

# OBSERVATIONS IN USING GRID TECHNOLOGIES FOR MULTI-OBJECTIVE OPTIMIZATION

A.J. NEBRO , E. ALBA\*, AND F. LUNA

University of Málaga, E.T.S. Ingeniería Informática, 29071 – Málaga (Spain)  
{antonio,eat,flv}@lcc.uma.es

**Abstract.** Solving optimization problems that involve two or more objective functions is of great interest in disciplines such as engineering and economics. Since real-world multi-objective optimization problems (MOPs) usually are very complex, grid computing appears as a powerful technology to solve these kinds of problems. However, few attention has been paid to this issue in the multi-objective optimization research community until now. In this chapter, we study the use of grid computing technology in deterministic and heuristic techniques for solving MOPs. We have analyzed the utilization of Condor and Globus with the goal of gaining experience with these grid technologies, which will guide us to consider the development of more complex algorithms in the future. The obtained results indicate that solving complex multi-objective optimization problems in grids is a very promising research line.

**Key words.** Grid computing, Globus, Condor, Multi-objective optimization, Evolutionary algorithms

**AMS subject classifications.** 90B50, 68M14, 68W10, 68W15, 68W20

**1. Introduction.** Most optimization problems in the industry involve the simultaneous satisfaction of more than one objective function, and these functions are usually in conflict with each other. In this context, parallelism and also grid computing [6, 15] can be considered as powerful tools for solving multi-objective optimization problems (MOPs). However, few works have been devoted to this issue until now [32]. In this chapter, we analyze the utilization of grid computing technologies with the goal of obtaining experiences for solving this kind of problems in computational grids.

In multi-objective optimization, the term “optimize” means finding a solution which would contain the values of all the conflicting objective functions acceptable to the designer. Compared to mono-objective optimization, multi-objective optimization does not restrict to find a unique single solution, but a set of solutions called *non-dominated solutions*. Each solution in this set is said to be a *Pareto optimum*, and when they are plotted in the objective space they are collectively known as the *Pareto front*. Obtaining the Pareto front of a given problem is the main goal of multi-objective optimization.

There are several techniques that can be used to obtain the Pareto front of a multi-objective optimization problem. They can be classified into three categories: enumerative, deterministic, and stochastic [8]. Among them, stochastic methods are the most popular; in particular, evolutionary algorithms (EAs) have been investigated by many authors, and some of the most well-known algorithms for solving MOPs, such as NSGA-II [11], PAES [20], MOGA [12], microGA [7], and SPEA2 [34], belong to this kind of techniques. These methods do not guarantee to obtain the optimal solution, but they provide good solutions to a wide range of optimization problems which other deterministic methods find difficult. On the contrary, deterministic techniques ensure to find optimal solutions. Some algorithms of this class, such as branch and bound methods, are also very popular, but they have the inconvenient of requiring deep knowledge of the problem being solved and they may not be able to solve difficult

---

\*CORRESPONDING AUTHOR

problems by time and/or resource constraints. Finally, we could rely on enumerative search for solving MOPs, which is an exhaustive search not taking advantage of any heuristics, and it can be easily applied to a wide range of problems. Enumerative search is a conceptually simple search strategy based on evaluating each possible solution from a given finite search space. This is an attractive technique because the obtained results are optimum points in the Pareto front, so that they can be used later by the multi-objective optimization research community to be compared against those obtained by using stochastic methods. In fact, if no hypothesis either on the objective functions or the restrictions are made, it is the only known method to obtain the Pareto front. The drawback of this technique is that it is inherently inefficient and it can be computationally expensive and even prohibitive as the search space becomes larger.

The last generation of distributed systems, based on the popularity of the Internet and the availability of a large amount of geographically disperse computational resources, has led to the emergence grid computing [6, 15]. Our interest here is to investigate the applicability of grid technologies to enumerative search and EAs on MOPs. Both techniques can take advantage of grids because they can use the computing power offered by such platforms to tackle complex problems which involve intensive computing tasks. However, parallelism is not only a tool for solving more complex problems, but the resulting parallel EAs (and heuristic techniques in general) usually lead to better results than the sequential ones, even when the former is executed in a single processor computer [1].

We have chosen Globus [14] and Condor [30] for our study. Globus is a common sense choice since it is a standard *de facto* in grid computing, highly sophisticated, and also complex to deal with. Compared to Globus, Condor is attractive because it is easy to install, administrate, and use. Furthermore, they can be combined to obtain the benefits of both systems. This way, we have used Condor to develop a distributed enumerative search algorithm, and a grid-based evolutionary algorithm has been developed for Globus (gPAES). Two are the main contributions of this work, i.e., two new grid approaches to solve multi-objective optimization problems. It is difficult to find works where Globus is directly used to deal with MOPs; even more rare is to find them in the Condor world. And, in this work, we are using two grid technologies, Globus and Condor, to solve fully unrestricted multi-objective optimization.

This chapter is organized as follows. In the next section, we present a brief theoretical background of multi-objective optimization as well as some related works about solving MOPs with grid technologies. Section 3 is devoted to the Condor-based enumerative search. The gPAES heuristic algorithm is described in Section 4. Finally, in Section 5, we outline the conclusions and the future research lines.

**2. Background and Related Work.** In this section, we start explaining some basic multi-objective optimization concepts. Then, we review some related works about using grid computing systems for solving MOPs.

Let us begin with the multi-objective optimization theoretical background. A general multi-objective optimization problem (MOP) can be formally defined as follows:

**DEFINITION 2.1 (MOP).** *Find a vector  $\vec{x}^* = [x_1^*, x_2^*, \dots, x_n^*]$  which satisfies the  $m$  inequality constraints  $g_i(\vec{x}) \geq 0, i = 1, 2, \dots, m$ , the  $p$  equality constraints  $h_i(\vec{x}) = 0, i = 1, 2, \dots, p$ , and minimizes the vector function  $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \dots, f_k(\vec{x})]^T$ , where  $\vec{x} = [x_1, x_2, \dots, x_n]^T$  is the vector of decision variables.  $\square$*

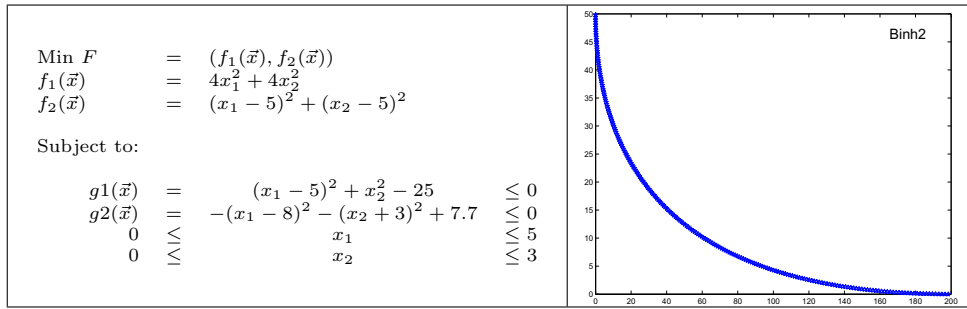


FIGURE 1. Formulation and Pareto front of the problem Bihn2

The set of all values satisfying the constraints define the *feasible region*  $\Omega$  and any point  $\vec{x} \in \Omega$  is a *feasible solution*. As mentioned before, we seek for the *Pareto optimum*. The formal definition of it is provided next:

**DEFINITION 2.2** (Pareto Optimality). *A point  $\vec{x}^* \in \Omega$  is Pareto Optimal if for every  $\vec{x} \in \Omega$  and  $I = \{1, 2, \dots, k\}$  either,  $\forall_{i \in I} (f_i(\vec{x}) = f_i(\vec{x}^*))$  or, there is at least one  $i \in I$  such that  $f_i(\vec{x}) > f_i(\vec{x}^*)$ .*  $\square$

This definition says that  $\vec{x}^*$  is Pareto optimal if there exists no feasible vector  $\vec{x}$  which would decrease some criterion without causing a simultaneous increase in at least one other criterion. Other important definitions associated with Pareto optimality are the following:

**DEFINITION 2.3** (Pareto Dominance). *A vector  $\vec{u} = (u_1, \dots, u_k)$  is said to dominate  $\vec{v} = (v_1, \dots, v_k)$  (denoted by  $\vec{u} \preceq \vec{v}$ ) if and only if  $u$  is partially less than  $v$ , i.e.,  $\forall i \in \{1, \dots, k\}, u_i \leq v_i \wedge \exists i \in \{1, \dots, k\} : u_i < v_i$ .*  $\square$

**DEFINITION 2.4** (Pareto Optimal Set). *For a given MOP  $\vec{f}(\vec{x})$ , the Pareto optimal set ( $\mathcal{P}^*$ ) is defined as  $\mathcal{P}^* = \{\vec{x} \in \Omega | \neg \exists \vec{x}' \in \Omega, \vec{f}(\vec{x}') \preceq \vec{f}(\vec{x})\}$ .*  $\square$

**DEFINITION 2.5** (Pareto Front). *For a given MOP  $\vec{f}(\vec{x})$  and its Pareto optimal set  $\mathcal{P}^*$ , the Pareto front ( $\mathcal{PF}^*$ ) is defined as  $\mathcal{PF}^* = \{\vec{f}(\vec{x}), \vec{x} \in \Omega\}$ .*  $\square$

That is, the Pareto front is composed of the values in the objective space of the Pareto optimal set. As an example, in Figure 1 we show the formulation and the Pareto front of a constrained MOP called Bihn2 [8]. It is a two-objective problem with a single convex Pareto front.

We now turn to review the related work about grid computing and multi-objective optimization. Parallel computers have been widely used in the field of mono-objective optimization [18, 24]. In the case of exact techniques, a typical example is the solution of optimization problems by means of parallel branch and bound algorithms [17]. The idea is, in general, to solve the problems more rapidly or to solve more complex problems. In the context of heuristic methods, parallelism is not only a way for solving problems more rapidly, but for developing more efficient models of search: a parallel heuristic algorithm can be more effective than a sequential one, even when executed on a single processor. For example, see [1, 2] for surveys concerning this effect in parallel evolutionary algorithms.

The idea of using the amount of computing power offered by grid computing systems is very attractive because researchers can solve optimization problems that were considered as intractable until now. For example, in [4], an instance of the quadratic assignment problem (QAP) that would have required about seven years of computation on a single workstation was solved in a week using a Condor-based computational grid of over 2500 processors. This problem was solved by using a dis-

tributed branch and bound algorithm. In [16], grid computing is used to provide a large computing power to a successive convex relaxation method for quadratic optimization problems. The authors use the Ninf system to parallelize the algorithms in several cluster systems connected via LAN and/or the Internet. A combination of parallel genetic algorithms and a local search methodology for the Steiner Problem in Networks is presented in [23]. The authors have considered large dimensions in the sample networks so that they have adopted an efficient grid parallel implementation based on Globus and MPICH-G2. Some other references related to optimization and grid computing are [19, 28, 29].

As to the existing literature, previous to our work, a parallel enumerative search algorithm for multi-objective optimization is described in [31], but it is implemented with MPI and is intended to be used in clusters and computers such as the IBM SP-2. Our work is a further step in the sense of using grid technologies in order to solve MOPs. Nebro *et al.* have also studied the implementation of a distributed enumerative algorithm using grid technologies based on Condor in [26]. In fact, the last is related to the work presented below in this chapter (Section 3). Concerning heuristic methods for multi-objective optimization and grid computing, an implementation of the micro-GA [7] using a grid system to distribute the function evaluations is presented in [3, 9]. In the present work we do not just distribute the function evaluations (e.g., the algorithm remains as in sequential) but we develop a new grid-based model of search based on the PAES [20] algorithm.

**3. Enumerative Search with Condor.** In this section we first show the enumerative algorithm we have used (Section 3.1). Then, in Section 3.2, we detail the Condor-based implementation of this algorithm, and, finally, Section 3.3 is devoted to the presentation of the experiments performed and the analysis of the results.

**3.1. Sequential Enumerative Search.** The enumerative algorithm we use here is similar to the approach for finding non-dominated sets described in [10] (pp. 36-38). The decision variables, assumed continuous, are discretized with a certain granularity (we here use the term granularity to indicate the discretizing degree of each decision variable; thus, a value of 500 implies that each range of values is divided into 500 partitions), and for each combination of values of the variables, the objective functions are evaluated; the resulting vectors are compared among them by using a Pareto dominance test, and the set of non-dominated solutions are obtained. Obviously, the finer the granularity the better the precision of the results, and the larger the computational effort. Thus, granularity and effort are tradeoff factors. If constraints are to be considered, then a feasibility test is performed just before the evaluation and the dominance check. An outline of the algorithm is shown in Figure 2.

**3.2. Condor-Based Enumerative Algorithm.** The parallel algorithm we consider is based on the execution of several processes in parallel running the sequential enumerative algorithm (Figure 2), each one exploring a different region of the search space. We name this program `ParetoGenerate`. When each process finishes, it writes into two files its results. Once all these processes have finished, a new process, called `ParetoMerge`, reads these files and merges their contents applying the dominance test, thus obtaining the Pareto front. This parallel algorithm is simple, because inter-process communication is not necessary, and there is only a synchronization point at the end.

To implement our distributed enumerative search program, we have used Condor [30]. Compared to other grid computing software, it is easy to install and to

```

1  F[M] = {F1, F2, ..., FM} // Objective functions
2  R[C] = {R1, R2, ..., RC} // Constraints
3  x[N] = {x1, x2, ..., xN} // Decision variables
4  f[M] = {f1, f2, ..., fM} // Function values
5  P =  $\emptyset$  // Set of non-dominated solutions
6
7  fix the granularity G of the decision variables
8  for each vector x[i]
9    if x[i] satisfies the constraints R[C]
10   f[j] = evaluation of x[i] by F[M]
11   compare f[j] with members of P for dominance
12   if f[j] is a non-dominated solution
13     add f[j] to P
14   remove the solutions dominated by f[j] from P

```

FIGURE 2. Pseudo-code of the sequential enumerative algorithm

administrate, and existing programs do not need to be modified or re-compiled to be executed under Condor (they must be re-linked with the Condor library). Condor is designed to manage distributed collections (pools) of processors spread among a campus or other organizations [30], having each machine an owner. A feature of Condor is that the owner of each machine can specify the conditions under which jobs are allowed to run; by default, a Condor job stops when a workstation's owner begins using the computer. Thus, Condor jobs use processor cycles that otherwise would be wasted. This way, users are not reluctant to donate their machines, and thus large Condor pools can be built (voluntary computing).

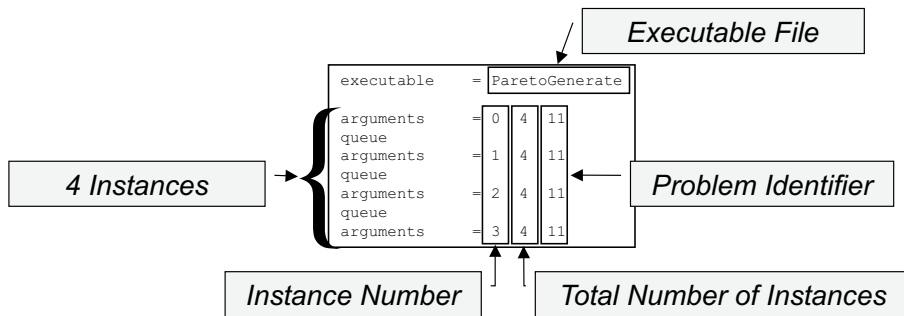


FIGURE 3. Example of Condor submit file

Using Condor to develop a distributed version of our enumerative program has been an easy task. We only need to obtain an executable version of the programs for each different kind of platform (i.e., operating system) in the Condor pool, and we have to write a configuration file. This file contains, among other information, the name of the executable program, the number of instances of the program to be created, and the parameters to be passed to each instance. An example of such a file for executing four instances of `ParetoGenerate` is presented in Figure 3. Here, the search space of the problem named Golinski (whose problem identifier is 11) is divided into four regions, each one explored by an instance. Next, we submit the jobs specified in the configuration file to Condor via `condor_submit` command line tool. Finally, the Condor system executes the tasks on the available machines in the pool. When all the tasks have finished, we collect the information in the result files to obtain the Pareto front.

**3.3. Experiments.** We now show the MOPs we use in our experiments as well as the grid platform used. Finally, an analysis of the results is performed.

**3.3.1. Benchmark.** We have selected four multi-objective problems from the specialized literature: Fonseca [13], Kursawe [22], Osyczka2 [27], and Golinski's speed reducer problem [21]. Their definition appears in Table 1.

TABLE 1. Formulation of the multi-objective problems used

Problem	Definition	Constraints
<b>Fonseca</b>	$\text{Min } F = (f_1(\vec{x}), f_2(\vec{x}))$ $f_1(\vec{x}) = 1 - e^{-\sum_{i=1}^n (x_i - \frac{1}{\sqrt{n}})^2}$ $f_2(\vec{x}) = 1 - e^{-\sum_{i=1}^n (x_i + \frac{1}{\sqrt{n}})^2}$	$-4 \leq x_i \leq 4$ $i = 1, 2, 3$
<b>Kursawe</b>	$\text{Min } F = (f_1(\vec{x}), f_2(\vec{x}))$ $f_1(\vec{x}) = \sum_{i=1}^{n-1} (-10e^{(-0.2 * \sqrt{x_i^2 + x_{i+1}^2})})$ $f_2(\vec{x}) = \sum_{i=1}^n ( x_i ^a + 5 \sin(x_i)^b)$	$-5 \leq x_i \leq 5$ $i = 1, 2, 3$ $a = 0.8$ $b = 3$
<b>Osyczka2</b>	$\text{Min } F = (f_1(\vec{x}), f_2(\vec{x}))$ $f_1(\vec{x}) = -(25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2(x_4 - 4)^2 + (x_5 - 1)^2)$ $f_2(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$	$0 \leq x_1 + x_2 - 2 \leq 10$ $0 \leq 6 - x_1 - x_2 \leq 5$ $0 \leq 2 - x_2 + x_1 \leq 5$ $0 \leq 2 - x_1 + 3x_2 \leq 5$ $0 \leq 4 - (x_3 - 3)^2 - x_4 \leq 6$ $0 \leq (x_5 - 3)^3 + x_6 - 4 \leq 6$ $0 \leq x_1, x_2, x_6 \leq 10$ $1 \leq x_3, x_5 \leq 5$ $0 \leq x_4 \leq 6$
<b>Golinski</b>	$\text{Min } F = (f_1(\vec{x}), f_2(\vec{x}))$ $f_1(\vec{x}) = 0.7854x_1x_2^2 \times (10x_3^2/3 + 14.933x_3 - 43.0934) - 1.508x_1(x_6^2 + x_7^2) + 7.477(x_6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2)$ $f_2(\vec{x}) = \frac{\sqrt{(745.0x_4/x_2x_3)^2 + 1.69e^7}}{0.1x_6^3}$	$\frac{1.0}{x_1x_2^2x_3} - \frac{1.0}{27.0} \leq 0$ $\frac{1.0}{x_1x_2^2x_3} - \frac{1.0}{27.0} \leq 0$ $\frac{x_4}{x_2x_3^2x_6} - \frac{1.0}{1.93} \leq 0$ $\frac{x_5}{x_2x_3x_7} - \frac{1.0}{1.93} \leq 0$ $x_2x_3 - 40 \leq 0$ $x_1/x_2 - 12 \leq 0$ $5 - x_1/x_2 \leq 0$ $1.9 - x_4 + 1.5x_6 \leq 0$ $1.9 - x_5 + 1.1x_7 \leq 0$ $f_2(x) \leq 1300$ $\sqrt{\frac{745.0x_5^2}{x_2x_3} + 1.575e^8} \leq 1100$ $0.1x_6^3 \leq 3.6$ $x_1 \leq 3.6$ $x_2 \leq 0.8$ $x_3 \leq 28.0$ $x_4 \leq 8.3$ $x_5 \leq 8.3$ $x_6 \leq 3.9$ $x_7 \leq 5.5$

**3.3.2. Grid System.** Our Condor pool is composed of desktop PCs, workstations, and servers of several laboratories of the Department of Computer Science at the University of Málaga. We have used the following machines:

- A cluster of four Digital AlphaServer 4100, each one with four Alpha processors at 300 MHz and 256 MB of RAM. The operating system of these machines is Digital Unix 4.0D.
- A cluster of 22 Sun Ultra 1 workstations. The machines have a UltraSPARC II processor at 400 MHz, 256 MB of RAM, and they execute Solaris 2.8.
- Two Sun Ultra Enterprise 450 servers, with four UltraSPARC II at 450 MHz processors and 4 GB of main memory. They run Solaris 2.8.
- A cluster of 16 PCs, each one with an Intel Pentium 4 processor at 2.4 GHz and 512 MB of RAM. They run Suse Linux 8.1 (Kernel 2.4.20).

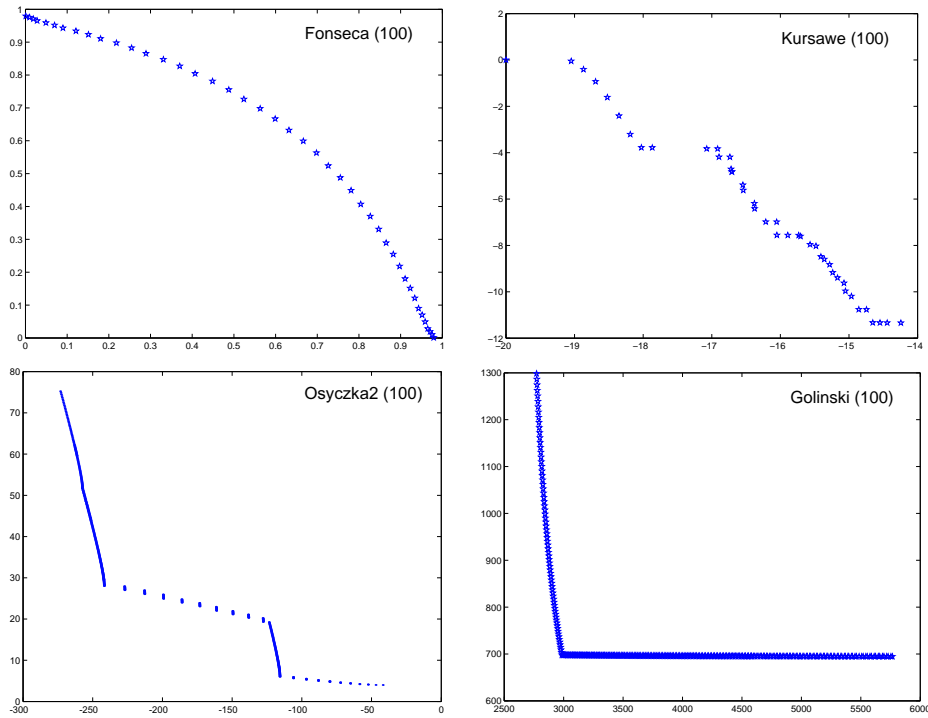


FIGURE 4. Pareto fronts of the considered MOPs  
(the values between parentheses indicate the granularity)

- A cluster of 22 PCs, each one with a AMD Athlon XP 1.2 GHz processor and 256 MB of RAM. They run Debian Linux 3.0 (Kernel 2.2).
- A cluster of 20 PCs, each one with an Intel Pentium III at 600 MHz and 128 MB of RAM. The operating system is Suse Linux 8.1 (Kernel 2.4.20).
- 6 desktop Intel-based PCs running several versions of Linux.

In total, we have been able to use up to 110 processors: 16 Alpha, 30 UltraSparc, 16 Pentium 4, 26 Pentium III, and 22 Athlon.

**3.3.3. Results.** We have solved the four benchmarking problems by dividing the decision variables into 100 partitions each one. Their exact Pareto fronts, which would serve as a reference for heuristically computed fronts (as done in Section 4), are shown in Figure 4. The optimum Pareto fronts are useful by themselves for the community and in this chapter, in order to evaluate a heuristic technique in Section 4. The negative counterpart of the enumeration is its computational complexity, that can be only addressed with a grid to reduce the waiting time down to affordable values. We want to remark that the problems Fonseca and Kursawe are shown as an illustration of the enumerative technique, not for solving them on the grid, since their Pareto fronts can be obtained in a few seconds using a modern Pentium 4 machine.

Then, let us comment first the results of the problem Oszczyka2. The problem was solved in less than two days (wall-clock time), while the total CPU time needed for such task has been reported by Condor of being 90.25 days. Given that the pool is composed of machines of different computing power, the time that would require the sequential program with the fastest processor will be below that amount of time,

but in any case it would be around several tens of days. The problem Golinski was solved in 27 days, and the total CPU time reported by Condor was 775.4 days.

These results show the benefits of using grid computing technologies to solve problems that are infeasible to be computed in a monoprocessor system. We have used a pool of machines of a small size, but Condor also allows to combine several pools of different organizations. Thus, building a grid computer with thousands of processors is nowadays possible. However, even when using such a system, higher accuracy of problems such as Osyczka2 and Golinski with a higher number of partitions per variable (e.g., 1000) will remain intractable, so heuristic techniques are mandatory in these cases.

**4. Evolutionary Algorithms and Globus.** We now turn to apply grid technologies, based on Globus, to run a distributed evolutionary algorithm. Furthermore, we will make use of the Pareto fronts obtained by the enumerative algorithm in the previous section to evaluate the quality of the solutions produced by this evolutionary algorithm. As commented in the introduction, this is the main use that justifies the interest in developing a grid-enabled enumerative search algorithm, as we have already shown in the previous section of this chapter and in [26].

Globus is constructed as a layered architecture which provides three elements necessary for computing in a grid environment [14]:

- *Resource Management.* It provides support for resource allocation, job submission, and managing job status and progress. Its primary components are the Grid Resource Allocation Manager (GRAM) and the Global Access to Secondary Storage (GASS). RSL (Resource Specification Language) is the language used to submit jobs. An example of an RSL specification appears in Figure 7. It specifies, among others, the command line arguments for the executable file and its current working directory (we give a detailed description of this RSL specification in Section 4.2).
- *Information Services.* It provides support for collecting information in the grid and for querying this information. They are collectively known as the Monitoring and Discovery Service (MDS).
- *Data Management.* It provides support for transferring files among machines in the grid and for the management of these transfers.

All these services are built on top of the underlying Grid Security Infrastructure (GSI) that provides elements for secure authentication and communication.

Our grid-enabled evolutionary algorithm, named gPAES, is based in the Pareto Archived Evolution Strategy (PAES) algorithm [20]. PAES is a well-known multi-objective algorithm against which new proposals are compared. The PAES version used here is a  $(1 + 1)$  evolution strategy employing local search and a reference archive of previously found solutions in order to identify the approximate dominance ranking of the current and candidate solutions vectors.

**4.1.  $(1 + 1)$ -PAES.** We have implemented a C++ version of the  $(1 + 1)$ -PAES algorithm based on the description presented in [20]. The  $(1 + 1)$ -PAES represents the simplest nontrivial approach to a multi-objective local search procedure. An outline of this algorithm is included in Figure 5. It maintains a single solution that, at each iteration, is mutated (line 2 in Figure 5) according to normal ES mutation operations [5] to generate a new candidate solution. After that, the algorithm determines whether to accept or reject the mutant solution and whether to archive or not it in a list of non-dominated solutions (archive) by means of an acceptance criterion (line 10 in Figure 5). Since the aim of the multi-objective search is to find a

```

1 generate initial random solution  $c$  and add it to the archive
2 mutate  $c$  to produce  $m$  and evaluate  $m$ 
3 if ( $c$  dominates  $m$ )
4   discard  $m$ 
5 else if ( $m$  dominates  $c$ )
6   replace  $c$  with  $m$ , and add  $m$  to the archive
7 else if ( $m$  is dominated by any member of the archive)
8   discard  $m$ 
9 else
10  apply test( $c, m, \text{archive}$ ) to determine which becomes the new current solution
    and whether to add  $m$  to the archive
11 until a termination criterion has been reached, return to line 2

```

FIGURE 5. Pseudo-code for (1 + 1)-PAES

set of spread non-dominated solutions, PAES uses a crowding procedure based on an adaptive numerical grid that recursively divides up the objective space [20].

**4.2. gPAES.** Heuristic algorithms can take advantage of grid-enabled technologies by using the enormous computational power they offer to carry out a deep exploration of the search space. This is the idea behind gPAES: the model of search consists in remotely executing a number of sequential PAES algorithms (separately exploring the whole search space) on machines of the grid and locating the best solutions according to some defined metrics. In this case, we have used the metric  $M_1^*$  (see Section 4.3.2) to rank the fronts obtained by these PAES algorithms.

```

1 MAX_PAES = 100 // Maximum number of PAES
2 MIN_M1* = MAX_REAL
3
4 user authentication and authorization // security
5 availableMachines = queryMDS() // look for available machines
6 gassServerURL = startGASSServer()
7 numberOfPAESExecuted = 0
8 while (numberOfPAESExecuted < MAX_PAES)
9   rsl = buildRSL(numberOfPAESExecuted, gassServerURL)
10  machine = getAvailableMachine()
11  newThread -> run(rsl, machine)
12  while (any new PAES has finished)
13    PF = retrieveParetoFront()
14    metricValue =  $M_1^*(PF)$ 
15    if (metricValue < MIN_M1*)
16      MIN_M1* = metricValue
17      storeParetoFront(PF)
18      numberOfPAESExecuted ++
19    end if
20  end while
21  waitForAvailableMachines()
22 end while

```

FIGURE 6. Pseudo-code of gPAES

An outline of gPAES is shown in Figure 6: the application firstly checks for user authentication and authorization (needed for Globus execution). Next, it performs a query to the Globus information service (MDS) looking for available machines, and it starts a GASS server for file transfers. On each machine, PAES is executed using an RSL specification as it is depicted in Figure 7. As the computation of each instance finishes, its Pareto front is retrieved, and the metric  $M_1^*$  is calculated for this front; if it has the lowest value for  $M_1^*$  found so far, the front is stored. Finally, a new

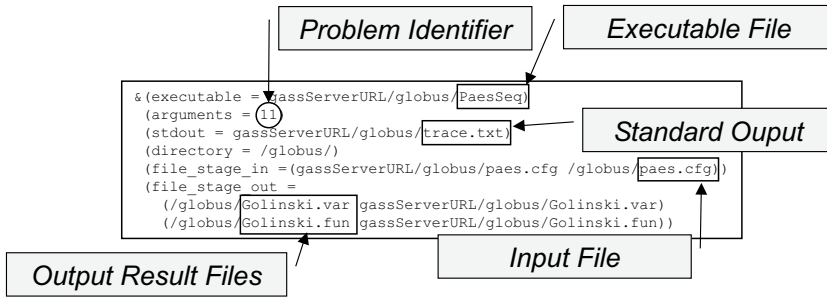


FIGURE 7. Example of an RSL specification

PAES is launched to find a new Pareto optimal set. Note that this independent-runs search model is loosely-coupled (i.e., well suited for large grids) and one of the simplest approaches of using grid technologies to solve multi-objective optimization problems. More elaborated strategies are in progress in an ongoing work.

The RSL string in Figure 7 is composed of several RSL attributes such as: the name of the executable, the command line arguments for the executable, the name of the remote file to store the standard output from the job, the path of the directory for the requested job, a first list of filename pairs that indicate the files to be transferred from the local host to the execution host, and a second list of filename pairs showing the files to be transferred from the execution host to the local host. The RSL example in Figure 7 specifically executes the PAES algorithm for the problem Golinski (whose problem identifier is 11); the standard output is stored in a file called `trace.txt`, the configuration file for PAES (`paes.cfg`) is stored in the local directory `/globus/` and it is copied, with the same name into the same directory in the remote machine. Finally, the output result files are retrieved from the remote directory `/globus/` to the same local directory and also with the same name. The `gassServerURL` is used to enable the GASS file transfers, so the executable file, the `stdout` file, the configuration file, and the output results files are transferred automatically by the GASS server started previously.

**4.3. Experimentation.** We next present the Globus platform we have built. Then, we show the performance metrics used and analyze the obtained results.

**4.3.1. Grid System.** We have installed Globus Toolkit 2.2.4 in each machine of a cluster of 16 PCs, each one with a Pentium 4 processor at 2.4 GHz, 512 MB of RAM, and running SuSE Linux 8.1. The interconnection network is a Fast-Ethernet at 100 Mbps. The programs have been compiled with GCC v.2.95 using the option `-O3`.

**4.3.2. Metrics.** Several metrics have been proposed for guiding the search (online use) and for measuring the results (offline use) of Pareto-based multi-objective optimization algorithms. In this work we use the metrics  $M_1^*$  [33] and  $\Delta$  [11]. The former gives the average distance to the Pareto optimal set and the latter is a diversity metric that measures the extent of spread achieved among the obtained solutions. Their formulations are:

$$(4.1) \quad M_1^* = \frac{1}{|Y'|} \sum_{d' \in Y'} \min\{\|d' - \bar{d}\|^*; \bar{d} \in \bar{Y}\}$$

where, if  $Y$  is the set of all possible objective vectors, then  $Y' \subseteq Y$  is the set of

objective vectors found, and  $\bar{Y} \subseteq Y$  is the Pareto optimal set. Ideally, this metric should be zero meaning that the front is exactly the real Pareto front of the MOP. The  $\Delta$  metric is defined as:

$$(4.2) \quad \Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}}$$

where  $d_i$  is the Euclidean distance between consecutive solutions,  $\bar{d}$  is the mean of these distances, and  $d_f$  and  $d_l$  are the Euclidean distances to the *extreme* (bounding) solutions of the exact Pareto front in the objective space (see [11] for the details). The optimal value for this metric is also 0, pointing out a perfect spreadout of the solutions in the Pareto front.

Note that these two metrics can only be used because we have the Pareto optimal set computed by the enumerative algorithm (Section 3). However, if this optimal non-dominated set is not available, any other metrics (for example, see [10]) would be valid for guiding the search of the algorithm at run time (online) as well as for measuring its results (offline).

**4.3.3. Results.** The gPAES algorithm needs two configuration parameters: the number of parallel copies of the sequential PAES algorithm and the metrics to rank the fronts obtained by these PAES algorithms in a final front. For the first parameter, we have tested two configurations, named gPAES<sub>10</sub> and gPAES<sub>100</sub>, running 10 and 100 parallel elementary algorithms. For the second parameter, we have used the  $M_1^*$  metric (see line 14 in Figure 6).

TABLE 2. Results of gPAES

Problem	$M_1^*$				$\Delta$			
	PAES	gPAES <sub>10</sub>	gPAES <sub>100</sub>	$\mathcal{A}$	PAES	gPAES <sub>10</sub>	gPAES <sub>100</sub>	$\mathcal{A}$
Fonseca	0.0150	0.0039	0.0015	+	1.0291	1.0401	1.0949	+
Kursawe	0.0972	0.0153	0.0124	+	1.1648	1.0821	1.1026	+
Osyczka2	18.2065	5.3253	1.6898	+	1.2316	1.3115	1.3521	+
Golinski	52.9564	15.1230	8.5270	+	1.1893	1.2355	1.2226	-

We also present an evaluation of our implementation of the sequential PAES algorithm for comparison purposes. It uses the following configuration: binary representation with a precision of 5 digits, bit-flip mutation using a mutation rate of  $1/L$ , where  $L$  is the binary string length, and, at most, 100 non-dominated solutions in the archive. The stopping criterion is to reach 25000 function evaluations. Note that, while the sequential PAES performs 25000 function evaluations, gPAES<sub>10</sub> and gPAES<sub>100</sub> carry out 250000 and 2500000 evaluations, respectively. We have considered the same problems as in the enumerative algorithm: Fonseca, Kursawe, Osyczka2, and Golinski. All the presented results are the average values over 30 independent runs. We also include an ANOVA (Analysis of Variance) test comparing the three algorithms with a 5% of significance (marked as “+” in column  $\mathcal{A}$  of Table 2). An ANOVA [25] tests the means of two or more sets of numeric values.

In order to compare the results (see Table 2) of the sequential PAES and both configurations of gPAES, gPAES<sub>10</sub> and gPAES<sub>100</sub>, we have applied the metrics  $M_1^*$  and  $\Delta$ . We want to remark that we have used the metric  $M_1^*$  in two ways. First, it allows us to rank the fronts inside the gPAES algorithm (online usage), and, second, to compare the final results of the sequential PAES algorithm and the two configurations of gPAES, gPAES<sub>10</sub> and gPAES<sub>100</sub>, like  $\Delta$  (always offline).

Let us now turn to the analysis of the results according to the first considered metric  $M1^*$ . First, it can be observed that  $gPAES_{10}$  outperforms the sequential PAES algorithm. For example, for the problem Kursawe, the Pareto front of  $gPAES_{10}$  is more than six times closer to the exact Pareto front than the one obtained by the sequential PAES (0.0972 against 0.0153). For the other three problems, the fronts obtained by  $gPAES_{10}$  are around 3.5 times closer than the sequential PAES front according to the resulting  $M1^*$  values. Second,  $gPAES_{100}$  also obtains Pareto fronts closer to the optimum Pareto front than both PAES and  $gPAES_{10}$ . Thus, for example, in the problem Osyczka2 the  $M1^*$  metric values for the sequential PAES and  $gPAES_{10}$  are respectively 3.15 and 10.77 larger than  $gPAES_{100}$  values (5.3253 against 1.6898, and 18.2065 against 1.6898, respectively). Note that this behavior holds for the four considered problems and with statistical confidence (see “+” symbols in Table 2 indicating 95% of confidence). This is an expected result since the larger number of function evaluations done by the sequential PAES,  $gPAES_{10}$ , and  $gPAES_{100}$  ( $2.5e4$ ,  $2.5e5$ , and  $2.5e6$ , respectively), induces a deeper exploration of the search space.

In the right part of Table 2 we show the metric  $\Delta$ . We can notice that the  $gPAES$  algorithm improves the diversity of the obtained Pareto front for the problem Kursawe, and there exists statistical confidence for this claim. However, for the problems Fonseca and Osyczka2, the results of this metric indicate that the sequential PAES yields the most widely and uniformly spreadout set of non-dominated solutions. We can explain this behavior because of the  $gPAES$  configuration. It uses the metric  $M1^*$  to select the best Pareto front from the independent PAES runs and it seems that reducing the distance as the single criterion to the real Pareto optimal set is harmful for the diversity of the archive. In fact, if  $gPAES$  considers several and possibly conflicting metrics to arrange the fronts, a tradeoff between them should be achieved (this is again a multi-objective problem).

Experiences with Globus show that we are able to utilize grids in order to heuristically solve MOPs. Our approach,  $gPAES$ , is designed for being well suited for large grids (loosely-coupled independent-runs search model). In order to build these platforms, Globus is the most promising option due to its security issues that allow to put together computational resources from different organizations (universities, research centers, etc.). In addition, Globus is able to integrate pools of machines that are managed by specialized batch system such as Condor or PBS, thus enabling the creation of potential grids composed of hundreds (even thousands) of machines.

**5. Conclusions.** Grid technologies offer a strategic opportunity to develop new algorithms for solving optimization problems. In this context, we have used the Condor system with 110 processors to implement a distributed enumerative search algorithm for solving multi-objective problems. Our experiences indicate that it is possible to solve in a few days problems which would have otherwise required hundreds of days to be solved in a single computer.

We have solved a benchmark composed of both constrained and unconstrained multi-objective problems. We conclude that the multi-objective research community can take advantage of computational grids to solve difficult problems and to obtain the exact Pareto fronts, thus allowing fair and meaningful exhaustive metrics, such as the distance to the Pareto front.

On the other hand, we have presented an approximation of using grid technologies based on Globus to numerically improve an evolutionary algorithm. The algorithm presented, named  $gPAES$ , uses Globus to execute a number of sequential PAES algorithms in parallel. Then, with respect to some given metric, the fronts are ranked

and the best one is presented as a result. Our experiments reveal that, as the number of sequential algorithms is increased, more accurate results are obtained. Hence, this grid-based EA is able to take advantage of the computational power offered by the grid system.

Future research is in the line of using Globus, possibly in combination with Condor, in a grid composed of hundreds of computers. We also intend to apply the experiences obtained in this work to face the parallelization of other heuristic techniques in grid.

**Acknowledgments.** This work has been partially funded by the Ministry of Science and Technology and FEDER under contracts TIC2002-04498-C05-02 (the TRACER project) and TIC2002-04309-C02-02.

#### REFERENCES

- [1] E. Alba and M. Tomassini, *Parallelism and Evolutionary Algorithms*, IEEE Transactions on Evolutionary Computation, **6:5** (2002), pp. 443–462.
- [2] E. Alba and J. Troya, *A Survey of Parallel Distributed Genetic Algorithms*, Complexity, **4:4** (1999), pp. 31–52.
- [3] G. Aloisio, E. Blasi, M. Cafaro, I. Epicoco, S. Fiore, and S. Mocavero, *A Grid Environment for Diesel Engine Chamber Optimization*, in Proc. of ParCo2003, Elsevier (2003).
- [4] K. Anstreicher, N. Brixius, J.-P. Goux, and J. Linderoth, *Solving Large Quadratic Assignment Problems on Computational Grids*, Mathematical Programming, **91** (2002), pp. 563–588.
- [5] T. Bäck, *Evolutionary Algorithms: Theory and Practice*, Oxford University Press, New York, USA (1996).
- [6] F. Berman, G. Fox, and A. Hey, *Grid Computing. Making the Global Infrastructure a Reality*, John Wiley & Sons (2003).
- [7] C. Coello and G. Toscano, *Multiobjective Optimization Using a Micro-Genetic Algorithm*, in GECCO-2001, Morgan Kaufmann (2001), pp. 274–282.
- [8] C. Coello, D. Van Veldhuizen, and G. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer Academic Publishers (2002).
- [9] A. de Risi, T. Donato, D. Laforgia, G. Aloisio, E. Blasi, and S. Mocavero, *An Evolutionary Methodology for the Design of a D.I. Combustion Chamber for Diesel Engines*, in Conf. on Thermo- and Fluid Dynamics Processes in Diesel Engines, (2004).
- [10] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons (2001).
- [11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II*, IEEE Trans. on Evolutionary Computation, **6:2** (2002), pp. 182–197.
- [12] C. Fonseca and P. Fleming, *Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization*, in Proc. of the 5th Int. Conf. on Genetic Algorithms, Morgan Kaufmann (1993), pp. 416–423.
- [13] C. Fonseca and P. Flemming, *Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms - Part II: Application Example*, IEEE Transactions on System, Man, and Cybernetics, **28** (1998), pp. 38–47.
- [14] I. Foster and C. Kesselman, *Globus: a Metacomputing Infrastructure Toolkit*, International Journal of Supercomputer Applications, **11:2** (1997), pp. 115–128.
- [15] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann (1999).
- [16] K. Fujisawa, M. Kojima, and A. T. M. Yamashita, *High Performance Grid and Cluster Computing for Some Optimization Problems*, in Proc. of the 2004 Int. Symp. on Applications and the Internet Workshops (SAINTW'04), IEEE Press (2004), pp. 612–615.
- [17] B. Gendron and T. G. Crainic, *Parallel Branch and Bound Algorithms: Survey and Synthesis*, Operations Research, **42** (1994).
- [18] A. Grama and V. Kumar, *State of the Art in Parallel Search Techniques for Discrete Optimization*, IEEE Transactions on Knowledge and Data Engineering, **11:1** (1999), pp. 28–35.
- [19] H. Imade, R. Morishita, I. Ono, and N. Ono, *A Grid-Oriented Genetic Framework for Bioinformatics*, New Generation Computing, **22:2** (2004), pp. 177–186.
- [20] J. Knowles and D. Corne, *Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy*, Evolutionary Computation, **8:2** (2000), pp. 149–172.

- [21] A. Kurpati, S. Azarm, and J. Wu, *Constraint Handling Improvements for Multi-Objective Genetic Algorithms*, Structural and Multidisciplinary Optimization, **23:3** (2002), pp. 204–213.
- [22] F. Kursawe, *A Variant of Evolution Strategies for Vector Optimization*, in Parallel Problem Solving for Nature, H. Schwefel and R. Männer, Eds., Springer (1990), pp. 193–197.
- [23] G. Lo Presti, G. Lo Re, P. Stornio, and A. Urso, *A Grid Enabled Parallel Hybrid Genetic Algorithm for SPN*, in ICCS 2004, LNCS **3036**, Springer (2004), pp. 156–163.
- [24] A. Migdalas, P. Pardalos, and S. Story, *Parallel Computing in Optimization (Applied Optimization, Vol. 7)*, Kluwer (1997).
- [25] D. C. Montgomery, *Design and Analysis of Experiments*, John Wiley & Sons (1991).
- [26] A. Nebro, F. Luna, and E. Alba, *Multi-Objective Optimization Using Grid Computing*, Soft Computing Journal. To appear, (2005).
- [27] A. Osyczka and S. Kundo, *A New Method to Solve Generalized Multicriteria Optimization Problems Using a Simple Genetic Algorithm*, Structural Optimization, **10** (1995), pp. 94–99.
- [28] M. Parashar, H. Klie, U. Catalyurek, T. Kurc, V. Matossian, J. Saltz, and M. Wheeler, *Application of Grid-Enabled Technologies for Solving Optimization Problems in Data-Driven Reservoir Studies*, in ICCS 2004, LNCS **3036**, Springer (2004), pp. 805–812.
- [29] Y. Tanimura, T. Hiroyasu, M. Miki, and K. Aoi, *The System for Evolutionary Computing on the Computational Grid*, in Parallel and Distributed Computing and Systems (PDCS 2002), IASTED/ACTA Press (2002), pp. 56–65.
- [30] D. Thain, T. Tannenbaum, and M. Livny, *Condor and the Grid*, in Grid Computing: Making the Global Infrastructure a Reality, F. Berman, G. Fox, and T. Hey, Eds., John Wiley & Sons (2002).
- [31] D. Van Veldhuizen and G. Lamont, *Multiobjective Evolutionary Algorithm Test Suites*, in Proc. of the 1999 ACM Symp. on Applied Computing, ACM Press (1999), pp. 351–357.
- [32] D. Van Veldhuizen, J. Zydallis, and G. Lamont, *Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms*, IEEE Trans. on Evolutionary Computation, **8:2** (2003), pp. 144–173.
- [33] E. Zitzler, K. Deb, and L. Thiele, *Comparison of Multiobjective Evolutionary Algorithms: Empirical Results*, Evolutionary Computation, **8:2** (2000), pp. 173–195.
- [34] E. Zitzler, M. Laumanns, and L. Thiele, *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*, Tech. Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), (2001).