

A Globus-Based Distributed Enumerative Search Algorithm for Multi-objective Optimization

Francisco Luna Antonio J. Nebro Enrique Alba
Departamento de Lenguajes y Ciencias de la Computación
E.T.S.I. Ingeniería Informática, Universidad de Málaga, Málaga (Spain)
Email: {flv,antonio,eat}@lcc.uma.es

Technical Report LCC 2004/02
March, 2004

Abstract—Enumerative search is a technique to solve multi-objective optimization problems based on evaluating each possible solution from a given finite set. The technique is simple and computationally expensive, but it is the only way at present to compute exact Pareto fronts in multi-objective problems. In this context, Grid computing systems offer a potentially large amount of computing power that can be used to overcome the mentioned drawback to some extent. In this paper, we analyze several practical and technical issues concerning the use of the Globus Toolkit, a *de facto* standard system for Grid computing, to implement a distributed enumerative search algorithm. We have solved a benchmark of multi-objective problems in a cluster of computers and we have analyzed issues such as the parallel efficiency, mean CPU use, and network bandwidth utilization. Furthermore, we also use Globus to develop a new technique named grid- μ GA, an extension of the micro-GA algorithm. The results indicate that using Globus is a promising choice to solve multi-objective problems in Grid computing systems.

I. INTRODUCTION

One important goal of multi-objective optimization is to find the Pareto front of a problem. The techniques that can be used to obtain a Pareto front can be classified into three categories: enumerative, deterministic, and stochastic [1]. In recent years, stochastic methods have been widely studied; in particular, evolutionary algorithms have been investigated by many authors [2], [3], [4]. These methods do not guarantee to obtain the optimal solution, but they provide appropriate solutions to a wide range of optimization problems which other deterministic methods find difficult. On the contrary, enumerative search, which is a non heuristic (deterministic) technique, is a conceptually simple search strategy based on evaluating each possible solution in a finite search space.

The drawback of this technique resides in its inability to scale as the search space becomes larger. Despite of 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 stochastic algorithms. In consequence, the quality of the solutions produced by these stochastic algorithms can be measured in a non-subjective way by researchers.

In this context, the increasing impact in last years of Grid computing [5], [6] appears as an alternative that allows to

apply techniques and algorithms that are impractical for typical distributed systems such as clusters of computers. Grids enable applications to handle thousands of distributed heterogeneous computing resources as a single virtual machine; thus, enumerative methods can be viable to obtain optimal solutions of a number of problems where heuristics are not able of ensuring the computation of the optimum Pareto Front.

Grids can be considered as systems that coordinate elements that are not subject to centralized control, integrating resources and users from different domains. They use standard, open, general-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. Besides, a Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, security, and/or co-allocation of multiple resource types. Thus, complex user demands can be satisfied.

Much of the Grid technology relies on the Globus Toolkit [7] (<http://www.globus.org>). Globus has emerged as the *de facto* standard for Grid computing so that various large-scale Grid deployments being undertaken within the scientific community utilize the software services it provides. Examples are the *EU DataGrid* (<http://www.eu-datagrid.org>) and the *NASA's Information Power Grid* (<http://www.ipg.nasa.gov>).

In this paper, we analyze the Globus Toolkit in order to parallelize a distributed 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 which lead us to consider in the future the development of more complex algorithms in true computational grids. An additional contribution of this work is the use of Globus to develop a new technique named grid- μ GA, an extension of the micro-GA algorithm [8].

The paper is organized as follows. In Section II, we discuss related work concerning multi-objective optimization and parallel computing. Then, Section III reviews the major components of the Globus Toolkit. It is followed by the description of the enumerative search algorithm for multi-objective optimization in Section IV. The next section (V)

presents the results obtained. Section VI details the use of Globus to introduce the new grid- μ GA algorithm. Finally, we outline the conclusions and future research lines in Section VII.

II. RELATED WORK

Parallel computers have been widely used in the field of mono-objective optimization [9], [10]. In the case of deterministic techniques, the idea is, in general, to solve the problems more rapidly, or to solve more complex problems. In the context of stochastic methods, parallelism is not only used for solving problems more rapidly, but for developing more efficient models of search, because a parallel stochastic algorithm can be more effective than a sequential one, even when run in a single processor [11].

However, in the context of multi-objective optimization few efforts have been devoted to parallel implementations, as stated in [1]. Some works concerning evolutionary techniques and distributed systems are [12], [13], but, to the best of our knowledge, there is not any related work concerning multi-objective optimization and Grid computing.

A parallel enumerative search algorithm for multi-objective optimization is described in [14], but it 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 taking advantage of the power that Grid systems offers in order to use hundreds and thousands of computers to solve a given problem.

III. 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: *Resource Management*, *Information Services*, and *Data Management*. They are built on top of the underlying Grid Security Infrastructure (GSI).

A. Grid Security Infrastructure (GSI)

GSI provides elements for secure authentication and communication. The infrastructure is based on the TLS protocol (Transport Layer Security), public key encryption, and X.509

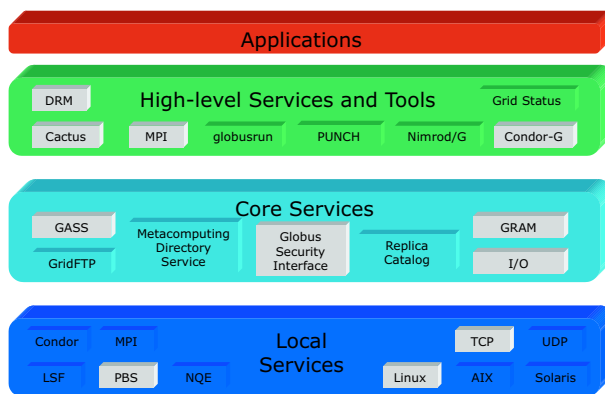


Fig. 1. Globus layered architecture.

certificates. It provides single sign-on authentication, communication protection, and some initial support for restricted delegation. In brief, *single sign-on* allows a user to authenticate once and thus creates a proxy credential that a program can use to authenticate with any remote service on the user's behalf. *Delegation* allows for the creation and communication to a remote service of delegated proxy credentials that the remote service can use to act on the user's behalf.

B. Resource Management

The resource management provides support for resource allocation, job submission (remotely running executable files and receiving results), and managing job status and progress. Its primary components are the Grid Resource Allocation Manager (GRAM) and the Global Access to Secondary Storage (GASS).

GRAM is the module that provides remote execution of tasks and status management of the execution. When a job is submitted by a client, the request is sent to the remote host and handled by a daemon located in the remote host. Then this daemon creates a job manager to start and monitor the job. When the job is finished, the job manager sends the status information back to the client and terminates.

RSL (Resource Specification Language) is the language used by the clients to submit a job. Any job submission request is described in RSL, including the executable file and the conditions under which it must be executed. For example, we can specify the amount of memory needed to execute a job in a remote machine. GRAM uses GASS for providing the mechanism to transfer the output files from server to clients. Some application programming interfaces (APIs) are provided under GSI to furnish secure transfers.

C. Information Services

The information services provide support for collecting information in the Grid and for querying this information. Based on the Lightweight Directory Access Protocol (LDAP), the Grid Resource Information Service (GRIS) and the Grid Index Information Service (GIIS) can be configured in a hierarchy to collect and distribute the information. These two

services are called the Monitoring and Discovery Service (MDS).

The information collected can be static information about the machines as well as dynamic information showing the current CPU usage or disk activity. A rich set of information providers is included with the Toolkit, but it can be extended by the Globus users. While GRIS is the repository of local resource information derived from information providers, GIIS is the repository that contains indexes of resource information registered by the GRIS and other GIIS (it can be seen as a Grid wide information server). The LDAP query language is used to retrieve the desired information.

D. Data management

The data management provides support for transfer files among machines in the Grid and for the management of these transfers. The key component for this service is GridFTP. The word GridFTP can refer to a protocol or a service.

As a protocol, GridFTP intends to be used in all data transfers on the Grid. It is based on FTP, but extends the standard protocol with facilities such as multistreamed transfer, auto-tuning, and Globus based security. This protocol is still in draft level, so Globus does not support the entire set of features currently specified.

On the other hand, Globus Toolkit provides the GridFTP server and GridFTP client. They support two types of file transfer: standard and third-party. The standard file transfer is used when a client sends the local file to the remote machine, which runs the FTP server. Third-party transfer occurs when there is a large file in remote storage and the client wants to copy it to another remote server.

IV. ENUMERATIVE SEARCH ALGORITHM

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

A. Sequential Algorithm

The sequential algorithm we have used is similar to the approach for finding non-dominated sets described in [15] (pp. 36-38). The decision variables, assumed continuous, are discretized with a certain granularity, 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 evaluation and dominance check. An outline of the algorithm is shown in Fig. 2.

The number of iterations carried out by the algorithm depends on the number of decision variables N and the desired granularity G . However, the complexity of the algorithm can be strongly influenced by the constraint test and the evaluation of the objective functions. Clearly, the time required

```

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

Fix the granularity 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. 2. Pseudo-code of the sequential enumerative algorithm.

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 V-C.

This algorithm has been implemented in C++, what ensures portability and efficiency. It has been designed to simplify the incorporation of the problems to solve, what is achieved by using the inheritance mechanism. The code is available for download in <http://neo.lcc.uma.es/Software/ESaM/>.

B. Globus-Based Distributed Enumerative Search

The distributed algorithm we have developed is based on the execution of several processes, each of them executing the sequential algorithm but exploring a different part of the search space. We name this program `ParetoGenerate`. When each process finishes, it writes in secondary memory the results it obtains. Once all these processes have finished, a new process, called `ParetoMerge`, reads all the secondary memory and gathers the results to obtain the Pareto optimal solutions and the Pareto front by means of applying the dominance test. This parallel algorithm is simple, because inter-process communication is not necessary, and there is only a synchronization point at the end.

To implement this distributed algorithm in Globus, the idea is, given a number of subdivisions of the search space, to launch an instance of `ParetoGenerate` on each available machine of the Grid that we have configured. As the computation of each instance finishes, the result files are retrieved and a new instance is sent to explore a new part of the search space. If the number of machines is smaller than the number of tasks (what it is a usual scenario), we need a job scheduler that can initiate and manage jobs.

To build a distributed system based on Globus, we need to configure the following services:

- Resource management. GRAM and GASS are used to remotely execute the instances of `ParetoGenerate` and to stage in the executable file and stage out their two output result files, respectively.

```

&(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. 3. Example of an RSL specification.

- Information services. MDS is used to search for the available machines in the Grid.
- Security infrastructure. We must obtain X.509 certificates for authenticating and authorizing users and hosts, as well as the LDAP services running.

Once the system has been built and these services are operative, we must program a job scheduler because Globus does not include such feature. We have used the C API provided by the toolkit that allows to integrate our application with the various components and services provided by the Globus framework. Briefly, our application firstly checks if the user credentials are valid. Next, it performs an LDAP query to the 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. 3.

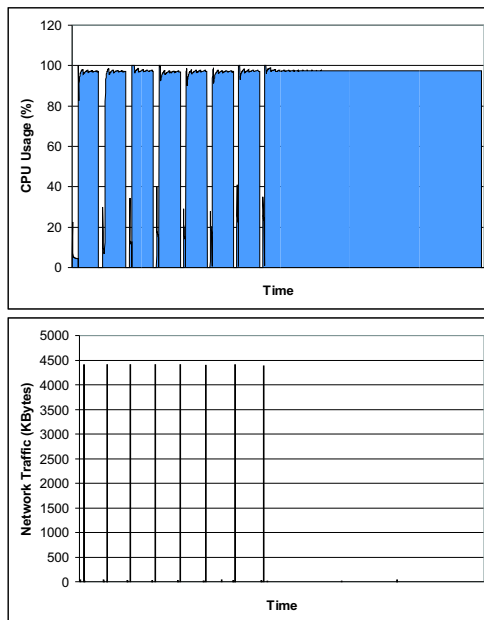


Fig. 4. Example of CPU usage and network traffic during a computation.

This RSL string is composed of several RSL attributes which provide (in the order they appear in the figure): 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 staged out from the execution host to the local host. This RSL example executes specifically the 15th instance of `ParetoGenerate`

for the problem named Fonseca (its problem identifier is 1), where the search space has been partitioned into 128 regions; 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.

For each submitted job, we use a thread that waits for job completion. When the job finishes, its thread synchronizes with the application's main thread, which creates an RSL specification for another `ParetoGenerate` instance. Then, another thread is created in order to wait for this job completion, and so on. The process continues until no more search space regions are left to explore. When they all have finished, the `ParetoMerge` program builds the exact Pareto front. A typical example of a node computation with this application appears in Fig. 4, where we show the CPU usage of a node and the network traffic it produces. Note that, for each task computed by the node, there exists a transmission of information through the network. As it can be seen in the figure, the execution time of the different tasks is not uniform (we will explain this fact later, in Subsection V-C).

V. COMPUTATIONAL RESULTS

This section is devoted to evaluate 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 (C++ plain program). Next, we carry out a further analysis of the sequential and distributed enumerative algorithms in order to explain the results obtained when we use 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 compiled with GCC v. 2.95 using the option `-O3`.

A. Benchmark of Multi-objective Problems

We have selected four multi-objective problems from the specialized literature. In concrete, we have chosen problems from the book of Coello *et al.* [1]. The problems are named according to the terminology used in that book: Fonseca, Kursawe, and Osyczka2. Additionally, we have included in our study Golinski's speed reducer problem [16]. The definition

TABLE I
FORMULATION OF THE MULTI-OBJECTIVE PROBLEMS USED.

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

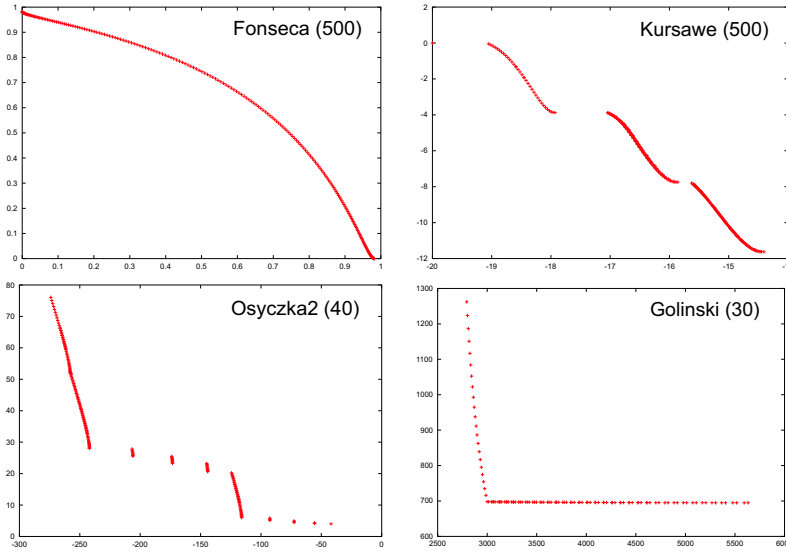


Fig. 5. Pareto fronts of problems (the values between parenthesis indicate the granularity) with the sequential algorithm.

of these problems appears in Table I, accounting for both unconstrained and constrained problems.

B. Results of the Sequential Program

We have solved the four problems included in the benchmark with the sequential problem using the granularity indicated in Table II. This table also shows the times required to solve each problem.

If we analyze the Pareto fronts obtained (Fig. 5), we observe that they show a good quality, which should be enough to serve

as a reference to be compared against the results produced by heuristics techniques (see Section VI).

The times reported in Table II indicate that 1.8 and 2.33 hours of computing time are required to solve the problems Osyczka2 and Golinski with a granularity of 40 and 30, respectively. We have estimated that several months would be required to solve them if a granularity of 100 is desired, what justify the need of using a Grid computing system in order to obtain the Pareto fronts in a reasonable amount of time.

TABLE II

GRANULARITY USED AND SEQUENTIAL EXECUTION TIMES (SECONDS) OF THE MULTI-OBJECTIVE PROBLEMS.

Problem	Fonseca	Kursawe	Osyczka2	Golinski
Granularity	500	500	40	30
$t_{1\text{ CPU}}$	3782	5377	6379	8382

C. Analysis of the Sequential Program

In this subsection, 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 have divided the search space into five parts, and each of them has been solved separately by the sequential program. We have used the GNU's `gprof` tool to obtain profile information of the execution of the five subtasks. We show the results obtained in Table III. For each subtask, we include the total running time (in seconds), the time consumed in evaluating the constraints, the time dedicated for evaluating the functions and carrying out the dominance tests, and, finally, the number of points of the resulting sub-Pareto front found.

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 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 of the Pareto front, since the constraints are not fulfilled by any vector of variables (the time consumed in the evaluation and domination tests is spent in updating local variables). Thus, in these situations

TABLE III

RESULTS OF THE EXECUTION (IN SECONDS) OF THE KURSAWE AND GOLINSKI PROBLEMS.

Subtask		Kursawe	Golinski
1	Total Time	43.53	35.45
	Constraints	–	34.73
	Eval. & Dom.	43.53	3.72
	Points	88	0
2	Total Time	52.94	40.10
	Constraints	–	35.84
	Eval. & Dom.	52.94	4.26
	Points	137	0
3	Total Time	64.05	40.79
	Constraints	–	37.08
	Eval. & Dom.	64.05	3.74
	Points	153	0
4	Total Time	49.60	41.03
	Constraints	–	37.14
	Eval. & Dom.	49.60	3.89
	Points	102	0
5	Total Time	41.77	108.12
	Constraints	–	48.03
	Eval. & Dom.	41.77	60.09
	Points	89	70

(subtasks 1 to 4), the 90% of the time is spent in checking the constraints. However, the last subtask produces a front composed of 70 points, which correspond to the true Pareto front of the problem. Consequently, there is an increment in the total execution time, and the 65% of it is consumed in the function evaluations and the dominance tests. In fact, the results reported by `gprof` show that most of this time is spent in the dominance tests.

We then conclude that the efficiency of the enumerative search algorithm is heavily problem-dependent. Maybe this also holds for some heuristic techniques. In this example, Golinski's problem has 11 constraints and two objective functions, being one of the functions part of a constraint (see the formulation of the problem in Table I). In problems without any constraints, as the Kursawe's one, 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.

D. Parallel Executions with Globus

To evaluate the performance of the Globus-based distributed enumerative search program, we have performed, for each of the problems, four different tests that divide the search space into 16, 32, 64, and 128 regions, respectively. Therefore, each test executes 16, 32, 64, and 128 instances of `ParetoGenerate`. The results are shown in Table IV. In this table we include, for each test, the parallel execution time ($t_{16\text{ CPU}s}$), 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 network bandwidth used in MBytes/second (BW). The parallel efficiency can be defined as

$$\eta = \frac{s_N}{N} = \frac{t_{1\text{ CPU}}}{t_{N\text{ CPU}s} \cdot N}$$

where N is the number of processors ($N = 16$ in this case) and s_N is the speedup ($s_N = t_{1\text{ CPU}}/t_{N\text{ CPU}s}$). A graphical comparison of these results are shown in Fig. 6. We thoroughly discuss the results next.

First, we must notice that the parallel efficiency measures the advantage of using more processors for running the distributed enumerative algorithm. With 16 tasks, a process per processor is assigned, and only the problems Fonseca and Kursawe yield an efficiency above the 60%. The problems Osyczka2 and Golinski suffer of an unbalanced workload, for the reasons analyzed in Subsection V-C, which is also the cause of their poor mean CPU usage (30.78% and 10.67%, respectively). This unbalanced workload is also characterized by the high standard deviation values of the CPU usage (σ_n).

An increment in the number of tasks implies two facts. First, the Grid software must create, manage, and schedule more tasks, and a higher network bandwidth is needed. As a consequence, the efficiency actually drops for 64 and 128 tasks. However, a higher number of tasks with a finer granularity could enhance the load balance, but this only happens

TABLE IV
STATISTICS OF RUNNING 16, 32, 64, AND 128 TASKS.

Tasks		Fonseca	Kursawe	Osyczka2	Golinski
16	$t_{16} CPU_s$	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	$t_{16} CPU_s$	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	$t_{16} CPU_s$	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	$t_{16} CPU_s$	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

when we execute 32 tasks; thus, the efficiency of the problems Kursawe and Osyczka2 improve from 60.44% to 80.01% and from 39.47% to 79.74%, respectively.

An analysis of the times with 64 and 128 tasks reveals that the times required to solve the problems Fonseca and Kursawe are similar, while the solution of the problem Golinski is about 5 times higher. Thus, the same amount of data (see the TIT rows in Table IV) must be transmitted in less time, what is the reason of the increment in the network traffic and the decrement in the CPU usage in the case of the two unconstrained problems. If these two problems were more time consuming, better speedups should be expected.

Finally, the presence of constraints in the problems Osyczka2 and Golinski and their influence in the obtained results lead us to consider that our simple parallelization scheme could be enhanced to include a dynamic load balancing strategy (maybe a search for variable regions of factibility).

VI. GLOBUS AND HEURISTIC TECHNIQUES FOR MULTI-OBJECTIVE OPTIMIZATION

We now turn to apply the optimum Pareto fronts to evaluate a heuristic evolutionary algorithm on a Grid. The new grid- μ GA runs N micro-GAs [8] on Globus with a final merging step of the resulting Pareto fronts. This simple scenario allows to obtain a Pareto front, but it is also useful when reduce the time required to execute a number of instances of an algorithm in order to gather statistical information.

We have used Globus to launch 16, 32, 64, and 128 instances of the micro-GA. As a example, Fig. 7 shows the Pareto front of the problem Osyczka2 calculated with the enumerative algorithm and the front obtained when merging 128 replications of the micro-GA. We have compared the fronts obtained by

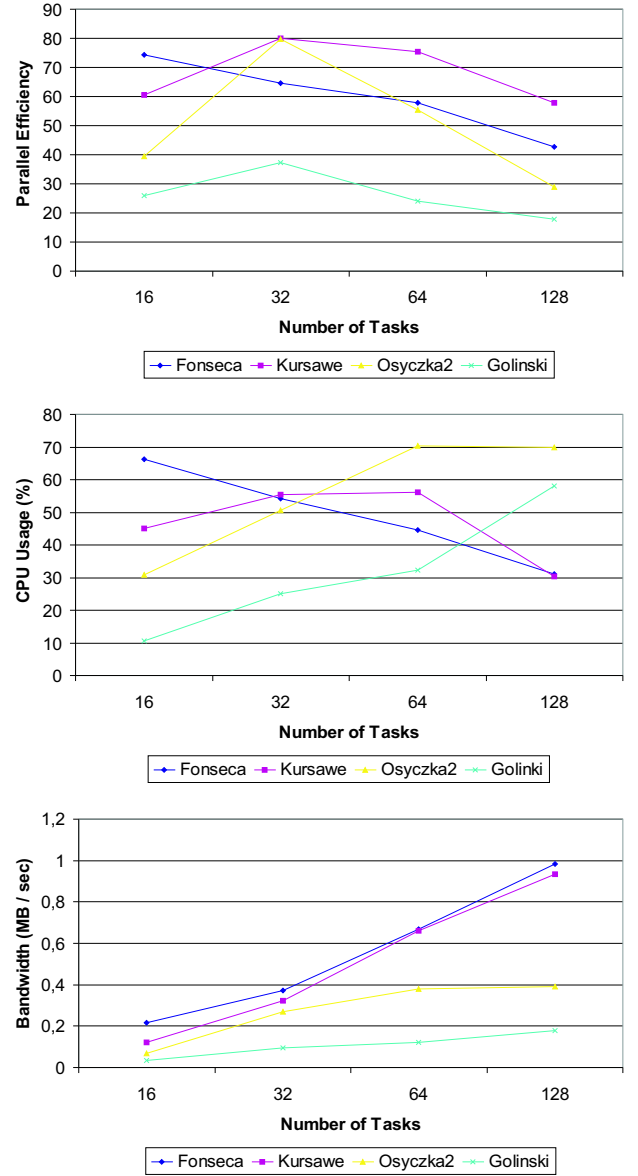


Fig. 6. Comparison of efficiency, CPU usage, and network bandwidth.

replication with the true Pareto fronts obtained by enumeration using the metric M_1^* [17]:

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

where: $Y', \bar{Y} \subseteq Y$ are the sets of objective vectors that corresponds to a set of pairwise nondominating decision vectors $X', \bar{X} \subseteq X$, respectively, and X corresponds to the decision variables of the problem. M_1^* gives the average distance to the Pareto optimal set (see [17] for further details).

In Table V we show the number of points obtained when we replicate the micro-GA and the M_1^* metric. It can be observed that the number of points found for all the problems is increased as a higher number of replicas is used, especially for the unconstrained problems Fonseca and Kursawe. This is

TABLE V
RESULTS OF THE GRID- μ GA.

Tasks		Fonseca	Kursawe	Osyczka2	Golinski
16	Points	1385	427	24	36
	M_1^*	6.6905e-6	6.5451e-4	7.3307e+1	3.7707e+2
32	Points	1970	677	37	37
	M_1^*	6.8230e-6	5.4077e-4	5.7351e+1	4.2092e+2
64	Points	2671	886	37	76
	M_1^*	5.9709e-6	5.1671e-4	6.2918e+1	4.2723e+2
128	Points	4109	1328	60	74
	M_1^*	5.6334e-6	5.9456e-4	4.5089e+1	2.6136e+2

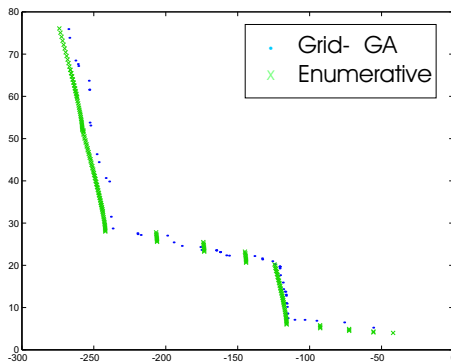


Fig. 7. Pareto fronts of the problem Osyczka2 generated by the enumerative algorithm and the grid- μ GA.

an expected result since we have carried out a more thorough exploration of the search space (we have merged a larger number of fronts).

If we focus on the M_1^* metric, the results show that the Pareto front obtained by replication of the unconstrained problems (Fonseca and Kursawe) are very close to the optimum Pareto front resulting from the enumerative search. Such accuracy does not exist for Osyczka2 and Golinski, since the grid- μ GA does not converge to the optimum Pareto front (see the high values of the metric), indicating that the micro-GA approach for constraint handling is not appropriate as its authors suggest [8].

VII. CONCLUSIONS AND FUTURE WORK

Enumerative search is a technique for solving multi-objective optimization problems based on evaluating each possible solution from a given finite search space. The technique allows to obtain the optimum Pareto front, but it does not scale as the search space becomes larger. In this context, Grid computing systems offer a strategic opportunity with a potentially large amount of computing power that can be used to overcome the drawbacks to some extent.

We have used the Globus Toolkit, a *de facto* standard system for Grid computing, to develop a distributed enumerative search algorithm, and we have used it to solve a benchmark composed of both constrained and unconstrained 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 the context

of a workstation cluster of 16 nodes. The results show that promising results are expected when solving unconstrained problems if they are computationally expensive; in the case of constrained problems, a more elaborated parallelization scheme is required, what constitutes a matter of future work. On the other hand, we have presented an approximation of using Globus to numerically improve a evolutionary algorithm.

Future research is in the line of using Globus in a Grid composed of hundred of computers as we did in [18], [19]. We also intend to apply the experiences obtained in this work to face the parallelization in the context of Grid computing of heuristic techniques for multi-objective optimization.

ACKNOWLEDGEMENT

This work has been partially funded by contracts TIC2002-04498-C05-02 and TIC2002-04309-C02-02.

REFERENCES

- [1] C.A. Coello, D.A. Van Veldhuizen, and G.B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2002.
- [2] C.A. Coello, "A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques," *Knowledge and Information Systems. An International Journal*, vol. 1, no. 3, pp. 269–308, 1999.
- [3] C.M. Fonseca and P.J. Fleming, "An Overview of Evolutionary Algorithms in Multiobjective Optimization," *Evolutionary Computation*, vol. 3, no. 1, pp. 1–16, 1995.
- [4] D.A. Van Veldhuizen and G.B. Lamont, "Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art," *Evolutionary Computation*, vol. 8, no. 2, pp. 125–147, 2000.
- [5] M. Baker, R. Buyya, and D. Laforenza, "Grids and Grid Technologies for Wide Area Distributed Computing," *Software: Practice and Experience*, vol. 32, pp. 1437–1466, 2002.
- [6] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, 1999.
- [7] I. Foster and C. Kesselman, "Globus: a Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.
- [8] C.A. Coello and G. Toscano, "Multiobjective Optimization using a Micro-Genetic Algorithm," in *GECCO-2001*, 2001, pp. 274–282.
- [9] A. Grama and V. Kumar, "State of the Art in Parallel Search Techniques for Discrete Optimization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, pp. 28–35, 1999.
- [10] A. Migdalas, P.M. Pardalos, and S. Stroy, *Parallel Computing in Optimization (Applied Optimization, Vol. 7)*, Kluwer, 1997.
- [11] E. Alba and M. Tomassini, "Parallelism and Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.
- [12] K. Deb, P. Zope, and A. Jain, "Distributed Computing of Pareto-Optimal Solutions Using Multi-Objective Evolutionary Algorithms," Tech. Rep. 2002008, KanGAL, September 2002.
- [13] D.A. Van Veldhuizen, J.B. Zydallis, and G.B. Lamont, "Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms," *IEEE Trans. on Evolutionary Computation*, vol. 8, no. 2, pp. 144–173, 2003.
- [14] D.A. Van Veldhuizen and G.B. Lamont, "Multiobjective Evolutionary Algorithm Test Suites," in *Proc. of the 1999 ACM Symp. on Applied Computing*, San Antonio, Texas, 1999, pp. 351–357.
- [15] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, 2001.
- [16] A. Kurpati, S. Azarm, and J. Wu, "Constraint Handling Improvements for Multi-Objective Genetic Algorithms," *Structural and Multidisciplinary Optimization*, vol. 23, no. 3, pp. 204–213, April 2002.
- [17] E. Zitzler, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000.
- [18] A.J. Nebro, F. Luna, and E. Alba, "Optimizaci3n Multi-Objetivo y Computaci3n Grid," in *MAEB'04 (In Spanish)*, 2004, pp. 365–372.
- [19] A.J. Nebro, F. Luna, and E. Alba, "Multi-Objective Optimization Using Grid Computing," Submitted for publication.