Available online at www.sciencedirect.com

ScienceDirect

Parallel Computing 32 (2006) 377–393

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Observations in using Grid-enabled technologies for solving multi-objective optimization problems

F. Luna, A.J. Nebro [*], E. Alba

Universidad de Málaga, Departamento de Lenguajes y Ciencias de la Computación, ETSI Ingeniería Informática, 29071 Málaga, Spain

Received 21 February 2006; accepted 19 June 2006
Available online 8 August 2006

Abstract

In this paper, we analyze some technical issues concerning the use of Grid-enabled technologies based on the Globus Toolkit to solve multi-objective optimization problems. We develop two distributed algorithms: an enumerative search and an evolutionary algorithm. The former is a simple technique, whose high time complexity is mastered down to acceptable execution times to some extent by the use of a Grid-enabled computing system such Globus. The second algorithm is an extension of PAES, a sequential evolutionary algorithm. The parallel results indicate that using Globus is a promising research line to solve multi-objective problems in Grid computing environments.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Grid computing; Globus; Multi-objective optimization; Enumerative search; Evolutionary algorithms

1. Introduction

A multi-objective optimization problem (MOP) can be defined as the problem of finding a vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent a set of objective functions. These functions form a mathematical description of performance criteria which are usually in conflict with each other. Hence, the term "optimize" means finding a solution that hopefully contains values for all the objective functions that are acceptable to the designer [1]. More formally:

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

In other words, one wishes to determine the particular set $\{x_1^*, x_2^*, \ldots, x_n^*\}$ yielding optimum values for all simultaneously considered k objective functions from among the set of all values satisfying the constraints.

[*] Corresponding author. Fax: +34 952 13 13 97.
E-mail addresses: flv@lcc.uma.es (F. Luna), antonio@lcc.uma.es (A.J. Nebro), eat@lcc.uma.es (E. Alba).

0167-8191/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2006.06.004

Generally, multi-objective optimization does not restrict to finding 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. The techniques that can be used to compute a Pareto front can be classified into exact and heuristic ones. In recent years, heuristic methods have been widely studied. In particular, evolutionary algorithms have been investigated by many authors [2,3]. These methods do not guarantee to obtain the optimal solution, but they do provide near optimal solutions to a wide range of optimization problems for which exact methods are impractical.

In contrast to heuristic methods, enumerative search is a conceptually simple search strategy based on evaluating each possible solution in a finite search space, and thus it is able to find optimal solutions. The drawback of this technique resides in its inability to scale as the search space becomes larger. Despite this inconvenience, the results that can be obtained by using enumeration are of great interest to the multi-objective optimization research community, because the resulting Pareto fronts can be used to be compared against those obtained by using heuristic algorithms. In consequence, the quality of the solutions produced by these heuristic algorithms can be measured in a non-subjective way by researchers. In fact, enumeration is the only way of computing the exact Pareto front for an arbitrary multi-objective problem.

In this context, the increasing impact of Grid computing in last years [4,5] makes this kind of systems to appear as a powerful solution for complex tasks demanding high computational resources that cannot be addressed in normal clusters. By using some Grid-enabled technology, enumerative methods can be viable to obtain optimal solutions for scenarios in which this is mandatory. Obviously, this does not mean that Grids are to replace heuristic techniques by enumerative search; when optimization problems exhibit a combinatorial behavior, the computational requirements grow exponentially, so even using a large Grid computing system is impractical. On the other hand, heuristic techniques can take advantage of Grid computing systems to solve complex problems of practical interest.

Much of the Grid technology relies on the Globus Toolkit [6]. Globus has emerged as the *de facto* standard for Grid computing. Globus allows researchers to set up a Grid of computers through different Internet administrative domains, solving the problems of resource management and security. Some examples of its wide are *EU DataGrid* (http://eu-datagrid.web.cern.ch/eu-datagrid/), *TeraGrid* (http://www.teragrid.org), and the *NASA's Information Power Grid* (http://www.ipg.nasa.gov).

In this paper, we first analyze the utilization of the Globus Toolkit in order to parallelize an enumerative search algorithm for solving multi-objective optimization problems. We have built our test Grid in the context of a cluster of computers with the goal of obtaining experiences with this Grid technology, which will guide us to consider the development of more complex algorithms in the future in larger computational Grids. Second, we use Globus to develop a Grid-based evolutionary algorithm; in concrete, we have developed gPAES, a Grid extension of the well-known PAES algorithm [7].

The paper is organized as follows. In Section 2, we discuss related work concerning multi-objective optimization and parallel computing. Section 3 reviews the major components of the Globus Toolkit. It is followed by the description of the enumerative search algorithm for computing exact Pareto fronts in Section 4. Section 5 details the application of Globus to run the new gPAES algorithm. Finally, we outline the conclusions and future research lines in Section 6.

2. Related work

Parallel computers have been widely used in the field of mono-objective optimization [8,9]. In the case of exact techniques, a typical example is the solution of optimization problems by means of parallel branch and bound algorithms [10]. 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 speeding up the computations, but for developing more efficient models of search: a parallel heuristic algorithm can be more effective than a sequential one, even when the former is executed on a single processor. For example, see [11,12] for surveys concerning this effect in parallel evolutionary algorithms.

The advantages that parallel computing offers to mono-objective optimization should also hold in multi-objective optimization. Even though few efforts have been devoted to parallel implementations in this field until recently [3], parallelism and multi-objective optimization is a growing field [13]. Some works concerning parallel exact methods are [14,15]. With regard to parallel heuristic methods, we can find parallel Evolutionary Algorithms [16–23], parallel Differential Evolution algorithms [24], parallel Ant Colonies [25], and parallel Particle Swarm Optimization algorithms [26]. Parallel hybrid approaches have also been considered [27].

In general, most works on parallel computing and optimization involve two kinds of parallel systems: shared-memory multi-processor systems and distributed systems, these last based on clusters. In the first case, operating system support is enough to develop parallel programs: we can use sequential languages with thread libraries or system calls to write multi-threaded or multi-process applications that take advantage of the many processors of the system. Another option is to use parallel languages, such as Java. In the case of distributed systems, there is an overwhelming amount of issues concerning interconnection networks (Ethernet, Myrinet, Infiniband), topologies (bus, ring, mesh, tree, hypercube), programming models (message passing, RPC, distributed shared memory), communication libraries for sequential languages (MPI, PVM, sockets), parallel languages (Java, C#), middleware (CORBA, DCOM, RMI), and even Internet technologies (Web services).

The last generation of distributed systems is based precisely on the popularity of the Internet and the availability of a large amount of geographically disperse computational resources, that can be linked together to provide a single, unified computing resource. This has led to what it is called Grid computing [5]. The idea of using such amount of computing power is very attractive because researchers can solve optimization problems that were considered as intractable until now. Some references related to optimization and Grid computing are [28–31].

Previous to our work, a parallel enumerative search algorithm for multi-objective optimization is described in [15], 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 search algorithms using Grid technologies in [32], although they use a Grid system based on Condor. Concerning heuristic methods for multi-objective optimization and Grid computing, an implementation of the micro-GA [33] using a Grid system to distribute the function evaluations is presented in [34]. 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 algorithm.

3. Globus

The Globus project seeks to enable the construction of computational Grids. In this context, a Grid is a hardware and software infrastructure that provides dependable, consistent, and pervasive access to high-end computational capabilities, despite the geographical distribution of both, resources and users. A central element of Globus is the Globus Toolkit, a community-based, open-architecture, open-source set of services and software libraries that supports Grids and Grid applications.

Computational Grids intend to support a wide variety of applications and programming paradigms. Consequently, rather than providing a uniform programming model, such as the object-oriented model, the Globus Toolkit provides a bag of services that developers of specific tools or applications can use to meet their own particular needs.

Globus is constructed as a layered architecture, as illustrated in Fig. 1, in which high-level global services are built upon essential low-level core local services. It provides three elements necessary for computing in a Grid environment [6]:

- *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 for submitting jobs. An example of an RSL specification appears in Fig. 2. 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).

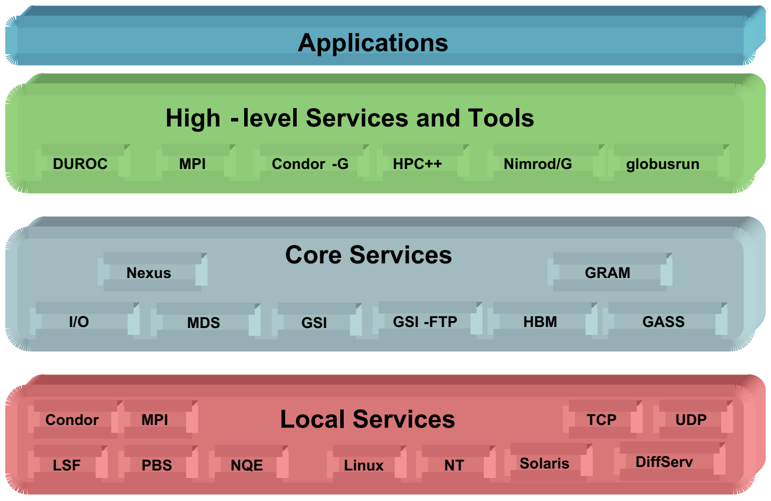

Fig. 1. Globus layered architecture.

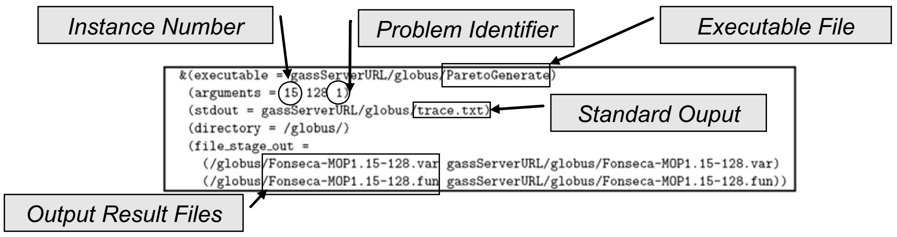
```
&(executable = gassServerURL/globus/ParetoGenerate)
  (arguments = 15 128 1)
  (stdout = gassServerURL/globus/trace.txt)
  (directory = /globus/)
  (file_stage_out =
    (/globus/Fonseca-MOP1.15-128.var gassServerURL/globus/Fonseca-MOP1.15-128.var)
    (/globus/Fonseca-MOP1.15-128.fun gassServerURL/globus/Fonseca-MOP1.15-128.fun))
```

Fig. 2. Example of an RSL specification.

- *Data management*. It provides support for transfer 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.

Globus is the common-sense software to be used in an initial evaluation of research lines like complex optimization in Grid. The initial complexity of installing, managing, and using a Globus Grid in Internet will hopefully be worthwhile because of the milestones in optimization these researchers could probably reach with Grid computing.

4. Enumerative search algorithm

In this section, we describe the sequential enumerative search algorithm we have developed as well as its parallelization using Globus.

4.1. Sequential algorithm

The sequential algorithm we have used is outlined in Fig. 3. The decision variables (see the definition of MOP in Section 1), assumed continuous, are discretized with a certain degree 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 is obtained. Note that the finer the discretization degree the better the precision of the results and the larger the computational effort. Thus, precision and effort are tradeoff factors. If constraints are to be considered, then a feasibility test is performed just before the evaluation and dominance checking.

```
F[M] = {F1, F2, ..., FM} // Objective functions
R[C] = {R1, R2, ..., RC} // Constraints
x[N] = {x1, x2, ..., xN} // Decision variables
f[M] = {f1, f2, ..., fM} // Function values
P = $\emptyset$ // Set of non-dominated solutions

Fix the discretization degree G of the decision variables
For each vector x[i]
  If x[i] satisfies the constraints R[C]
    f[j] = evaluation of x[i] by F[M]
    Compare f[j] with members of P for dominance
    If f[j] is a non-dominated solution
      Add f[j] to P
    Remove the solutions dominated by f[j] from P
```

Fig. 3. Pseudo-code of the sequential enumerative algorithm.

The number of iterations carried out by the algorithm depends on the number of decision variables, N, and the desired precision, which depends on the discretization degree, G; concretely, the number of iterations is G^N. Thus, if a problem has five variables and all of them are discretized dividing their value ranges by 100, then 100^5 iterations have to be carried out. However, the complexity of the algorithm is strongly influenced by the constraint test and the evaluation of the objective functions. Clearly, the time required to compute each iteration is related to the number of objective functions and their complexity. In addition, the evaluation step is only performed if the constraint test is passed, so the more restrictive the constraints the smaller the number of evaluations. Furthermore, the constraint test can also take a significant amount of time. We analyze these issues in greater detail in Section 4.3.3. Following the previous example, if we assume no constraints and the total function evaluation time is 1 ms, the algorithm will require around 10^7 s, i.e., 116 days. This example justifies the need of using a Grid system to address this kind of problems.

4.2. Globus-based distributed enumerative search

The distributed algorithm we have developed is based on the execution of several processes, each of them executing a sequential algorithm that explores a different part of the search space. We name this program `ParetoGenerate`. When each process finishes, it writes in secondary memory the obtained results (the results are stored into two files, one containing the values of the decision variables, and other one containing the values of the objective functions). Once all these processes have finished, a new process, called `ParetoMerge`, reads all the secondary memory files and gathers the results to obtain the Pareto optimal solutions and the Pareto front, by means of the application of dominance tests. The resulting parallel algorithm is simple, because inter-process communication is not necessary, and there is only a synchronization point at the end.

The idea of our parallelization scheme is to divide the search space into a set of disjoint subspaces, each of them explored by an instance of `ParetoGenerate`. Briefly, our application firstly checks for user authentication and authorization. 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, an instance of `ParetoGenerate` is executed using an RSL specification as it is depicted in Fig. 2. When each instance of this program finishes, it writes in secondary memory the results, which are stored into two files in the remote filesystem of the remote machine, one containing the values of the decision variables, and other one containing the values of the objective functions. These two files are transferred to the local filesystem using the GASS server and, if there are more subspaces left, a new instance is created to solve a remaining unexplored part of the search space. For each remote job submitted to Globus, we use a thread that waits for job completion. When the job finishes, its associated thread synchronizes with the application's main thread, which creates an RSL specification for another `ParetoGenerate` instance dynamically. Then, another thread is created in order to wait for this job completion, and so on.

As an example, we show in Fig. 2 a typical RSL string composed of several RSL attributes for such a remote execution. These attributes are 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, and a list of filename pairs that indicate files to be transferred from the execution host to the local host. This RSL example executes specifically the 15th instance of `ParetoGenerate` for the problem named `Fonseca` (which problem identifier is 1), where the search space has been partitioned into 128 disjoint subspaces; the standard output is stored in a file called `trace.txt` and the output result files are retrieved from the remote directory `/globus/` to the same local directory and with the same name. The `gassServerURL` is used to enable the GASS file transfers, so the executable file, the `stdout` file, and the output results files are transferred automatically by the GASS server started previously.

A typical example of the computation in a remote machine appears in Fig. 4, where we show the CPU usage of a node and the network traffic it produces. Note that a transmission of information through the network exists for each task computed by the node. As it can be seen in the figure, the last instance of `ParetoGenerate` computed is the most time consuming one. This points out that the execution time of the different tasks is not uniform (we will analyze this fact later, in Section 4.3.3).

Designing, developing and deploying such parallel algorithm using a Grid system like Globus requires tackling some major Grid computing issues such as security, heterogeneity, dynamic availability, communication, and load balancing. They all are discussed in the following sections.

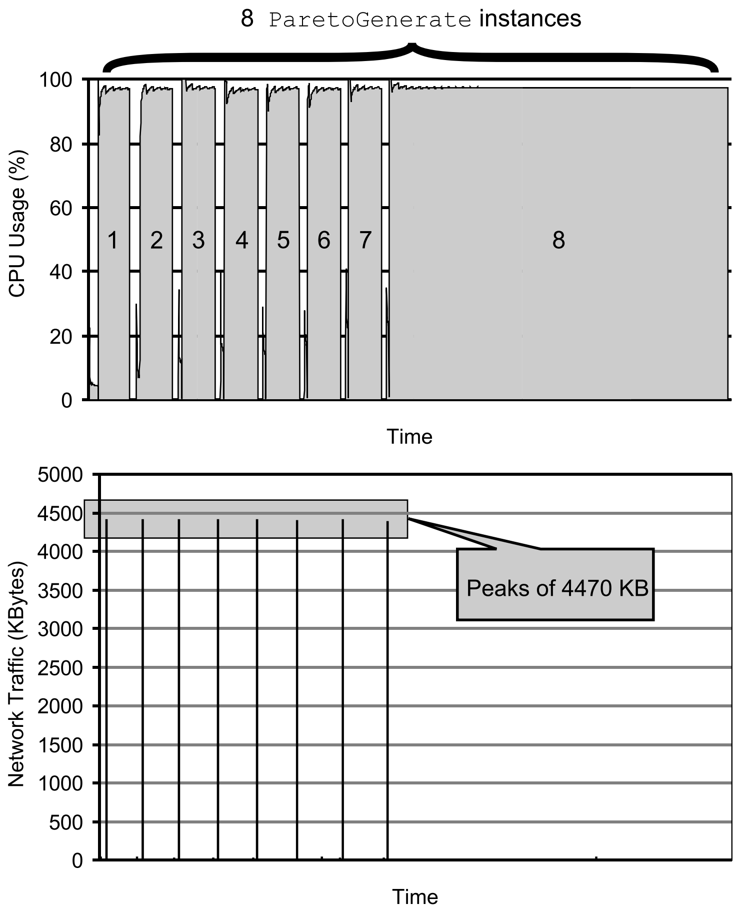

Fig. 4. Example of CPU usage and network traffic during a typical Grid computation.

4.2.1. Security

As stated above, our algorithm firstly checks whether the user credentials are valid or not using the Globus GSI API functions. If so, the next phase of the application starts; otherwise, the user is requested to create a proxy certificate (i.e., a valid credential) by using `grid-proxy-init` command, thus allowing user and user programs to authenticate to resources on the Grid. Once authenticated, they are granted certain rights to use grid resources.

4.2.2. Heterogeneity and dynamic availability

MDS is the Globus service for providing our algorithm with the ability of handling with the heterogeneity and dynamic availability of grid resources. A large set of information from each machine in the Grid is supplied by MDS [6], but we are specially concerned with those showing their current load status and their operating system type and version. Querying MDS periodically for the current load status allows our application to incorporate newly available machines thus responding to the dynamism of this kind of platforms. With the operating system type and version information our algorithm is empowered to handle heterogeneous machines. Let us now detail this procedure. As commented before, the distributed algorithm we have developed is based on the execution of several processes, each of them executing a sequential algorithm that explores a different part of the search space. We name this program `ParetoGenerate`. If the Grid is composed of heterogeneous computing resources, an appropriate executable file for each one must be obtained (e.g., `ParetoGenerate-Linux` or `ParetoGenerate-Solaris`); so, when a new machine is available, our application retrieves its operating system type and version from MDS and the corresponding executable version is sent.

4.2.3. Communication

As shown in its basic operation above, our application uses shared files, which are managed by the GRAM and GASS Globus services, for exchanging information with remote computing resources: transferring the executable files and retrieving the output result files. This communication mechanism allows for robustness and simplicity and, since those services are built on top of Grid Security Infrastructure (GSI), integrity and confidentiality of communication are guaranteed.

4.2.4. Load balancing

To cope with load balancing due the possible heterogeneity of the processors and the networks in the Grid, we have to adjust the grain of the computation to achieve a good balance between computation/communication. The adjustment can be static or dynamic; as our objective in this paper is to obtain experiences using Globus we have used a static approach in order to simplify our algorithm. However, Globus also provides the developers with tools for implementing dynamic load balancing strategies. This can be done via an RSL attribute called `MaxTime`, which allows for explicitly set the maximum walltime or CPU time of a single execution of the executable file. Using this attribute we can dynamically adjust the grain of the computation to some extent as follows. Initially, a number of processes explores subspaces of equal size. Then, each one is let to run for `MaxTime` seconds and, if this time is elapsed, new processes exploring smaller portions of the search space are considered (i.e., dynamically reducing the grain of the computation). This procedure can be recursively done until a desired granularity for the processes is reached.

4.3. Computational results

This section is devoted to present the evaluation of both the sequential and the distributed enumerative search algorithm. First, we detail the multi-objective problems selected as benchmark and the Pareto fronts produced when solving them with the sequential algorithm (i.e., the `ParetoGenerate` plain program). Next, we carry out a further analysis of the sequential and distributed enumerative algorithms in order to explain the results obtained with Globus.

We have installed Globus Toolkit 2.2.4 in 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 implemented in C++ and compiled with `GCC v.2.95` using the option

-O3. Their source code is available for public download in http://neo.lcc.uma.es/Software/ESaM/. Although a 16 PCs cluster is far from being considered a Grid system, we consider that it is enough for our purposes of evaluating the use of Globus to provide useful experiences to the multi-objective optimization research community.

4.3.1. Benchmark of multi-objective problems

We have selected four multi-objective problems from the specialized literature. In concrete, we have chosen the following problems from the book of Coello et al. [3]: Fonseca, Kursawe, and Osyczka2. They are well-known problems and they are frequently used as benchmark to evaluate multi-objective optimization algorithms. Additionally, we have included in our study Golinski's speed reducer problem [35]; this is an interesting problem due to its high number of constraints. The definition of these problems appears in Table 1.

4.3.2. Results of the sequential program

We have solved the four problems included in the benchmark with the sequential program using the discretization degree indicated in Table 2 (e.g., a value of 500 implies that each range of values is divided into

Table 1
Formulation of the multi-objective problems used

Problem	Definition	Constraints
Fonseca	$\mathrm{Min}\,F = (f_1(\vec{x}), f_2(\vec{x}))$ $f_1(\vec{x}) = 1 - e^{-\sum_{i=1}^{n}\left(x_i - \frac{1}{\sqrt{n}}\right)^2}$ $f_2(\vec{x}) = 1 - e^{-\sum_{i=1}^{n}\left(x_i + \frac{1}{\sqrt{n}}\right)^2}$	$-4 \leqslant x_i \leqslant 4;\ i = 1,2,3$
Kursawe	$\mathrm{Min}\,F = (f_1(\vec{x}), f_2(\vec{x}))$ $f_1(\vec{x}) = \sum_{i=1}^{n-1}\left(-10e^{\left(-0.2*\sqrt{x_i^2+x_{i+1}^2}\right)}\right)$ $f_2(\vec{x}) = \sum_{i=1}^{n}\left(\lvert x_i\rvert^a + 5\sin(x_i)^b\right)$	$-5 \leqslant x_i \leqslant 5$ $i = 1,2,3$ $a = 0.8$ $b = 3$
Osyczka2	$\mathrm{Min}\,F = (f_1(\vec{x}), f_2(\vec{x}))$ $f_1(\vec{x}) = -(25(x_1-2)^2 + (x_2-2)^2$ $+(x_3-1)^2(x_4-4)^2 + (x_5-1)^2)$ $f_2(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$	$0 \leqslant x_1 + x_2 - 2$ $0 \leqslant 6 - x_1 - x_2$ $0 \leqslant 2 - x_2 + x_1$ $0 \leqslant 2 - x_1 + 3x_2$ $0 \leqslant 4 - (x_3-3)^2 - x_4$ $0 \leqslant (x_5-3)^3 + x_6 - 4$ $0 \leqslant x_1, x_2, x_6 \leqslant 10$ $1 \leqslant x_3, x_5 \leqslant 5$ $0 \leqslant x_4 \leqslant 6$
Golinski	$\mathrm{Min}\,F = (f_1(\vec{x}), f_2(\vec{x}))$ $f_1(\vec{x}) = 0.7854x_1x_2^2(10x_3^2/3 + 14.933x_3 - 43.0934)$ $-1.508x_1(x_6^2 + x_7^2) + 7.477(x_6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2)$ $f_2(\vec{x}) = \dfrac{\sqrt{(745.0x_4/x_2x_3)^2 + 1.69*10^7}}{0.1x_6^3}$	$\dfrac{1.0}{x_1x_2^2x_3} - \dfrac{1.0}{27.0} \leqslant 0;\ \dfrac{1.0}{x_1x_2^2x_3} - \dfrac{1.0}{27.0} \leqslant 0$ $\dfrac{x_4^3}{x_2x_3^2x_6^4} - \dfrac{1.0}{1.93} \leqslant 0;\ \dfrac{x_5^3}{x_2x_3x_7^4} - \dfrac{1.0}{1.93} \leqslant 0$ $x_2x_3 - 40 \leqslant 0;\ x_1/x_2 - 12 \leqslant 0$ $5 - x_1/x_2 \leqslant 0;\ 1.9 - x_4 + 1.5x_6 \leqslant 0$ $1.9 - x_5 + 1.1x_7 \leqslant 0;\ f_2(x) \leqslant 1300$ $\dfrac{\sqrt{(745.0x_5/x_2x_3)^2 + 1.575*10^8}}{0.1x_7^3} \leqslant 1100$ $2.6 \leqslant x_1 \leqslant 3.6;\ 0.7 \leqslant x_2 \leqslant 0.8$ $17.0 \leqslant x_3 \leqslant 28.0;\ 7.3 \leqslant x_4 \leqslant 8.3$ $7.3 \leqslant x_5 \leqslant 8.3;\ 2.9 \leqslant x_6 \leqslant 3.9$ $5.0 \leqslant x_7 \leqslant 5.5$

Table 2
Discretization degree used and sequential execution times (seconds) of the multi-objective problems

Problem	Fonseca	Kursawe	Osyczka2	Golinski
Discretization degree	500	500	40	30
$t_{1\,\mathrm{CPU}}$	3782	5377	6379	8382

500 partitions). This table also shows the time required to solve each problem. The Pareto fronts obtained are included in Fig. 5. From these results we observe that the points of the problems Fonseca, Kursawe, and Osyczka2 yield almost continuous lines, which is an accurate result to serve as a reference for heuristically computed fronts (see Section 5). The times to solve these problems range between 1 and 2 h, so their solution does not require using a Grid system.

Turning to the Golinski problem, we observe that a finer precision is needed to obtain a Pareto front of good quality. The shown one is too imprecise intentionally, to get results in acceptable times. The time reported in Table 2 concerning this problem indicates that 2.33 h of computing time is required with a discretization degree of 30. To roughly estimate the order of magnitude of the computing time that would be necessary to solve the problem with a precision of 100, we can assume that if the computing of 30^7 iterations (30 partitions and the problem has 7 variables) takes 8.382 s, a constraint test plus an evaluation of the two functions (i.e., an iteration) requires an average time of 3.8e−7 s. Thus, in the case of a precision of 100 partitions per variable, the total computing time to carry out 100^7 iterations would be in the order of 3.8e+7 s (443 days). This justifies the development of Grid-enabled algorithms in order to obtain a more accurate Pareto fronts in an affordable amount of time.

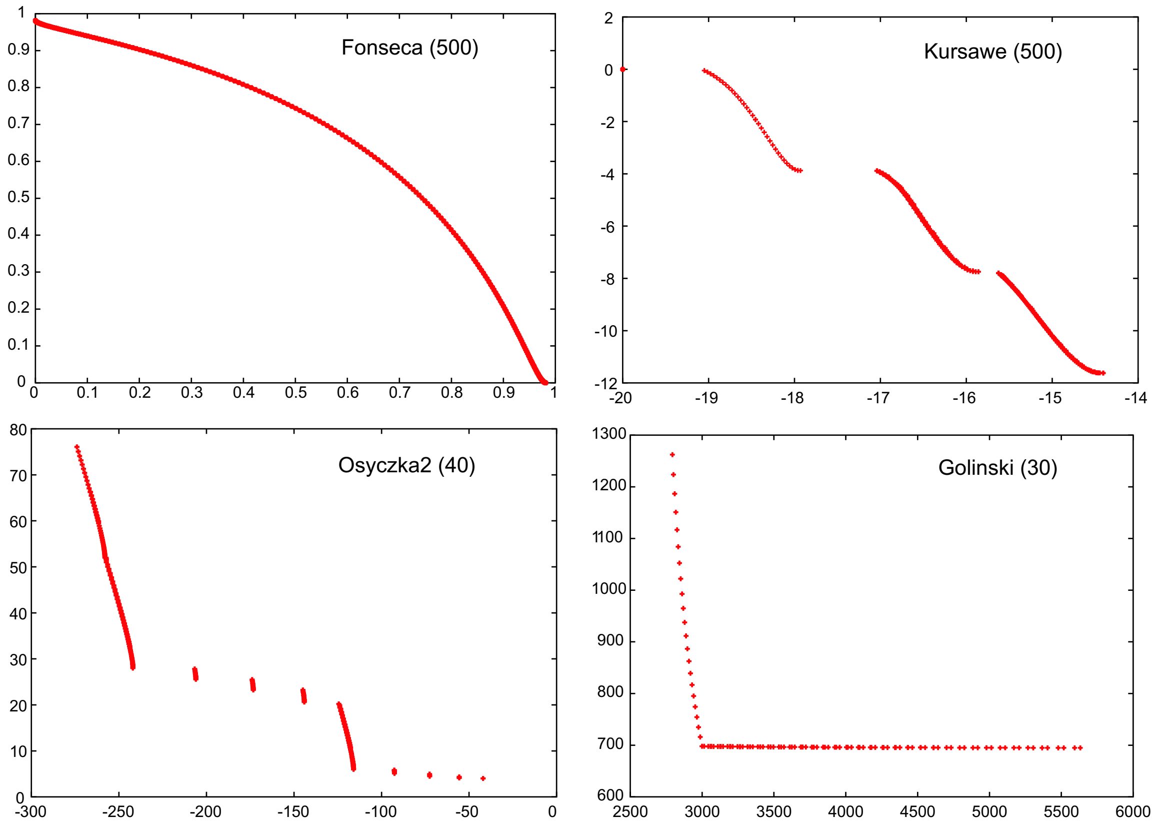

Fig. 5. Pareto fronts of the problems (the values between parentheses indicate the discretization degree) with the sequential algorithm.

4.3.3. *Analysis of the sequential program*

In this section, we present a quantitative analysis of the sequential program by studying the solution of the problems Kursawe and Golinski, with 300 and 20 partitions per decision variable, respectively.

To perform the analysis, we split the search space into several disjoint sets by dividing the range of the first decision variable of the multi-objective problem into disjoint subranges, and each of them has been explored separately by the sequential program; then, we use the GNUs `gprof` tool to obtain profile information of the execution of the subtasks. To get useful information it is enough to consider a small number of subtasks; in our case, we use 5 subtasks. We show the results in Table 3. For each subtask, we include the total running time (in seconds), the time consumed in evaluating the constraints, the time dedicated to evaluate the objective functions and carrying out the dominance tests, and, finally, the number of points of the resulting sub-Pareto front.

For the problem Kursawe, the whole time is spent in evaluating the functions and performing dominance tests, since it is an unconstrained problem. Note that the execution time of a subtask is directly proportional to the number of non-dominated points found, since more points induce the computation of more dominance tests.

When analyzing the problem Golinski, we observe that the first four subtasks do not locate any point in the Pareto front, since the constraints are not fulfilled by any vector of variables. Thus, in these situations (subtasks 1–4), 70% of the time is spent in checking the constraints. However, the last subtask produces a front composed of 70 points, which correspond to the optimum Pareto front of the problem. Consequently, there is an increment in the total execution time, and 55% of it is consumed in the function evaluations and the dominance tests.

For each subtask, we have also displayed graphically in Figs. 6 and 7 the percentage of the computation time spent in the different parts of the enumerative algorithm when solving the problems Kursawe and Golinski, respectively. In the former case, Fig. 6 shows that all subtasks require 95% of time for dominance checking and 5% for fitness function evaluation, i.e., distribution of computing time is almost homogeneous in all subtasks for the problem Kursawe. However, this does not hold in Fig. 7 because it is clear that Subtask 5 is different from the first four ones. As explained above, constraints are fulfilled in the region of the search space explored by Subtask 5 and now dominance checking becomes the most time-consuming element of the enumerative algorithm (55% instead of 30% in subtasks 1–4).

Table 3
Results of the execution (in seconds) of the Kursawe and Golinski problems

Subtask		Kursawe	Golinski
1	Total time	22.22	48.27
	Constraints	–	31.94
	Eval. and dom.	22.22	16.33
	Points	88	0
2	Total time	27.77	49.44
	Constraints	–	34.10
	Eval. and dom.	27.77	15.34
	Points	137	0
3	Total time	36.55	50.55
	Constraints	–	34.82
	Eval. and dom.	36.55	15.73
	Points	153	0
4	Total time	25.58	52.27
	Constraints	–	35.86
	Eval. and dom.	25.58	16.41
	Points	102	0
5	Total time	24.30	104.4
	Constraints	–	45.91
	Eval. and dom.	24.30	58.49
	Points	89	70

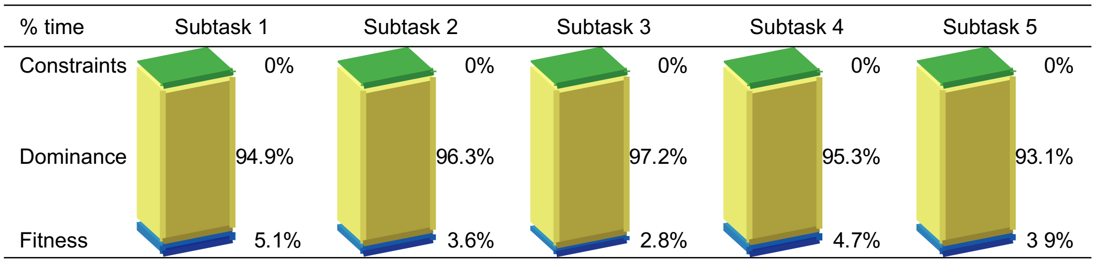

Fig. 6. Percentage of time spent in evaluating the constraints, the dominance tests, and the fitness function for the problem Kursawe.

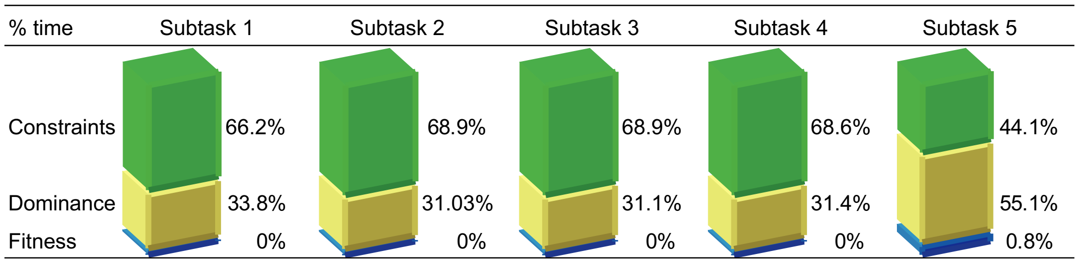

Fig. 7. Percentage of time spent in evaluating the constraints, the dominance tests, and the fitness function for the problem Golinski.

We then conclude that the efficiency of the subtasks in the enumerative search algorithm is heavily problem-dependent. Most probably, this also holds for some other heuristic techniques. In this example, the problem Golinski has 11 constraints and two objective functions, being one of the functions part of a constraint (see the formulation of the problem in Table 1). In unconstrained problems, as the Kursawe's case, a significant amount of time is spent in the dominance tests, while the time required to perform the function evaluation will depend on the number and/or complexity of the functions themselves.

4.3.4. Parallel observations with globus

To evaluate the performance of the Globus-based distributed enumerative search program we have performed four different tests which divide the search space into 16, 32, 64, and 128 subspaces, respectively. Consequently, each test executes 16, 32, 64, and 128 instances of `ParetoGenerate`, always on 16 CPUs. Since the algorithm uses a static load balancing approach, we initially allow for a perfect mapping of instances to processors in the test with 16 subdivisions. Then, we just use a whole multiple of 16 in order to get a better understanding of the working principles of the parallel algorithm with an increasing number of tasks. The results are shown in Table 4. In this table we include, for each test, the parallel execution time ($t_{16\,\mathrm{CPUs}}$), the parallel efficiency (η), the mean and standard deviation of the CPU usage of each machine during the computation ($\bar{x}$ and σ_n, respectively), the total amount of information transmitted (in MBytes) through the network (TIT), and the average network bandwidth used in MBytes/s (BW). The parallel efficiency can be defined as

$$\eta = \frac{s_N}{N} = \frac{\frac{t_{1\,\mathrm{CPU}}}{t_{N\,\mathrm{CPUs}}}}{N} \qquad (1)$$

where N is the number of processors ($N = 16$ in this case) and s_N is the speedup ($s_N = t_{1\,\mathrm{CPU}}/t_{N\,\mathrm{CPUs}}$). A graphical comparison of these results is shown in Fig. 8.

We first begin with the analysis of the parallel efficiency η (Eq. (1)). With 16 tasks, the distributed enumerative search obtains acceptable η values over 60% for the unconstrained problems (Fonseca and Kursawe). Note that each task is assigned to each processor. The reasons stated in Section 4.3.3 (constraints management) explain the unbalanced workload of the problems Osyczka2 and Golinski, that drop the parallel efficiency down to 39.47% and 25.91%, respectively. The increment in the number of tasks (32, 64, and 128

Table 4
Statistics of running 16, 32, 64, and 128 tasks on 16 CPUs

Tasks		Fonseca	Kursawe	Osyczka2	Golinski
16	$t_{16\,\mathrm{CPUs}}$	318	556	1010	2094
	η	74.33%	60.44%	39.47%	25.91%
	$\bar{x}$	66.28%	44.96%	30.78%	10.67%
	σ_n	3.87%	12.94%	34.51%	22.45%
	TIT	68.76	68.60	68.80	70.28
	BW	0.216	0.123	0.068	0.034
32	$t_{16\,\mathrm{CPUs}}$	364	420	500	1452
	η	64.69%	80.01%	79.74%	37.37%
	$\bar{x}$	54.33%	55.49%	50.52%	24.94%
	σ_n	3.46%	5.01%	19.03%	19.43%
	TIT	136.27	136.39	135.52	135.95
	BW	0.374	0.324	0.271	0.094
64	$t_{16\,\mathrm{CPUs}}$	408	445	718	2256
	η	57.93%	75.52%	55.53%	24.05%
	$\bar{x}$	44.62%	56.14%	70.42%	32.25%
	σ_n	3.17%	4.53%	7.29%	21.82%
	TIT	272.78	271.72	271.95	274.06
	BW	0.669	0.611	0.379	0.121
128	$t_{16\,\mathrm{CPUs}}$	554	582	1381	3053
	η	42.67%	57.74%	28.87%	17.77%
	$\bar{x}$	31.01%	30.38%	69.86%	58.09%
	σ_n	1.77%	2.35%	2.33%	12.20%
	TIT	544.57	544.74	543.22	542.24
	BW	0.983	0.936	0.393	0.178

tasks) with a constant number of processors (16 CPUs) has two a priori consequences. First, the execution time should grow because our Globus-based application must create, manage, and schedule an incremented number of tasks. On the contrary, there exists a higher number of tasks with a finer granularity, and this could enhance the load balance, thus decreasing the run time. These two tradeoff factors allow to improve η for all test problems but Fonseca when we use 32 subdivisions of the search space (see Fig. 8). With 64 and 128 tasks the overload induced by this high "task/number of processors" ratio makes the parallel efficiency to decrease for all the problems.

The previous results about the parallel efficiency are further investigated in the light of the CPU usage. For the unconstrained problems, the configuration with the highest η value is generally achieved when the mean CPU usage ($\bar{x}$) is also the highest one (see the Fonseca problem using 16 tasks). This effect does not appear for the constrained problems, since the unbalanced workload provokes that some CPUs compute for a long time while others are idle, hence inducing high $\bar{x}$ values and low efficiency. This behavior is characterized by large standard deviations of the CPU usage (see the σ_n values in Table 4), i.e., these values of σ_n point out an unbalanced workload that therefore induces a low parallel efficiency.

Let us now turn to analyze the network utilization of our Globus-based distributed enumerative search. We can notice that the total information transmitted through the network (TIT) is almost the same for all the problems when the search space is divided into an equivalent number of regions: about 68 MB for 16 tasks, 136 MB for 32 tasks, 272 MB for 64 tasks, and 544 MB for 128 tasks. This is an expected (and quasi-linear) result because a higher number of tasks implies gathering a larger amount of results from the remote machines.

As to the bandwidth measure (BW) (Fig. 8), the constrained problem Golinski uses a low almost-constant bandwidth because, as the amount of information transmitted by the application is increased, the execution time is also increased, thus keeping the average bandwidth constant. For the rest of the test problems this does not hold, and therefore the BW metric grows as the search space is divided into a larger number of regions.

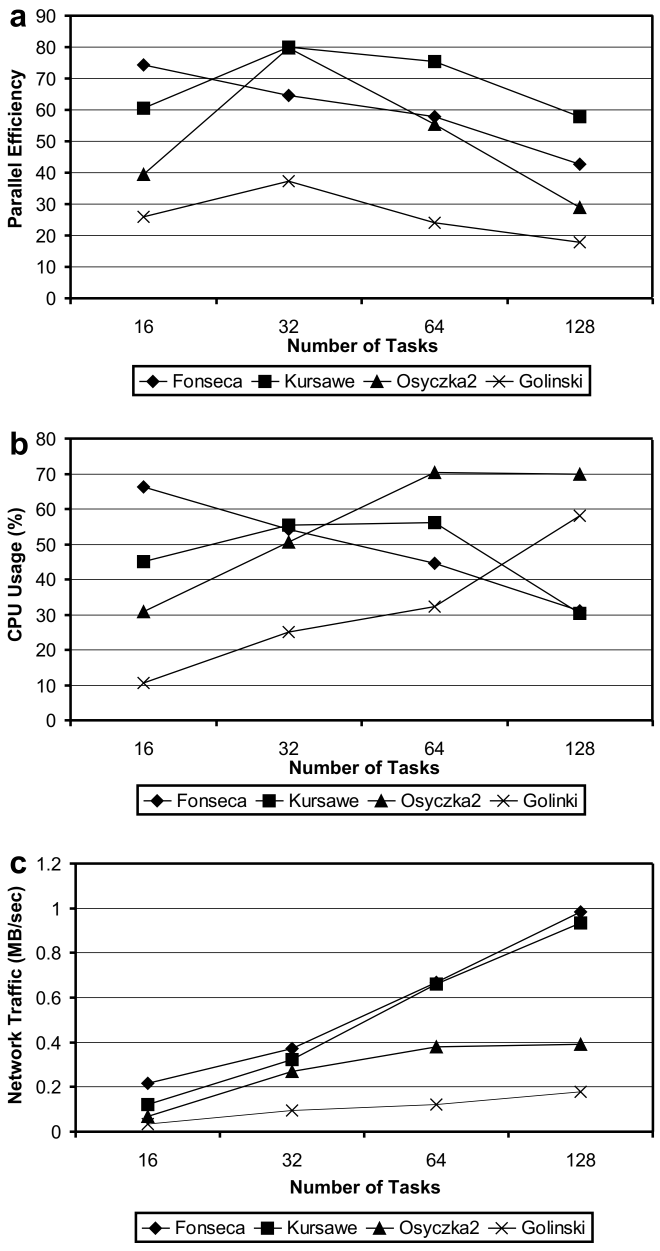

Fig. 8. Comparison of parallel efficiency (a), CPU usage (b), and network bandwidth (c).

5. Globus and heuristic techniques for multi-objective optimization

We now turn to apply Globus to run a distributed evolutionary algorithm. Furthermore, we will make use of the Pareto fronts obtained by the enumerative search algorithm 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 [32].

Our Grid-enabled evolutionary algorithm, named gPAES, is based on the Pareto Archived Evolution Strategy (PAES) algorithm [7]. 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

```
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,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
```

Fig. 9. Pseudo-code for (1 + 1)-PAES.

archive of previously found solutions in order to identify the approximate dominance ranking of the current and candidate solutions vectors.

5.1. (1 + 1)-PAES

We have implemented our C++ version of the (1 + 1)-PAES algorithm based on the description presented in [7]. The (1 + 1)-PAES represents the simplest non-trivial approach to a multi-objective local search procedure. An outline of this algorithm is included in Fig. 9. It maintains a single solution that, at each iteration, is mutated (line 2 in Fig. 9) according to normal ES mutation operations [36] 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 Fig. 9).

Since the aim of the multi-objective search is to find a set of spread non-dominated solutions, PAES uses a crowding procedure based on an adaptive numerical grid that recursively divides up the objective space [7].

5.2. gPAES

Heuristic algorithms can also 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 some defined metrics. The gPAES algorithm uses the same services as the Globus-based enumerative search (Section 4.2): it looks for available machines in the Grid, builds an RSL specification that launches the remote executions of the PAES algorithm and retrieves the resulting Pareto fronts, then it calculates the desired metrics and stores the best front with respect to these metrics. 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.

5.3. Results

Several metrics have been proposed for guiding the search (online use) and for measuring the results of Pareto-based multi-objective optimization algorithms (offline use). The quality of a Pareto front is given by its convergence degree to the true Pareto front and the homogenous diversity of the solutions. In this work we use the metrics M_1^* [37] and Δ [38]. 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. These metrics are commonly used in works in this field. Their formulations are

$$M_1^* = \frac{1}{|Y'|} \sum_{d' \in Y'} \min\{\|d' - \bar{d}\|^*; \bar{d} \in \overline{Y}\} \tag{2}$$

where Y', $\overline{Y} \subseteq Y$ are sets of objective vectors. Ideally, this metric should be zero. Δ is defined as

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1}|d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}} \tag{3}$$

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* solutions of the exact Pareto front in the objective space (see [38] for the details). Note that it is only possible to calculate these two metrics since we have the Pareto optimal set from the enumerative algorithm (Section 4).

For the Grid-enabled PAES algorithm, we have used the same Globus platform as for the enumerative case (Section 4.3). As stated before, gPAES 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. For the first parameter, we have tested two configurations, named gPAES$_{10}$ and gPAES$_{100}$, running 10 and 100 parallel subalgorithms. For the second one, we have used the M_1^* metric (online usage).

We also present an evaluation of our implementation of the sequential PAES algorithm for comparison purposes. This sequential PAES algorithm uses the following configuration: binary representation with a precision of 5 digits, bit-flip mutation, 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 25,000 function evaluations. Note that, while the sequential PAES performs 25,000 function evaluations, gPAES$_{10}$ and gPAES$_{100}$ carry out 250,000 and 2,500,000 evaluations, respectively. We have considered the same problems as in the enumerative algorithm: Fonseca, Kursawe, Osyzcka2, and Golinski. All the presented results are the average values over 30 independent runs.

In order to compare the results (see Table 5) of the 30 independent runs of the sequential PAEs and both configurations of gPAES, gPAES$_{10}$ and gPAES$_{100}$, we have applied the metrics $M1^*$ and Δ. We want to remark that we have used the metric $M1^*$ 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$_{10}$ and gPAES$_{100}$, like Δ (always offline). The columns "ANOVA" in Table 5 present the result of an ANOVA test comparing the three algorithms with a 5% of significance level. The ANOVA test is used to state whether the compared results of more than two experiments have statistical significance or not.

Let us now turn to the analysis of the results according to the first considered metric $M1^*$. First, it can be seen 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. 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, more than three and ten times 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 5 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. We can conclude that the model of search proposed by gPAES is able to take advantage of the high computational power offered by Grid technologies.

Table 5
Results of gPAES

Problem	$M1^*$				Δ			
	PAES	gPAES$_{10}$	gPAES$_{100}$	ANOVA	PAES	gPAES$_{10}$	gPAES$_{100}$	ANOVA
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	–

In the right part of Table 5 we show the metric Δ. 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, in gPAES a new multi-objective problem appears in the way of choosing the best front from the independent runs, i.e., if it considers several and possibly conflicting metrics to arrange the fronts, a tradeoff between them should be achieved.

6. Conclusions and future work

Grid-enabled technologies offer a strategic opportunity to develop new algorithms for solving optimization problems. In this context, we have used the Globus Toolkit, a *de facto* standard system for Grid computing, to implement a distributed enumerative search algorithm for solving multi-objective problems. Our goal has been to gain experience with Grid technologies to face more complex algorithms in the future, as we did with 110 machines and Condor in [32].

We have solved a benchmark composed of both constrained and unconstrained multi-objective problems. We have analyzed the parallel efficiency, mean CPU usage, and network bandwidth when partitioning the search space into 16, 32, 64, and 128 regions in a workstation cluster. We have observed that promising results are worked out when solving unconstrained problems if they are computationally expensive; in the case of constrained problems, a more elaborated parallelization scheme is required to devote more resources for subranges of the constrained variables, which constitutes a matter of future work.

On the other hand, we have presented an approximation of Grid-enabled technologies based on Globus to numerically improve an evolutionary algorithm. The algorithm presented, named gPAES, uses Globus to execute a number of sequential PAES algorithm 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.

Future research is in the line of using Globus in a Grid composed of hundred of computers. We also intend to apply the experiences obtained in this work to face the parallelization of other heuristic techniques in Grids, and the development of new models of search based on Grid platforms for multi-objective optimization. A theoretical study of the performance of the proposed algorithms is also a matter of future work.

Acknowledgements

This work has been partially funded by the Ministry of Science and Technology and FEDER under contract TIN2005-08818-C04-01 (the OPLINK project).

References

[1] A. Osyczka, Multicriteria Optimization for Engineering Design, Academic Press, 1985.
[2] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, John Wiley & Sons, 2001.
[3] C. Coello, D. Van Veldhuizen, G. Lamont, Evolutionary Algorithms for Solving Multi-Objective Problems, Genetic Algorithms and Evolutionary Computation, Kluwer Academic Publishers, 2002.
[4] M. Baker, R. Buyya, D. Laforenza, Grids and grid technologies for wide area distributed computing, Software: Practice and Experience 32 (2002) 1437–1466.
[5] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1999.
[6] I. Foster, C. Kesselman, Globus: a metacomputing infrastructure toolkit, International Journal of Supercomputer Applications 11 (2) (1997) 115–128.
[7] J. Knowles, D. Corne, Approximating the nondominated front using the Pareto archived evolution strategy, Evolutionary Computation 8 (2) (2000) 149–172.
[8] A. Grama, V. Kumar, State of the art in parallel search techniques for discrete optimization, IEEE Transactions on Knowledge and Data Engineering 11 (1) (1999) 28–35.
[9] A. Migdalas, P. Pardalos, S. Story, Parallel Computing in Optimization, Applied Optimization, vol. 7, Kluwer, 1997.

[10] B. Gendron, T.G. Crainic, Parallel branch and bound algorithms: survey and synthesis, Operations Research 42 (6) (1994) 1042–1066.
[11] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (5) (2002) 443–462.
[12] E. Alba, J. Troya, A survey of parallel distributed genetic algorithms, Complexity 4 (4) (1999) 31–52.
[13] D. Van Veldhuizen, J. Zydallis, G. Lamont, Considerations in engineering parallel multiobjective evolutionary algorithms, IEEE Transactions on Evolutionary Computation 87 (2) (2003) 144–173.
[14] J. Lemesre, C. Dhaenens, E.-G. Talbi, A parallel exact method for a bicriteria permutation flow-shop problem, in: Project Management and Scheduling (PMS'04), Nancy, France, 2004, pp. 359–362.
[15] D. Van Veldhuizen, G. Lamont, Multiobjective evolutionary algorithm test suites, in: Proc. of the 1999 ACM Symp. on Applied Computing, San Antonio, Texas, 1999, pp. 351–357.
[16] J. Kamiura, T. Hiroyasu, M. Miki, S. Watanabe, MOGADES: multi-objective genetic algorithm with distributed environment scheme, in: Proc. of the 2nd Int. Workshop on Intelligent Systems Design and Applications (ISDA'02), 2002, pp. 143–148.
[17] N. Keerativuttitumrong, C. Chaiyaratana, V. Varavithya, Multi-objective co-operative co-evolutionary genetic algorithm, in: Parallel Problem Solving from Nature (PPSN VII), 2002, pp. 288–297.
[18] M. Knarr, M. Goltz, G. Lamont, J. Huang, In situ bioremediation of perchlorate-contaminated groundwater using a multi-objective parallel evolutionary algorithm, in: Proc. of the 2003 Congress on Evolutionary Computation (CEC'2003), 2003, pp. 1604–1611.
[19] S. Manos, L. Poladian, Novel fibre Bragg grating design using multiobjective evolutionary algorithms, in: Proc. of the 2003 Congress on Evolutionary Computation (CEC'2003), 2003, pp. 2089–2095.
[20] N. Xiao, M. Armstrong, A specialized island model and its applications in multiobjective optimization, in: Genetic and Evolutionary Computation Conference (GECCO'03), LNCS 2724, 2003, pp. 1530–1540.
[21] S. Xiong, F. Li, Parallel strength Pareto multi-objective evolutionary algorithm, in: Proc. of the 2003 Congress on Evolutionary Computation (CEC'2003), 2003, pp. 681–683.
[22] J. Malard, A. Heredia-Langner, D. Baxter, K. Jarman, W. Cannon, Constrained De Novo peptide identification via multi-objective optimization, in: Online Proc. of the Third IEEE Int. Workshop on High Performance Computational Biology (HiCOMB 2004), 2004.
[23] F. de Toro Negro, J. Ortega, E. Ros, S. Mota, B. Paechter, J. Martín, PSFGA: parallel processing and evolutionary computation for multiobjective optimisation, Parallel Computing 30 (5–6) (2004) 721–739.
[24] K. Parsopoulos, D. Tasoulis, N. Pavlidis, V. Plagianakos, M. Vrahatis, Vector evaluated differential evolution for multiobjective optimization, in: Proc. of the IEEE 2004 Congress on Evolutionary Computation (CEC 2004), 2004, pp. 204–211.
[25] P. Delisle, M. Krajecki, M. Gravel, C. Gagn, Parallel implementation of an Ant Colony optimization metaheuristic with OpenMP, in: 3rd European Workshop on OpenMP (EWOMP01), 2001.
[26] K. Parsopoulos, D. Tasoulis, M. Vrahatis, Multiobjective optimization using parallel vector evaluated particle swarm optimization, in: Proc. of the IASTED International Conference on Artificial Intelligence and Applications, 2004, pp. 823–828.
[27] N. Jozefowiez, F. Semet, E.-G. Talbi, Parallel and hybrid models for multi-objective optimization: application to the vehicle routing problem, in: Parallel Problem Solving from Nature (PPSN VII), 2002, pp. 271–280.
[28] K. Anstreicher, N. Brixius, J.-P. Goux, J. Linderoth, Solving large quadratic assignment problems on computational grids, Mathematical Programming 91 (2002) 563–588.
[29] M. Neary, P. Cappello, Advanced Eager scheduling for Java-based adaptively parallel computing, in: Proc. ACM Java Grande/ISCOPE Conference, 2002, pp. 56–65.
[30] Y. Tanimura, T. Hiroyasu, M. Miki, K. Aoi, The system for evolutionary computing on the computational grid, in: Parallel and Distributed Computing and Systems (PDCS 2002), 2002, pp. 56–65.
[31] S. Wright, Solving optimization problems on computational grids, Tech. Rep., Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill, November 2000.
[32] A. Nebro, F. Luna, E. Alba, Multi-objective optimization using grid computing, Soft Computing Journal, in press, doi:10.1007/s00500-006-0096-0.
[33] C. Coello, G. Toscano, Multiobjective optimization using a micro-genetic algorithm, in: GECCO-2001, 2001, pp. 274–282.
[34] G. Aloisio, E. Blasi, M. Cafaro, I. Epicoco, S. Fiore, S. Mocavero, A grid environment for diesel engine chamber optimization, in: Proc. of ParCo2003, 2003, pp. 599–607.
[35] A. Kurpati, S. Azarm, J. Wu, Constraint handling improvements for multi-objective genetic algorithms, Structural and Multidisciplinary Optimization 23 (3) (2002) 204–213.
[36] T. Bäck, Evolutionary Algorithms: Theory and Practice, Oxford University Press, New York, USA, 1996.
[37] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Evolutionary Computation 8 (2) (2000) 173–195.
[38] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.