

A Study of Master-Slave Approaches to Parallelize NSGA-II

Juan J. Durillo, Antonio J. Nebro, F. Luna and E. Alba

Departamento de Lenguajes y Ciencias de la Computación
E.T.S.I. Informática
Universidad de Málaga (Spain)
{durillo,antonio,flv,eat}@lcc.uma.es

Abstract

Many of the optimization problems from the real world are multiobjective in nature, and the reference algorithm for multiobjective optimization is NSGA-II. Frequently, these problems present a high complexity, so classical metaheuristic algorithms fail to solve them in a reasonable amount of time; in this context, parallelism is a choice to overcome this fact to some extent. In this paper we study three parallel approaches (a synchronous and two asynchronous strategies) for the NSGA-II algorithm based on the master-worker paradigm. The asynchronous schemes are designed to be used in grid systems, so they can make use of hundreds of machines. We have applied them to solve a real world problem which lies in optimizing a broadcasting protocol using a network simulator. Our experiences reveal that significant time reductions can be achieved with the distributed approaches by using a grid system of more than 300 processors.

1. Introduction

Many sectors of the industry are concerned with complex problems of great dimension that must be optimized. Most of these problems have more than one function to optimize, which are frequently contradictory. This kind of problems is known as *multiobjective optimization problems* (MOPs). Multiobjective optimization is a discipline devoted to solve this kind of problems. The solution to a given MOP, in contrast to *single-objective* optimization, is not given by a single solution, but a set of *non-dominated* solutions, known as *Pareto optimal set*. The representation of this set on the objective space is known as *Pareto front*.

As in single-objective optimization, there are two different ways for solving a MOP. On the one hand,

exact techniques allow to obtain the optimal solution to a given problem; however, these techniques become unfeasible when they must deal with NP-hard problems, because of the huge amount of time needed to solve them. On the other hand, *stochastic techniques* are approximate strategies, so they do not guarantee to obtain optimal solutions, but they can produce high quality solutions in a reasonable amount of time.

Metaheuristics are a family of stochastic techniques that have been used to solve optimization problems [15]. Among them, multiobjective evolutionary algorithms (MOEAs) are widely used for solving MOPs [7][9] because they produce a complete set of solutions in a single run. In multi-objective optimization, NSGA-II [10], a genetic algorithm (GA), is the most widely known metaheuristic, and it is considered the reference algorithm in the field.

Although metaheuristics allow to reduce the time needed to produce satisfactory solutions for a given MOP, they have limitations to solve MOPs whose objective functions require minutes and hours to be computed, as use to happen in many real world problems. In this context, parallelism is a choice to overcome this issue to some extent. This way, a parallel MOEA executed in a system composed of many processors can provide solutions to a MOP in an acceptable time. This idea is particularly interesting when using grid systems [14], which are large-scale distributed systems composed of potentially thousands of computing resources that take advantage of the Internet infrastructure to provide a huge unique virtual supercomputer.

In this paper we study three parallel approaches for the NSGA-II metaheuristic on the grid. The three schemes are based on the master-slave paradigm. To address our study, we carry out firstly a comparison of the original NSGA-II algorithm, which is a generational GA, against a steady state version of it. This comparison is intended to provide us with an insight

of the influence of using a steady state approach in NSGA-II, because parallel schemes of steady state GAs have been successfully applied in a similar context in mono-objective optimization (e.g., see [20]), so it is an interesting choice to be applied to NSGA-II. Secondly, we propose the following parallel schemes to NSGA-II according to whether the resulting parallel GA is synchronous or asynchronous: synchronous generational, asynchronous generational, and asynchronous steady state. Finally, we apply all the approaches to solve a real world MOP from the telecommunication domain.

The rest of this work is structured as follow. Section 2 contains a brief review of related works of parallel multiobjective metaheuristics. Sections 3 and 4 are devoted to describe the NSGA-II algorithm and its parallel versions, respectively. In Section 5 we describe the test that we have carried out with the sequential versions of NSGA-II and in Section 6 we discuss the obtained results. Section 7 presents the application of a parallel approach to the optimization of a broadcasting protocol. Finally, Section 8 includes the conclusion and future work.

2. Related Works

In this section we review works related to proposals dealing with the parallelization of NSGA-II. As it is the most popular multiobjective metaheuristic, many researchers have proposed parallel versions based on it. A number of works are related to apply an island model to distribute the search for the Pareto front among several processors. We briefly comment three of them. In [11], a guided dominance approach is used to allocate to processors tasks of finding particular regions of the Pareto front, although these have the requirement of being convex. A geometrical approach to subdivide the search for the Pareto front is presented by Branke *et al.* in [5]; the resulting island-based NSGA-II performs well on bi-dimensional MOPs, but the distribution of individuals is not good on problems with three objectives. Finally, Streichert *et al.* [21] propose the use of clustering algorithms and the combination of the island model with a divide and conquer strategy; as a result, they prove that their distributed NSGA-II does not perform well in the considered test MOPs.

Concerning to the application of the master-slave to NSGA-II, some authors have used this paradigm with the goal of reducing the time needed by the sequential algorithm, like [3][4][8][6]. In these works no details about the number of machines or the environment are given and in all of them a synchronous behavior (i.e., the individuals are evaluated in parallel, but the behavior of the parallel algorithm is the same than an equiv-

alent sequential version) is used. A more detailed description of the parallel approach applied to NSGA-II is given in [16]. The proposed algorithm aims at adapting the grain of computation in the slaves with the idea in mind of executing the resulting parallel program in heterogeneous systems (e.g., grid computing systems). The hardware platform is composed of 50 PCs running Linux, belonging to four clusters. The reported results indicate a high search ability of the proposed model combined with a good performance concerning the computing resources.

Compared with these proposals, our work is unique in the following issues: the analysis of synchronous versus asynchronous master-slave models based on NSGA-II, the statistical methodology applied, and the use of more than 300 processors.

3. The NSGA-II Algorithm

This section is devoted to describe NSGA-II and to present its steady state version.

The NSGA-II algorithm was proposed by Deb *et al.* [10]. It is a GA based on obtaining a new offspring population from the original one by applying the typical genetic operators (selection, crossover, and mutation); then, the individuals in the two populations are sorted according to their rank, and the best solutions are chosen to create a new population. In the case of having to select some individuals with the same rank, a density estimation based on measuring the crowding distance to the surrounding individuals belonging to the same rank is used to get the most promising solutions.

NSGA-II is a generational algorithm, in which there is a current population that is used to create an auxiliary one (the offspring population); after that, both populations are combined to obtain the new current population. Typically, both the current and the auxiliary populations have the same size.

An alternative to a generational GA is a steady state GA, which basically means that there is only a population, without using an auxiliary one. This way, new individuals are incorporated immediately in the evolutionary cycle, so parents and offspring coexist in the same population. As a consequence, steady state GAs are characterized by incrementing the intensification capability of the search process.

A steady state version of NSGA-II can be easily implemented by using an offspring population of size 1. This means that the ranking and crowding procedures have to be applied each time a new individual is created, so the computational complexity of the algorithm will increase notably. As we are interested in solving

computational expensive MOPs, the computing time of the algorithm is not so important. Furthermore, our idea is to use this scheme as the basis of a parallel versions, so what really matters in this context is to know about the search capability of the steady state version against the original NSGA-II. In the rest of this work we will refer to the steady state version as NSGA-II_{ss} and, in the same manner, the original algorithm will be named NSGA-II_{gen}.

4. Parallel versions of NSGA-II

In this section we describe the three parallel approaches that we have proposed for NSGA-II. As mentioned earlier, all of them are based on the master-slave model.

4.1 Synchronous NSGA-II (NSGA-II_{gen}^{syn})

A direct application of the master-slave to NSGA-II lies simply in evaluating the offspring population in parallel. We refer to this version as synchronous generational NSGA-II, or NSGA-II_{gen}^{syn} for short. In this approach, a master process executes the base code of NSGA-II, and the workers evaluate the objective functions of each new individual. This way, if the population size is N , the same number of individuals are created and sent to the workers for evaluation; when all of them have been evaluated and returned to the master, the offspring population is filled and a new generation of NSGA-II is completed.

The behavior of NSGA-II_{gen}^{syn} is the same than que sequential algorithm, and its parallel performance will depend on the number of available processors. It is clear that a maximum of N processors will be used, so this algorithm will not take advantage of a higher number of processors.

It is worth mentioning that many variants of this scheme could be envisioned; for example, given a new individual, each of its objective functions could be evaluated in parallel, thus making use of number of processors higher than N . In this study we are interested in applying basic schemes, so we will not consider further variants of the proposed algorithms.

4.2 Asynchronous Generational NSGA-II (NSGA-II_{gen}^{asy})

The idea in this approach is to make use of all the available processors, which use to be larger than the population size of the algorithm in grid systems (typically, the value of N is 100). With this goal in mind, the asynchronous generational NSGA-II (NSGA-II_{gen}^{asy})

is based on NSGA-II_{gen}^{syn} but, instead of creating N individuals in the master, if the number of processors is P , then P individuals will be generated. Furthermore, when an evaluated individual is received by the master, a new one is created and sent to an idle worker; in fact, if W idle workers are detected, W individuals will be created. The NSGA-II_{gen}^{asy} algorithm is still generational, because when the offspring population is filled, a new generation takes place, leading to a new population.

In NSGA-II_{gen}^{asy}, the master does not have to wait until all the individuals of a generation have been evaluated; hence, we consider it as having an asynchronous behavior. As a consequence, the search capabilities of this algorithm are different compared to NSGA-II, because now it is possible that individuals which are generated *later* can be inserted into the evolutive cycle before than individuals generated *earlier*. An advantage of NSGA-II_{gen}^{asy} is that all the processors in the system can be used in the computation.

4.3 Asynchronous Steady State NSGA-II (NSGA-II_{ss}^{asy})

As commented before, the steady state scheme in GAs allows to improve the exploitation of the search, so the purposes of the asynchronous steady state NSGA-II (NSGA-II_{ss}^{asy}) is twofold: to achieve this behavior in a master-slave NSGA-II and to reduce the time needed to solve MOPs using as many processors as possible.

In NSGA-II_{ss}^{asy}, the master also generates as many individuals as available processors. Once each evaluated solution is received, it is inserted in the population applying ranking and crowding. As in NSGA-II_{gen}^{asy}, when idle workers are detected, new individuals are created and sent for evaluation.

4.4 Implementation Details

This section is aimed at describing the software that we have used to implement our proposals and to deploy them in a parallel environment.

We have used the NSGA-II implementation provided by jMetal [13]. jMetal stands for Metaheuristic Algorithms in Java, and it is an object-oriented Java-based framework aimed at facilitating the development, experimentation, and study of metaheuristics for solving multiobjective optimization problems (MOPs). jMetal is open source software, and it can be freely obtained from <http://neo.lcc.uma.es/metal>.

To deploy our algorithms in a distributed environment we have implemented a software tool called *sparrow*. Sparrow is inspired in MW [19], a C++ frame-

work developed on top of Condor [22] to implement master-worker applications in grids. Our motivation when developing Sparrow is that our needs are related to Java programs, so MW is not adequate for us, and existing Java based platforms (e.g., Proactive [2]) are too heavy-weighted for our purposes.

Sparrow is written using JDK 5.0, and no additional libraries are needed to use it. It is composed of three components: worker, driver, and workerServer. Each worker runs on a processor, and its behavior lies in executing a task sent by the driver; the workerServer contains information about the workers (how many are available, how many are computing tasks, etc); finally, the driver executes the master-slave application. To develop an application with Sparrow, the programmer has to define the task to be computed by the workers, the result returned when a task is completed, and some methods defining the main body of the algorithm.

The current version of Sparrow contains some desirable properties. First, as it is written in Java, it is portable to many platforms. Second, it is easy to deploy and to be used, since the users only have to re-define a small set of methods. Finally, it incorporates mechanisms providing fault tolerance and allowing to discover new machines in the system.

5. Methodology of the Experiments

This section is devoted to describing the methodology that we have used in the experiments carried out in this work. First we present the benchmark problems that we have used to compare NSGA-II and its steady state version, NSGA-II_{ss}. After that, we present the quality indicators used to measure the search capability of our proposals. Finally, we describe the statistical test that we have applied to ensure the confidence of the obtained results.

5.1 Test Problems

To have an insight of the search capability differences between NSGA-II_{gen} and NSGA-II_{ss}, we have chosen the ZDT family of MOPs as benchmark [24]. These problems are well-known and have been used in many studies in this area. This benchmark is composed of five bi-objective problems [24]: ZDT1 (convex), ZDT2 (nonconvex), ZDT3 (nonconvex, disconnected), ZDT4 (convex, multimodal), and ZDT6 (nonconvex, nonuniformly spaced).

5.2 Quality Indicators

To assess the performance of algorithms on the test problems, two different issues are normally taken into account: the distance between the Pareto front generated by the proposed algorithm to the exact Pareto front should be minimized and the spread of solutions found should be maximized in order to obtain as smooth and uniform a distribution of vectors as possible. According to these two properties, the indicators to measure the quality of the obtained Pareto fronts can be classified into three categories depending on whether they evaluate the closeness to the Pareto front, the diversity in the solutions obtained, or both [9].

We have adopted one indicator of each type. The Inverted Generational Distance (*IGD*) measures how far the elements are in the Pareto optimal set from those in the set of non-dominated vectors [23]; the Spread is a diversity metric that measures the extent of spread achieved among the obtained solutions [10]. Finally, we include the Hypervolume (*HV*) indicator, which measures both convergence and diversity [25].

To apply the *IGD* and the *Spread* indicators it is necessary to know the exact location of the true Pareto front. In the case of the problems belonging to the ZDT family they are known, while in the case study (see Section 7) the front is unknown. For this reason, we will use only the *HV* quality indicator on the real-world MOPs.

5.3 Statistical Tests

Since we are dealing with stochastic algorithms and we want to provide the results with confidence, we have made 30 independent runs of each experiment, and the following statistical analysis has been performed throughout this work [12]. Firstly, a Kolmogorov-Smirnov test was performed in order to check whether the values of the results follow a normal (gaussian) distribution or not. If the distribution is normal, the Levene test checks for the homogeneity of the variances. If samples have equal variance (positive Levene test), an ANOVA test is done; otherwise a Welch test is performed. For non-gaussian distributions, the non-parametric Kruskal-Wallis test is used to compare the medians of the algorithms. In the case of normal distributions, we include the mean \bar{x} and the standard deviation σ_n in the tables containing the results; otherwise, we use the median \tilde{x} and interquartile range *IQR*.

We always consider in this work a confidence level of 95% (i.e., significance level of 5% or *p*-value under 0.05) in the statistical tests, which means that the dif-

Table 1. NSGA-II_{gen} vs NSGA-II_{ss} (IGD)

Problem	NSGAII _{gen}		NSGAII _{ss}		
	\bar{x}_{IGD}	\hat{x}_{IGD}	\bar{x}_{IGD}	\hat{x}_{IGD}	
ZDT1	1.896e-04	1.1e-05	1.374e-04	1.3e-06	+
ZDT2	1.928e-04	1.8e-05	1.416e-04	2.1e-06	+
ZDT3	2.534e-04	1.7e-05	1.955e-04	4.3e-06	+
ZDT4	2.073e-04	4.6e-05	1.988e-04	7.7e-05	-
ZDT6	3.606e-04	7.1e-05	1.799e-04	1.9e-05	+

Table 2. NSGA-II_{gen} vs NSGA-II_{ss} (Spread)

Problem	NSGAII _{gen}		NSGAII _{ss}		
	\bar{x}_{IGD}	\hat{x}_{IGD}	\bar{x}_{IGD}	\hat{x}_{IGD}	
ZDT1	3.747e-01	3.1e-02	7.390e-02	8.5e-03	+
ZDT2	3.854e-01	3.7e-02	7.807e-02	1.0e-02	+
ZDT3	7.481e-01	1.7e-02	7.034e-01	3.4e-03	+
ZDT4	4.027e-01	5.2e-02	1.267e-01	2.0e-02	+
ZDT6	3.554e-01	4.1e-02	1.013e-01	1.2e-02	+

ferences are unlikely to have occurred by chance with a probability of 95%.

6. NSGA-II_{gen} versus NSGA-II_{ss}

This section is aimed at analyzing the results obtained when comparing the two versions of NSGA-II.

We have used the same configuration for both algorithms. The operators for crossover and mutation are SBX and polynomial mutation, with distribution indexes of $\eta_c = 20$ and $\eta_m = 20$, respectively. A crossover probability of $p_c = 0.9$ and a mutation probability $p_m = 1/n$ (where n is the number of decision variables) are used. The population size is 100 individuals.

Table 1 shows the results of the *IGD* quality indicator for the two variants of the algorithm. We observe that the steady state version obtains the best (lowest) on all the problems with statistical confidence (see ‘+’ symbols in the last column of Table 1). The values related to problem ZDT4 have not statistical confidence (see the symbol ‘-’ in the right column).

The values obtained after applying the *Spread* quality indicator are included in Table 2. We can observe that the steady state version of NSGA-II obtains the best values for all the problems. All the results are significant.

Finally, Table 3 shows the results obtained for the *HV* indicator. According to this indicator, NSGA-II_{ss} obtains the highest (best) values in four out of the five problems. We can observe that in the case of ZDT4, the results have not statistical confidence, so the algorithms are statistically equivalent according to this metric in this problem.

To illustrate the search capabilities of the two variants of NSGA-II, we include the Pareto fronts obtained when solving the ZDT1 problem in Figure 1. Clearly,

Table 3. NSGA-II_{gen} vs NSGA-II_{ss} (HV)

Problem	NSGAII _{gen}		NSGAII _{ss}		
	\bar{x}_{IGD}	\hat{x}_{IGD}	\bar{x}_{IGD}	\hat{x}_{IGD}	
ZDT1	6.593e-01	5.0e-04	6.616e-01	1.5e-04	+
ZDT2	3.261e-01	4.4e-04	3.282e-01	1.2e-04	+
ZDT3	5.148e-01	2.1e-04	5.157e-01	1.3e-04	+
ZDT4	6.563e-01	3.8e-03	6.560e-01	4.9e-03	-
ZDT6	3.878e-01	2.8e-03	3.960e-01	7.1e-04	+

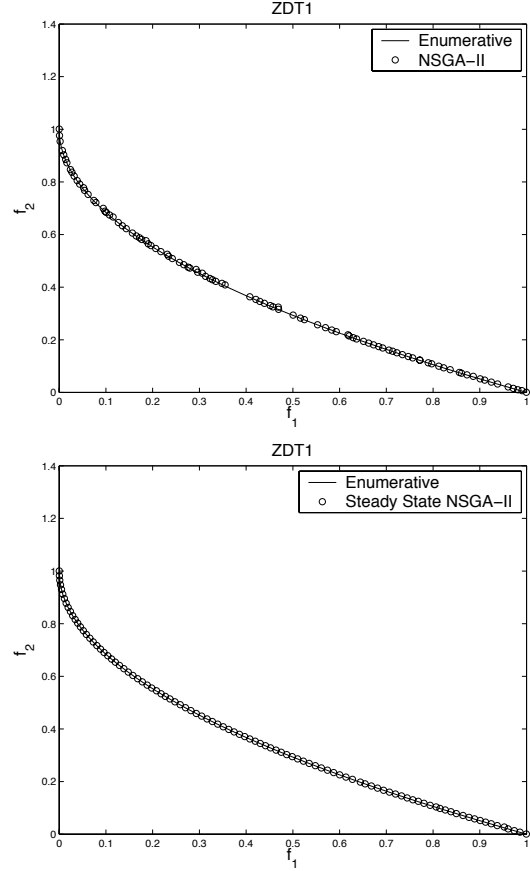


Figure 1. Pareto front for problem ZDT1 obtained with NSGA-II_{gen} (top) and NSGA-II_{ss} (bottom).

the front generated by the steady state version (Figure 1, bottom) presents an almost perfect convergence and spread, while the one obtained with the original NSGA-II algorithm (Figure 1, top) clearly fails in obtaining a uniform diversity.

If we deep into the two compared algorithms and in the results presented in this section, the behavior of NSGA-II_{ss} was somehow expected. The fact that we apply the ranking and crowding procedures each time a new individual is created eliminates precisely some situations in which the crowding distance does not per-

form well [18]. Obviously, these improved search capabilities do not come for free, and steady state version is computationally more expensive than the original NSGA-II. We have measured the times to solve the ZDT1 problem on a MacBook Intel Core 2 Duo 2GHz processor based laptop, running MacOS 10.4. Using Java JDK 1.5, NSGA-II_{gen} requires 2.78 s while NSGA-II_{ss} takes 50.4 s. As commented before, this increment in the computing time can be unacceptable when solving problems such as those composing the ZDT family. We have to consider that the time of executing the algorithm is roughly constant, so in the case of solving real world problems, which can require minutes or hours to evaluate the objective functions, spending 50 s can be acceptable.

7 A case of study: Optimize a Broadcasting Protocol in MANETs

In this section we describe first the problem of optimizing a broadcasting protocol in MANETs, which is the real world problem we have chosen to test the master-slave versions of NSGA-II proposed in this work. Then, we present and analyze the obtained results.

7.1 Definition of the problem

Mobile Ad Hoc Networks (MANETs) are fluctuating networks populated by a set of communicating devices called nodes (or devices) which can spontaneously interconnect each other without any pre-existing infrastructure. This means that no organization is present in such networks as it is usual in communication networks. Here we are interested in studying the optimization of the problem of broadcasting on this kind of networks. The scenario that we want to optimize is a *mall* environment.

The general principle of broadcasting is as follows: starting from a source node a message needs to be forwarded to all nodes in the network. In a given MANET, due to host mobility, broadcasting is expected to be performed very frequently. Additionally, broadcasting may also serve as a last resort to provide multicast services in networks with such rapidly changing topologies and systems for the organization of terminals in groups. Hence, having a well-tuned broadcasting strategy will result in a major impact in network performance.

Optimizing a broadcasting strategy implies multiple goals to be satisfied at the same time, such as maximizing the number of devices reached (coverage), minimizing the network use (bandwidth) and/or minimiz-

ing the duration of the process. The problem that we are considering consist of, given a model of a MANET environment as input, to select the best parameter settings for the broadcasting strategy. To deal with such kind of networks we have employed software simulators. A simulation tool for this kind of networks is *Madhoc* [17], a metropolitan MANET simulator.

Summarizing, the considered MOP has three objectives and five decision variables, and we have selected as scenario to optimize a mall environment. More details about the formulation of the problem or the properties of the scenario can be found in [1].

7.2 Obtained Results

Before commenting the results, we give next information about the grid system used to carry out these experiments. We have used up to 330 processors belonging to eight laboratories of the Computer Science Department at the University of Málaga. Each machine runs a flavour of Linux (Fedora 5, Fedora 6, and Debian 4.0). We have to point out that most of the computers are equipped with Intel Core 2 Duo processors, so we consider them as bi-processors.

We start by comparing first the results obtained with the sequential versions of NSGA-II when solving the broadcasting problem. In Table 4 we show the results obtained after applying the *HV* indicator to the obtained fronts. We observe that NSGA-II_{gen} yields a higher (better) value of the metric, but the results have no statistical confidence, as it is pointed out by the symbol “-” in the last column of the table. Thus, we cannot ensure that a version improves the other in this problem.

We analyze now the results of the parallel approaches. Here we are interested in two issues, the quality of the obtained fronts and the suitability of the algorithms to take advantage of the available processing resources. To decide about the first issue, we need to observe the *HV* values, which are contained in Table 5. According to the table, the more accurate algorithm is NSGA-II_{gen}^{syn}, followed by NSGA-II_{ss}^{asy} and NSGA-II_{gen}^{asy}. The last column shows that there are statistical confidence in the obtained results.

As the values of the *HV* indicator are similar, including the sequential and parallel versions, we decided

Table 4. *HV* Quality Indicator for the Sequential NSGA-II

NSGA-II _{gen}	NSGA-II _{ss}
\bar{x}_{σ_n}	\bar{x}_{σ_n}
8.677e-01 ± 4.7e-03	8.672e-01 ± 4.1e-03 -

Table 6. Statistical Test Between Pairs

	NSGA-II _{gen}	NSGA-II _{ss}	NSGA-II _{gen} ^{syn}	NSGA-II _{gen} ^{asy}	NSGA-II _{ss} ^{asy}
NSGA-II _{gen}		-	-	-	-
NSGA-II _{ss}	-		-	-	-
NSGA-II _{gen} ^{syn}	-	-		+	-
NSGA-II _{gen} ^{asy}	-	-	+		-
NSGA-II _{ss} ^{asy}	-	-	-	-	

Table 5. HV Quality Indicator for the Parallel Approaches

NSGA-II _{gen} ^{syn}	NSGA-II _{gen} ^{asy}	NSGA-II _{ss} ^{asy}	
\bar{x}_{σ_n}	\bar{x}_{σ_n}	\bar{x}_{σ_n}	
8.682e-01 \pm 4.6e-03	8.640e-01 \pm 4.4e-03	8.651e-01 \pm 4.5e-03	+

to apply a statistical test between each pair combination of the algorithms. We have employed for this purpose the MATLAB function Multcompare, that uses as input the output of the above described tests (i.e ANOVA and Kruskal-Wallis) and allows us to carry out different tests for comparing multiples groups. In this studio the selected one was the Tukey’s HSD test. The results are included in Table 6. We observe that we have only statistical confidence about that NSGA-II_{gen}^{syn} outperforms NSGA-II_{gen}^{asy}. Thus, we can conclude that the other versions present similar search capabilities.

Given that our interest is to study the capabilities of the proposed distributed versions of NSGA-II, the fact that NSGA-II_{gen}^{syn} and NSGA-II_{ss}^{asy} produce Pareto fronts of similar quality leads us to conclude that NSGA-II_{ss}^{asy} is the most salient algorithm in our study, because it can use all the processors of our grid system, while NSGA-II_{gen}^{syn} is limited to use at most 100 processors (the population size). To illustrate this point, we give the computing times when executing both algorithms in a representative run. Using 100 processors, NSGA-II_{gen}^{syn} calculates the Pareto front in about 1.83 hours (83.33 minutes), while NSGA-II_{ss}^{asy}, using up to 286 CPUs, needs only 0.125 hours (seven minutes), i.e., a time reduction of 11.9. A typical run of the sequential versions of NSGA-II takes roughly 32 hours of computing time in a single machine, so the time reduction using NSGA-II_{ss}^{asy} is about 109.7. These figures clearly state the advantages of using our proposed parallel schemes when solving real world optimization problems.

8. Conclusions and Future Work

In this work we have presented a study of the parallelization of the NSGA-II using the master-slave model. In concrete, we have designed three parallel versions, which are derived by considering the gen-

erational/steady state and synchronous/asynchronous features that can be applied to the parallel schemes we have applied. The three variants are called NSGA-II_{gen}^{syn}, NSGA-II_{gen}^{asy}, and NSGA-II_{ss}^{asy}, from synchronous, asynchronous generational, and asynchronous steady state, respectively.

A previous experiment designed to study the influence of using a steady state approach in NSGA-II has revealed that this algorithm clearly outperforms the classical NSGA-II in the considered benchmark. The drawback is the high computational time required.

To evaluate the search capabilities of the three parallel algorithms, we have used them to solve a real world problem, the optimization of a broadcasting protocol in MANETs, on a grid composed of up to 330 processors. A first experiment, using the sequential algorithms, shows that the sequential steady state NSGA-II algorithm outperforms the original NSGA-II. The comparison of the three parallel versions indicates that NSGA-II_{gen}^{syn} is the algorithm producing the more accurate fronts, although the differences with the solutions given by NSGA-II_{ss}^{asy} are not significant; however, the latter version needs 7 minutes while the former requires 83 minutes to solve the problem. These times are remarkably low taking into account that the sequential versions of NSGA-II require more than 30 hours of computing time to solve the problem in a single computer.

As future work, we plan to apply the algorithms described in this paper to solve other problems. The study and analysis of the search capabilities of the proposed master-slave versions of NSGA-II are also a matter of further research.

9 Acknowledgement

This work has been partially funded by the Spanish Ministry of Education and Science and by European FEDER under contract TIN2005-08818-C04-01 (the OPLINK project). Juan J. Durillo is supported by grant AP-2006-03349 from the Spanish government.

References

- [1] E. Alba, B. Dorronsoro, F. Luna, A. J. Nebro, P. Bouvry, and L. Hogie. A cellular multi-objective genetic algorithm for optimal broadcasting strategy in metropolitan MANETs. *Computer Communications*, 30(4):685 – 697, 2007.
- [2] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, and R. Quilici. *Grid Computing: Software Environments and Tools*, chapter Programming, Deploying, Composing, for the Grid. Springer-Verlag, January 2006.
- [3] I. V. Bazarov and C. K. Sinclair. Multivariate optimization of a high brightness dc gun photoinjector. *Phys. Rev. ST Accel. Beams*, 8:034202, 2005.
- [4] I. V. Bazarov, C. K. Sinclair, and I. Senderovich. Use of multiobjective evolutionary algorithms in high brightness electron source design. Particle Accelerator Conference (PAC 05). 2005.
- [5] J. Branke, H. Schmeck, K. Deb, and M. Reddy. Parallelizing Multi-Objective Evolutionary Algorithms: Cone Separation. In *2004 Congress on Evolutionary Computation (CEC'2004)*, pages 1952–1957, June 2004.
- [6] S. Choi, J. J. Alonso, and H. S. Chung. Design of a low-boom supersonic business jet using evolutionary algorithms and an adaptive unstructured mesh method.
- [7] C. Coello, D. Van Veldhuizen, and G. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems. 2nd Edition*. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2007.
- [8] P. I. Cowling, N. Colledge, K. P. Dahal, and S. Remde. The trade off between diversity and quality for multi-objective workforce scheduling. In *EvoCOP*, pages 13–24, 2006.
- [9] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.
- [10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
- [11] K. Deb, P. Zope, and A. Jain. Distributed Computing of Pareto-Optimal Solutions with Evolutionary Algorithms. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 534–549.
- [12] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.
- [13] J. J. Durillo, A. J. Nebro, F. Luna, and E. Alba. jMetal: A java framework for developing multiobjective optimization metaheuristics. 2006.
- [14] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.
- [15] F. W. Glover and G. A. Kochenberger. *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*. Springer, January 2003.
- [16] T. Hiroyasu, K. Yoshii, and M. Miki. Discussion of parallel model of multi-objective genetic algorithms on heterogeneous computational resources. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 904–904, 2007.
- [17] L. Hogie, F. Guinand, and P. Bouvry. The metropolitan adhoc network simulator. Technical report.
- [18] S. Kukkonen and K. Deb. Improved pruning of non-dominated solutions based on crowding distance for bi-objective optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation 2006 (CEC 2006)*, pages 1179–1186, 2006.
- [19] J. Linderoth, S. Kulkarni, J. Goux, and M. Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 43–50, 2000.
- [20] A. J. Nebro, G. Luque, F. Luna, and E. Alba. Dna fragment assembly using a grid-based genetic algorithmstar, open. *Computers & Operations Research* (in press), 2007.
- [21] F. Streichert, H. Ulmer, and A. Zell. Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In C. A. Coello Coello, A. H. Aguirre, and E. Zitzler, editors, *Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 3410 of *LNCS*, pages 92–107, Guanajuato, Mexico, 9-11 March 2005.
- [22] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.
- [23] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Dept. Elec. Comput. Eng., Graduate School of Eng., Air Force Inst. Technol., Wright-Patterson, AFB, OH, 1998.
- [24] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.
- [25] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.