_{II}) = 1 + \sum_{x_i \in X_{II}} x_i^2 \\ h(x_{III}, f_1, g) = 1 - \left(\dfrac{f_1}{g}\right)^{\left(H(t) + \sum_{x_i \in X_{III}} (x_i - H(t))^2\right)^{-1}} \\ H(t) = 0.75 + 0.7\sin(0.5\pi t), \ t = \dfrac{1}{n_t}\left\lfloor \dfrac{\tau}{\tau_t} \right\rfloor \\ x_I = (x_1) \in [0,1],\ x_{II}, \\ x_{II}, x_{III} = (x_2, \dots, x_n) \in [-1,1] \end{cases}$$
$$\text{with} \quad n = 16,\ n_t = 10,\ \tau_t = 5 \qquad (6)$$

The results provided in the following correspond to five runs of each experiment. In Figure 2, it is shown the Pareto front for FDA1 in the first five time intervals. In all of them, the found solutions approximate the actual Pareto front accurately, although all the values of the corresponding solutions should change in order to adapt them to the new dynamic function requirements.

With regard to FDA2, Figure 3 shows the solutions that have been obtained from t = 50 to t = 150. These solutions approach the Pareto fronts changed each $\tau_t = 5$. However, it is more difficult to reach them in the non-convex areas of the objective functions. In this case, when the generation changes, and so the current Pareto front, the values of the solution space also change accordingly. Table I shows the values for *acc, stb* and *reac* obtained by our procedure in FDA2. The stability could be improved in some cases (those in which $stb \neq 0$), mainly in the non-convex areas. On the other hand, the reaction capacity is also good enough, because *reac* is always equal to five. This is the smallest value it could take in our implementation, as this is the minimum time span in which functions are evaluated again, and it is long enough to allow the algorithm to adapt itself to the new Pareto fronts.

The use of the parallel processing gives a twofold improvement. It allows a reduction in the execution time required to reach a good approximation to the new Pareto fronts, thus widening the field of the problems that can be tackled (problems with faster rate of change in the Pareto front). On the other hand, each worker can run more

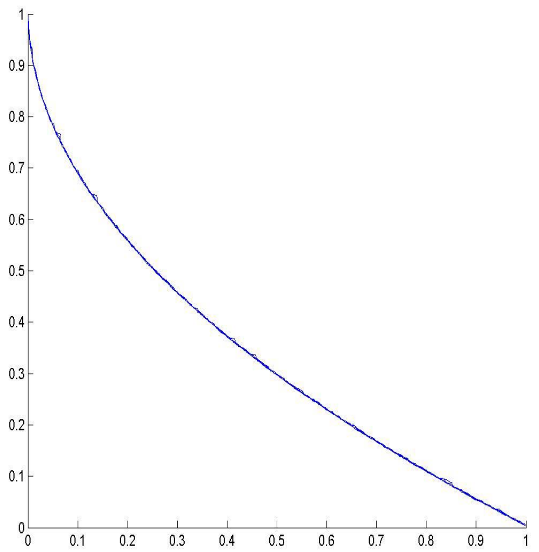

Figure 2. Pareto front and location of the solutions for FDA1 in 5 time intervals (between t = 5 and t = 25).

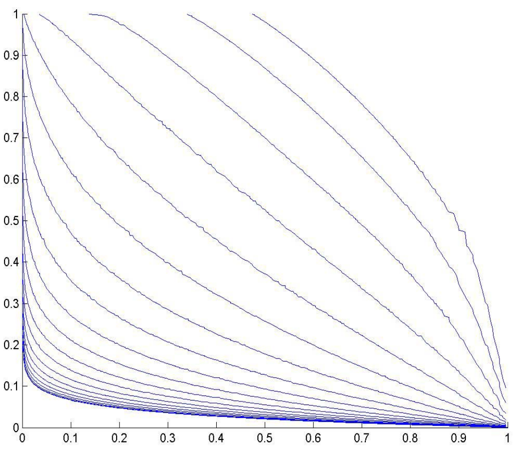

Figure 3. Location of the solutions (Pareto fronts approximated by our algorithms) for FDA2, from iteration t = 50 to t = 150 (with steps τ_t = 5).

iterations in the same amount of time, thus increasing the explored search space and making it easy the adaptation to changes.

Table 2 shows the speedups reached by our parallel algorithm as more workers are added to the parallel procedure. The speedups correspond to the rate between the time required by the parallel algorithm executed in *n+1* processors (the master and *n* workers) to get the solutions after τ_i iterations, and the time required by the SFGA algorithm (for the same number of iterations).

Table 1
Results for *acc*, *stb*, and *reac* in FDA2

τ_i	*Hypervolume*	*acc*	*stb*	*reac* (ε = 0.1)
5	0.345 ± 0.005	0.97 ± 0.01	0.000	5
10	0.350 ± 0.005	0.99 ± 0.01	0.000	5
15	0.355 ± 0.005	1.00 ± 0.01	0.000	5
20	0.347 ± 0.005	0.98 ± 0.01	0.012	5
25	0.352 ± 0.005	0.99 ± 0.01	0.000	5
30	0.347 ± 0.005	0.98 ± 0.01	0.015	5
35	0.347 ± 0.005	0.98 ± 0.01	0.000	5
40	0.344 ± 0.005	0.97 ± 0.01	0.010	5
45	0.349 ± 0.005	0.98 ± 0.01	0.000	5
50	0.345 ± 0.005	0.97 ± 0.01	0.009	5
55	0.347 ± 0.005	0.98 ± 0.01	0.000	5
60	0.336 ± 0.005	0.95 ± 0.01	0.029	5
65	0.341 ± 0.005	0.96 ± 0.01	0.000	5
70	0.353 ± 0.005	0.99 ± 0.01	0.000	5
75	0.340 ± 0.005	0.96 ± 0.01	0.035	5
80	0.344 ± 0.005	0.97 ± 0.01	0.000	5
85	0.347 ± 0.005	0.98 ± 0.01	0.000	5
90	0.339 ± 0.005	0.96 ± 0.01	0.021	5
95	0.349 ± 0.005	0.98 ± 0.01	0.000	5
100	0.347 ± 0.005	0.98 ± 0.01	0.006	5

It can be checked that the speedup is super-linear, especially for two and four workers.

Although the speedup keeps being super-linear for eight processes, it does not increase so much between four and eight processes. Anyway, there is a clear trend towards a reduction in the computation time. The improvement in the performance of the parallel algorithm has much to do with the super-linear behaviour, because it works with more diversified populations.

Table 2
Speedup results for FDA2.

τ_i	Number of workers			
	2	*4*	*8*	*16*
5	5,25	8,4	10,5	10,5
10	4,56	8,2	8,2	8,2
15	4,67	10,5	8,4	8,4
20	4,67	8,4	10,5	10,5
25	4,78	8,6	10,75	10,75
30	4,78	8,6	8,6	10,75

To evaluate the quality of the solutions we use the arithmetic mean of the hypervolumes that have been obtained for a given time interval. From that measure we can compare the solutions found by the algorithm when a different number of worker processes is used. Table 3 shows the values which have been obtained for FDA2 from times 5 to 200, at steps of 5 iterations, and using between 1 and 32 workers.

Table 3
Quality of the solutions depending on the number of workers for FDA2.

Workers	Mean hypervolume
1	0.2458 ± 0.0007
2	0.2463 ± 0.0007
4	0.2472 ± 0.0007
8	0.2474 ± 0.0007
16	0.2472 ± 0.0007
32	0.2474 ± 0.0007

As it can be seen, the quality of the solutions worsens slightly as the number of workers used to solve the problem increases.

6. Conclusions

The first results of our approach to the possibilities of parallel processing for dynamic multi-objective optimization are quite acceptable. First of all, it has been shown the ability of our parallel procedure to reach solution sets quite near to the changing Pareto fronts. This procedure is an adaptation to dynamic environments of the PSFGA algorithm for multi-objective optimization [11]. It uses a master process to distribute the population of solutions among the processors that evolve their corresponding subpopulations for *genpar* iterations. Then, the master collects the (partial) Pareto fronts independently determined by the worker processors, builds a whole Pareto front from the partial ones, executes *genser* iterations of the evolutionary algorithm, and distributes the obtained population of solutions again. Thus, the procedure we have proposed allows a continuous transition between a master-worker operation model, when *genpar* is set to 0 (the workers only compute the fitness of their subpopulations) and an island model (*genpar*>0) where the processors communicate through the master.

On the other hand, the speedup results obtained allow a reduction in the convergence times, and hence, the ability to satisfy stronger time restrictions in the dynamic problem. We consider that the super-linear speedups that have been observed in some cases show the usefulness of parallel processing in keeping up the diversity of the population, in the improvement of the reaction capability and in the algorithm adaptability.

It is clear that there are many things to do yet. On one side, we think that many algorithm characteristics and parameters should be analyzed and optimized, both in the sequential and parallel versions of the algorithm. Thus, we plan to study the scalability and performance behaviour for different versions of the algorithm in which the worker and master processes run asynchronously, with different communication schemes (including the ability of direct communications between workers), and *genser/genpar* rates. We also plan to consider other more flexible schemes where, for example, more than one process acts as a master at a given time. Furthermore, it is also necessary to evaluate the performance of the procedure with a broader set of benchmarks and some real world applications.

Acknowledgements—This work has been done with funding from the project TIN2004-01419 (by the Spanish Ministry of Science and Technology), and the program for early stage researchers by the Andalusian government, co financed with FEDER funds.

References

[1] Branke, J.; Mattfeld, D.C.: "Anticipation and flexibility in dynamic scheduling". *International Journal of Production Research*, Vol.43, No.15, pp.3103-3129. August, 2005.

[2] Farina, M.; Deb, K.; Amato, P.: "Dynamic Multi-objective Optimization Problems: Test cases, Approximations, and Applications". *IEEE Trans. on Evolutionary Computation*, Vol.8, No.5, pp.425-342. October, 2004.

[3] Coello, C.A.: "An Updated Survey of GA-Based Multi-objective Optimization Techniques", Technical Report Lania-RD-98-08, Laboratorio Nacional de Informática Avanzada (LANIA), Mexico, 1998.

[4] Jin, Y.; Branke, J.: "Evolutionary Optimization in Uncertain Environments – A Survey". *IEEE Trans. on Evolutionary Computation*, Vol.9, No.3, pp.303-317. June, 2005.

[5] Coello Coello, C.A, Van Veldhuizen, D.A, Lamont, G.B. "*Evolutionary Algorithms for Solving Multi-Objective Problems*". Kluwer Academic Publishers, 2002.

[6] Bibliography on Evolutive Algorithms for Multiobjective Optimization: http://www.lania.mx/~ccoello/EMOO/

[7] EvoDOP (Evolutionay Algorithms for Dynamic Optimization Problems): http://www.aifb.uni-karlsruhe.de/~jbr/EvoDOP/

[8] Waiker, K.: "Performance Measures for Dynamic Environments". *Parallel Problem Solving from Nature*, LNCS 2439, pp.64-73, 2002.

[9] Zitzler, E.; Deb, K.; Thiele, L.: "Comparison of Multi-objective Evolutionary Algorithms: Empirical Results". Tech. Report 70, ETH Zurich, December, 1999.

[10] Van Velhuizen, D.A.; Zydallis, J.B.; Lamont, G.B.: "Considerations in Engineering Parallel Multi-objective Evolutionary Algorithms". *IEEE Trans. Evolutionary Computation*, Vol. 7, No.2, pp.144-173. April, 2003.

[11] Toro, F.; Ortega, J.; Ros, E.; Mota, B.; Paechter, B.; Martín, J.M.: "PSFGA: Parallel processing and evolutionary computation for multi-objective optimization". *Parallel Computing*, Vol. 30, pp.721-739, 2004.

[12] Toro, F.; Ros, E.; Mota, S.; Ortega, J.: "Evolutionary Algorithms for Multi-objective and Multimodal Optimization of Diagnostic Schemes". *IEEE Trans. on Biomedical Engineering*, Vol.53, No.2, pp. 178-189. February, 2006.

[13] Cobb, H.J.; Grefenstette, J.J.: "Genetic Algorithms for tracking changing environments". *Proc. of the 5th Intl. Conference on Genetic Algorithms* (S. Forrest, Ed.), Morgan Kaufmann Pub., pp.523-530, 1993.

[14] Vavak, F.; Jukes, K.; Fogarty, T.C.: "Adaptive combustion balancing in multiple burner boiler using a genetic algorithm with variable range of local search". *7th Intl. Conference on Genetic Algorithms* (T. Bäck, Ed.), Morgan Kaufmann Pub., pp. 719-726, 1997.

[15] Mori, N.; Kita, H.; Nishikawa, Y.: "Adaptation to cahnging environment by means of the feedback thermodynamical genetic algorithm". *Parallel Problem Solving from Nature*, LNCS, 1498, Springer, pp.149-158, 1998.

[16] Cedeño, W.; Vemuri, V.R.: "On the use of Niching for Dynamic Landscapes". *IEEE Intl. Conference on Evolutionary Computation*, pp.361-366. April, 1997.

[17] Branke, J.: "Memory enhanced evolutionary algorithms for changing optimization problems". *Proc. of the Congress on Evolutionary Computation*, Vol.3, pp.1875-1882, 1999.

[18] Yang, S.: "Population-based incremental learning with memory scheme for changing environments". *Proc. of the Genetic and Evolutionary Computation Conference*, Vol.1, pp.711-718, 2005.

[19] Ursem, R.K.: "Multinational Gas Optimization techniques in dynamic environments". *Genetic and Evolutionary Conference* (D. Whitley et al., Eds.), Morgan Kaufmann Pub., pp.19-26, 2000.

[20] Branke, J.; Kaussler, T.; Schmidth, C.; Schmeck, H.: "A multi-population approach to dynamic optimization problems". *Adaptive Computing in Design and Manufacturing*, pp. 299-308, 2000.

[21] Morrison, R.: "Performance measurement in dynamic environments". *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems* (J. Branke, Ed.), pp.5-8, 2003.

[22] Horn, J; Nafpliotis, N.: "Multiobjective optimization using the niched Pareto genetic algorithm". IlliGAL Rep. No. 93005, Illinois Genetic Algorithm Laboratory, University of Illinois at Urbana-Champaign, 1993.