Soft Comput (2007) 11: 531–540
DOI 10.1007/s00500-006-0096-0

ORIGINAL PAPER

Antonio J. Nebro · Enrique Alba · Francisco Luna

Multi-objective optimization using grid computing

Published online: 3 May 2006

Abstract This paper analyzes some technical and practical issues concerning the use of parallel systems to solve multi-objective optimization problems using enumerative search. This technique constitutes a conceptually simple search strategy, and it is based on evaluating each possible solution from a given finite search space. The results obtained by enumeration are impractical for most computer platforms and researchers, but they exhibit a great interest because they can be used to be compared against the values obtained by stochastic techniques. We analyze here the use of a grid computing system to cope with the limits of enumerative search. After evaluating the performance of the sequential algorithm, we present, first, a parallel algorithm targeted to multiprocessor systems. Then, we design a distributed version prepared to be executed on a federation of geographically distributed computers known as a *computational grid*. Our conclusion is that this kind of systems can provide to the community with a large and precise set of Pareto fronts that would be otherwise unknown.

Keywords Multi-objective problem optimization · Enumerative search · Parallel computing · Grid computing

A. J. Nebro (✉)
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, E.T.S. Ingeniería Informática, Office 3.2.15 Campus de Teatinos, 29071 Malaga, Spain
E-mail: antonio@lcc.uma.es
Fax: +34-952131397

E. Alba
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, E.T.S. Ingeniería Informática, Office 3.2.12 Campus de Teatinos, 29071 Malaga, Spain

F. Luna
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, E.T.S. Ingeniería Informática, Office 3.3.4 Campus de Teatinos, 29071 Malaga, Spain

1 Introduction

A multi-objective optimization problem can be defined, as stated in [20], as the problem of finding a vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the 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 such a solution which would contain the values of all the objective functions acceptable to the designer.

Generally, multi-objective optimization does not restrict to find a unique single solution, but a set of solutions called *non-dominated solutions*. Each solution in this set is said to be a *Pareto optimum*, and when they are plotted in the objective space they are collectively known as the *Pareto front*. Obtaining the Pareto front of a given problem is the main goal of multi-objective optimization.

There are several techniques that can be used to obtain the Pareto front of a multi-objective optimization problem. They can be classified into three categories: enumerative, deterministic, and stochastic [8]. In recent years, stochastic methods have been widely studied; in particular, evolutionary algorithms have been investigated by many authors (see the surveys [7,11,24]). These methods do not guarantee to obtain the optimal solution, but they provide good solutions to a wide range of optimization problems which other deterministic methods find difficult. Enumerative search, which is deterministic but without employing any heuristics, is a conceptually simple search strategy, and it is based on evaluating each possible solution from a given finite search space. The drawback of this technique is that it is inherently inefficient and it can be computationally expensive and even prohibitive as the search space becomes larger.

Despite of their inconveniences, the results that can be obtained by using enumeration are of great interest to the multi-objective optimization research community, because they 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. Thus, scientific advances can clearly outcome from the comparison or evaluation of a new algorithm, which hardly contrasts with the present scenario of non standardized analyses.

In this paper, we address, first, the solution of a benchmark of multi-objective optimization problems in monoprocessor systems using an enumerative strategy. The problems are solved with a software application, written in C++, which have been specifically designed to easily accept new problems in the future. Second, we study parallel approaches that could be used to enhance the efficiency of enumerative search. We will focus mainly in multiprocessing and on the use of computational grids [5,6,13]. These are distributed systems composed of potentially thousands of computers that take advantage of the Internet infrastructure to provide a huge unique virtual supercomputer. Thus, these systems allow to attack problems that were considered as intractable a few years ago. Finally, we develop a distributed enumerative algorithm using Condor [22], a software system which allows to build distributed applications according to the approach of computational grids. Our Condor-based application is used to solve a number of multi-objective optimization problems that can be considered as intractable on monoprocessor systems.

The contributions of the paper can be summarized as follows:

- Solving a benchmark of multi-objective optimization problems in a monoprocessor system equipped with modern CPUs, allowing us to determine the base times. From there, we can gain an insight of the complexity of each problem to be solved in sequential and in parallel systems.
- We study parallel approaches that can be applied to solve hard problems by using enumeration. By providing parallel implementations we enhance the response time of the enumerative search. The use of a distributed enumerative algorithm based on grid computing concepts gives us a new perspective about the actual limits that we can find nowadays in order to decide which problems are solvable or not.
- The research community can take advantage of both the software we have developed and the results we have obtained (they can be found in the link `http://www.neo.lcc.uma.es/software/ESaM`). Thus, a worldwide common database of multi-objective optimization problems and their solutions provided by enumeration can be used as the basis for future comparison with other techniques.

The rest of this paper is organized as follows. In the next section, we discuss related work concerning the application of parallel computing to multi-objective problem optimization. Section 3 includes a description of the sequential algorithm, as well as its performance when solving a benchmark of problems. Section 4 describes the parallel implementations of the algorithm. Finally, we summarize and present some future work in Sect. 5.

2 Parallel computing and multi-objective optimization

Parallel computers have been widely used in the field of mono-objective optimization [15,18,23]. In the case of deterministic techniques, a typical example is the solution of optimization problems by means of parallel branch and bound algorithms [14]. The idea is, in general, to try to solve the problems more rapidly, or to solve more complex problems. In the context of stochastic methods, parallelism is not only a mean for solving problems more rapidly, but for developing more efficient models of search: a parallel stochastic algorithm can be more effective than a sequential one, even when the latter is executed on a single processor. For example, see [1,2] for surveys concerning evolutionary algorithms.

The advantages that offers parallel computing to mono-objective optimization can also hold in multi-objective optimization. Some works concerning evolutionary techniques are [8,10,25]. However, as stated in [8], few efforts have been devoted to parallel implementations in this field.

In general, most works on parallel computing and optimization are centered in two kinds of parallel systems: shared-memory multiprocessor systems and distributed systems, these lasts based on clusters or local area networks. 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 from 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, ATM), 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 (XML, SOAP, 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 lead to what it is called Grid computing [13], also known by several names such as metacomputing, Internet computing, or Web computing.

The idea of using hundreds or thousands of computers is very attractive because researchers can solve optimization problems that were previously considered as intractable. For example, in [4], an instance of the quadratic assignment problem (QAP) that would have required about 7 years of computation on a single workstation was solved in a week using a computational grid of over 2,500 processors. This problem was solved by using a distributed branch and bound algorithm. Apart from this work, few references can be found related to optimization and grid computing [19,21,26]; concerning multi-objective optimization and grid computing, we only know a related work [3], where grid technologies are used to parallelize a micro-genetic algorithm.

3 Sequential enumerative search

In this section we outline the sequential enumerative algorithm designed for this work, which is also the basis of the parallel implementations.

The algorithm is similar to the approach for finding non-dominated sets described in [9] (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 the evaluation and the dominance check. An outline of the algorithm is shown in Fig. 1.

The number of iterations carried out by the algorithm depends on the number of decision variables N and the desired granularity G, because the search space to be explored is G^N. 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 to compute each iteration is related to the number of objective functions (M) and their complexity. In addition, the evaluation step is only performed if the constraint test is passed, so the more restrictive the constraints the less the number of evaluations to compute. Besides, the constraint test can also take a significant amount of time. We analyze these issues in greater detail in Subsect. 3.3.

This algorithm has been implemented in C++, and designed with two goals in mind: first, it should be easy to incorporate new problems and, second, it must serve as the basis of parallel algorithms. The first goal is achieved by using the inheritance mechanism; there is a base class which must be inherited by each class representing a problem, so these classes share the same structure. To achieve the second goal (i.e., parallel extensibility), the program's main class includes, besides the problem to solve, additional parameters to explore the full search space or only a part of it. The first case is used in the sequential program, and the second one is applied in the parallel versions. In both cases, the program obtains the Pareto front (or a subfront if the search space is not fully explored) of the multi-objective problem. At the end of the computation, the values of the decision variables and the objective functions are stored into two files.

```
F[M] = {F1, F2, ..., FM} // Objective functions
R[C] = {R1, R2, ..., RC} // Constraints
x[N] = {x1, x2, ..., xN} // Vector of decision vars.
f[M] = {f1, f2, ..., fM} // Vector of 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 domination
    If f[j] is a non-dominated solution add f[j] to P
    Remove the solutions dominated by f[j] from P
```

Fig. 1 Pseudo-code of the sequential enumerative algorithm

3.1 Benchmark of multi-objective problems

We have selected several multi-objective problems from the specialized literature to measure the performance of the sequential implementation; in concrete, we have chosen problems from the book of Coello et al. [8]. The problems are named according to the terminology used in that book.

The selected problems are: Schaffer, Fonseca, Poloni, Kursawe, Viennet3, Deb, Viennet2, Bihn2, Viennet4, Tanaka, and Osyczka2. Additionally, we have include in our study Golinski's speed reducer problem [17]. The definition of these problems is included in Table 3, in the Appendix. This test suite covers a spectrum of both unconstrained and constrained problems, which are widely known and they have been used for comparison between algorithms.

3.2 Results

We have used a PC equipped with a Pentium 4 2.4 GHz processor and 512 MB of main memory; it runs Suse Linux 8.1 (Kernel 2.4.20). All the programs have been compiled with GCC `V.3.2`, with the compilation option `-O3`. In Table 1 we include the names of the problems, as well as their main features: number of decision variables, number of objective functions, and number of side-constraints. The last four columns contain the times required to solve the problems and the number of points of the Pareto front using 100 and 1,000 partitions per variable, respectively. To illustrate the influence of precision in the search, we include the two Pareto fronts obtained when solving Kursawe's problem in Fig. 2.

If we do not take into account the results of the problems Osyzcka2 and Golinski, for which the times can only be estimated because of their difficulty, we can observe in Table 1 that the times required to solve the first group of problems with 100 partitions per variable are around a few seconds in the slower cases. When we attack their solution with 1,000 partitions some of them require several hours, being Kursawe's problem the slower to solve (about 33 h). We show the Pareto fronts obtained in Figs. 6 and 7. Given these results, we conclude that all of them are solvable in a reasonable amount of time with a single processor.

Let us now examine the results of the last two problems. These are the most complex ones of the test suite, because they have the larger number of decision variables and constraints. As a consequence, the solution of these problems requires a considerable amount of time, even in the case of 100 partitions per variable. We include in Table 1 the estimated processor time to solve the problems with 100 partitions (more details about these estimations will be given in Subsect 4.2). Although these times are approximations, they

Table 1 Test suite: results of the sequential program (in seconds)

Problem	Variables	Functions	Constraints	Time (100)	Points (100)	Time (1,000)	Points (1,000)
Schaffer	1	2	0	< 1	9	< 1	73
Fonseca	3	2	0	7	43	61,466	434
Poloni	2	2	0	< 1	75	126	1,102
Kursawe	3	2	0	8	42	118,074	874
Viennet3	2	3	0	< 1	179	6,667	10,317
Deb	2	2	0	< 1	28	27	262
Viennet2	2	3	0	< 1	145	452	8,122
Bihn2	2	2	2	< 1	22	1	222
Viennet4	2	3	3	< 1	447	609	39,664
Tanaka	2	2	2	< 1	15	1	152
Osyczka2	6	2	6	90.2 days	570	N/A	N/A
Golinski	7	2	11	775.4 days	363	N/A	N/A

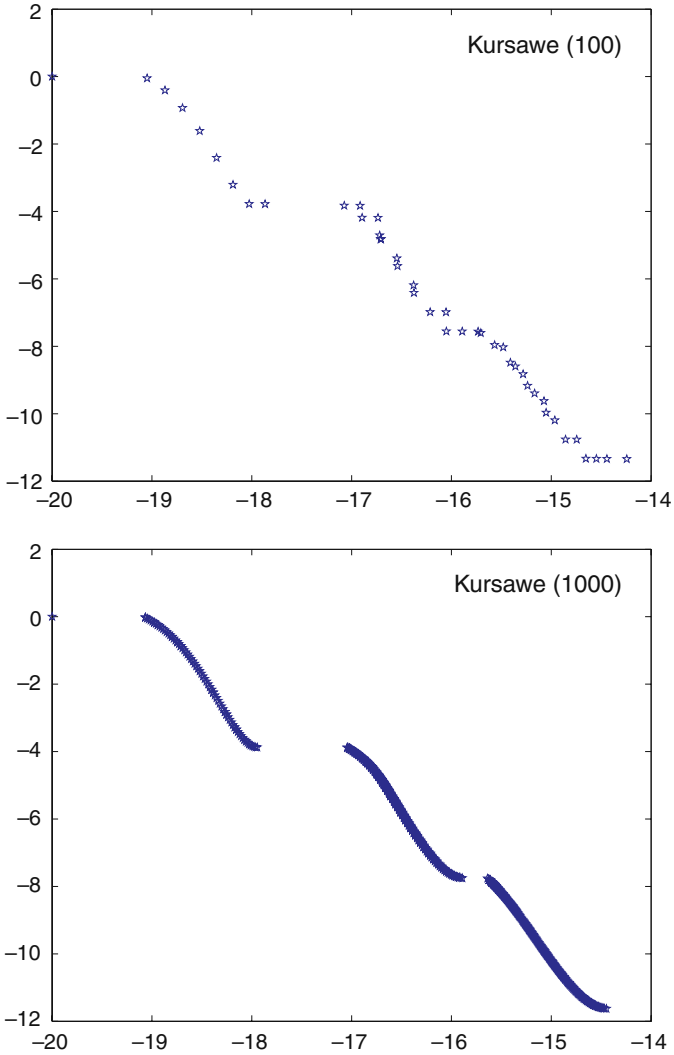

Fig. 2 Pareto fronts of Kursawe's problem (the values between parenthesis indicates the granularity)

give us an insight on the complexity of solving them by using a single monoprocessor computer.

These results clearly justify the interest in working on, first, the use of parallel systems to solve difficult problems and, second, the study of efficient heuristics when the enumerative search is infeasible.

Table 2 Results of the execution of Golinski's problem

Subtask	1	2	3	4	5
Total time	42.68	43.36	44.22	44.87	112.66
Constraints	26.74	27.21	27.55	27.71	37.53
Eval. & Dom.	15.94	16.15	16.67	17.16	75.13
Points	0	0	0	0	70

3.3 Analysis of the sequential program

In this subsection we present a quantitative analysis of the sequential program by studying the solution of Golinski's problem with 20 partitions per decision variable. We have chosen this problem for two reasons: first, it has a large number of constraints and, second, our experiments reveal that the points that conform its Pareto optima are located in a specific region of the search space.

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 use GNU's `gprof` tool to obtain profile information of the execution of the five subtasks. We show the results obtained in Table 2. For each subtask, we include the total running time (in seconds), the time consumed in evaluating the constraints, the time wasted in evaluating the functions and carrying out the dominance tests, and, finally, the number of points of the resulting sub-Pareto front found.

We observe that the first four subtasks do not locate any point of the Pareto front, what means that 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 (subtasks 1–4), the 62% of the time is spent in checking the constraints. However, the last subtask produces a front composed of 70 points, which correspond to the Pareto front of the problem (Fig. 3). Consequently, there is an increment in the total execution time, and the 66% 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.

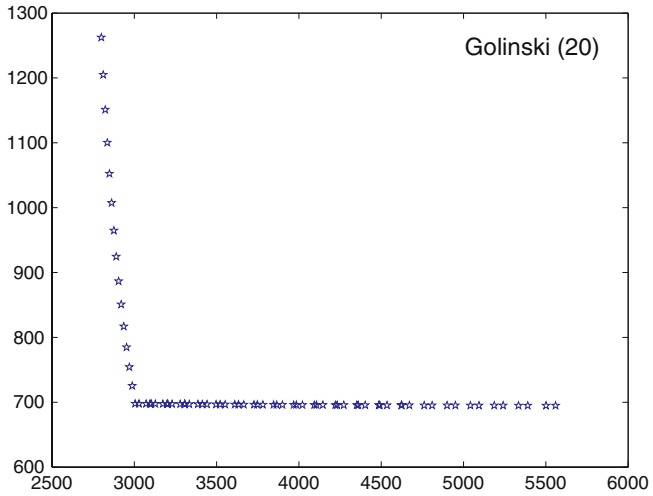

Fig. 3 Pareto front of Golinski's problem with 20 partitions per variable

We conclude then that the efficiency of the enumerative algorithm is heavily problem-dependent. 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 3). In problems without any constraints, 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.

4 Parallel approaches

Nowadays there are two main kinds of parallel computers: shared-memory multiprocessors and distributed systems. In this section we detail two parallel implementations of the enumerative search algorithm sketched in the previous section, one for each kind of parallel system.

4.1 Implementation for parallel shared-memory multiprocessors

The parallel algorithm we consider is based on the execution of several processes in parallel, each of them executing the sequential algorithm but exploring a different part of the search space. When each process finishes, it writes in two files the results it obtains (as explained in Sect. 3). Once these processes have finished, a new process reads these files and merges their contents applying the dominance test, thus obtaining the Pareto front. This parallel algorithm is simple, because inter-process communication is not necessary, and there is only a synchronization point.

To implement this algorithm in C++ we have two main options: threads and processes. With the use of threads in mind, the parallel algorithm would be even simpler, because it is not necessary to use files to store the results; instead, the sub-fronts can be stored in shared memory, and they could be directly accessed to obtain the Pareto front. However, we decided to use processes instead of threads for two reasons: first, a thread-based implementation provides no substantial advantages because of the low requirements concerning communication and synchronization in the algorithm; second, the multi-process implementation can be used without any changes in a distributed environment, as we will present in the next subsection.

In order to evaluate the efficiency of our multi-process program in a multiprocessor, we have ran the program in a Sun Ultra Enterprise 450, with four UltraSPARC II at 450 MHz processors and 4 GB of main memory. It runs Solaris 2.8, and we have used the G++ compiler `V.2.95`. As an example, we have solved Kursawe's problem with three decision variables and 200 partitions per variable. The sequential program takes 783 s, while the parallel one requires 196 s, leading to an almost perfect speed-up of 3.99.

We can conclude that parallel shared-memory implementations are simple and provide good performance. However, to solve difficult problems we still need systems with a higher number of processors, and this number is limited even when using a supercomputer. Therefore, if we need hundreds of processors, the best choice then is not using a multiprocessor, but using a distributed system.

4.2 Distributed application using condor

As discussed in Sect. 2, there are many issues concerning the development of parallel applications targeted to distributed systems. However, many of them fail when we intend to use a large amount of machines belonging to different owners or organizations (that is, a grid computer), because they do not cope well with some of the requirements of this kind of systems. Among these requirements we can consider heterogeneity, poor communication properties, unreliability, and dynamic availability. As an example, popular communication libraries such as PVM or MPI do not fulfill well the last two requirements, and many programs designed to use them fail to fulfill the second requirement. Grid computing is devoted to cope with these and other related issues, and it is an active research area in last years [5]. Thus, systems such as Globus [12], Legion [16], or Condor [22] are being used to run programs on grid computers composed of thousands of machines.

To implement our distributed enumerative search program, we have used Condor. Compared to other grid computing software, it is easy to install and to administrate, and existing programs do not need to be modified or re-compiled to be executed under Condor (they must only be re-linked with the Condor library). Condor is designed to manage distributed collections (pools) of processors spread among a campus or other organizations [22], having each machine an owner. A feature of Condor is that the owner of each machine can specify the conditions under which jobs are allowed to run; by default, a Condor job stops when a workstation's owner begins using the computer. Thus, Condor jobs use processor cycles that otherwise would be wasted. This way, users are not reluctant to donate their machines, and thus large Condor pools can be built.

Using Condor to develop a distributed version of our enumerative program has been an easy task, because we can re-use the same programs of the multi-process version proposed for shared-memory machines. We only need to obtain an executable version of the programs for each different kind of platform in the Condor pool, and we have to write a configuration file. This file contains, among other information, the name of the executable program, the number of instances of the program to be created, and the parameters to be passed to each instance. Then, we simply submit the jobs specified in the configuration file to Condor. As in the case of the multi-process program, when all the tasks have finished, we only have to collect the information in the result files to obtain the Pareto front.

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

- A cluster of four Digital AlphaServer 4100, each one with four Alpha processors at 300 MHz and 256 MB of RAM. The operating system of these machines is Digital Unix 4.0D.
- A cluster of 22 Sun Ultra 1 workstations. The machines have a UltraSPARC II processor at 400 MHz, 256 MB of RAM, and they execute Solaris 2.8.
- Two Sun Ultra Enterprise 450 servers, with four UltraSPARC II at 450 MHz processors and 4 GB of main memory. They run Solaris 2.8.
- A cluster of 16 PCs, each one with an Intel Pentium 4 processor at 2.4 GHz and 512 MB of RAM. They run Suse Linux 8.1 (Kernel 2.4.20).
- A cluster of 22 PCs, each one with a AMD Athlon XP 1.2 GHz processor and 256 MB of RAM. They run Debian Linux 3.0 (Kernel 2.2).
- A cluster of 20 PCs, each one with an Intel Pentium III at 600 MHz and 128 MB of RAM. The operating system is Suse Linux 8.1 (Kernel 2.4.20).
- Six desktop Intel-based PCs running several versions of Linux.

In total, we have been able to use up to 110 processors: 16 Alpha, 30 UltraSparc, 16 Pentium 4, 26 Pentium III, and 22 Athlon. To give the reader some information about the comparative performance of these processors, we have measured the time required by the first four kinds of machines in the above list to find the Pareto front of the problem Kursawe with 100 partitions per variable. The times obtained are 48, 50, 58 and 9 s, respectively.

We have used our Condor pool to solve the problems Osyczka2 and Golinski with 100 partitions per variable. Let us comment first the results of the problem Osyczka2 (see its Pareto front in Fig. 4). The problem was solved in less than 2 days (wall-clock time), while the total CPU time needed for such task has been reported by Condor of being 90.25 days. Given that the pool is composed of machines of different computing power, the time that would require the sequential program with the fastest processor will be below that amount of time, but in any case it would be around several tens of days. The problem Golinski was solved in 27 days, and the total CPU time reported by Condor was 775.4 days. Its Pareto front is included in Fig. 5.

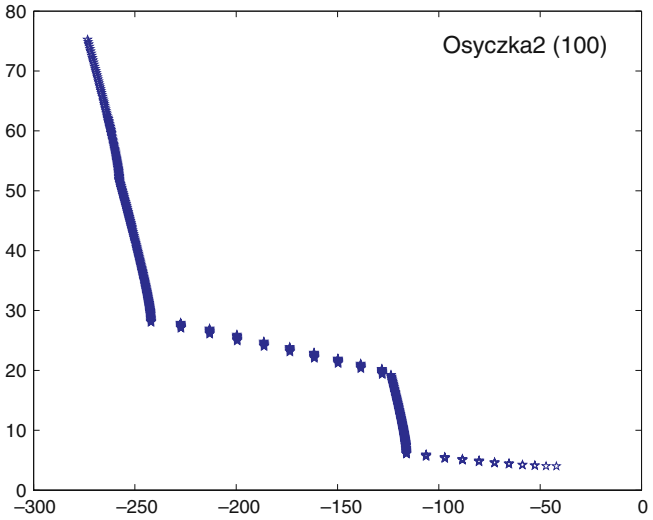

Fig. 4 Pareto front of the problem Osyczka2, with 100 partitions per variable

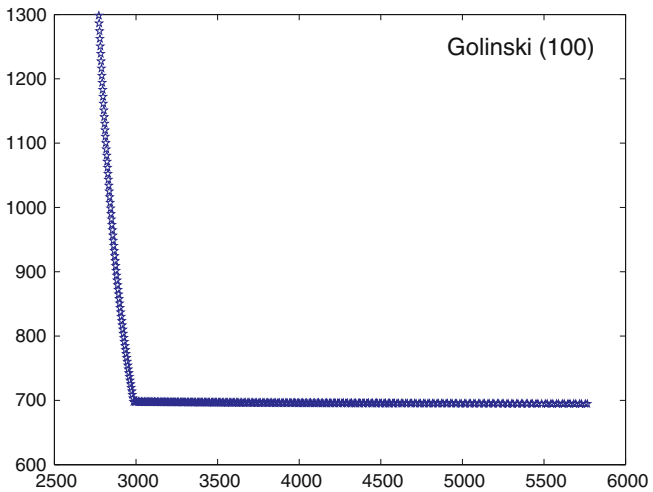

Fig. 5 Pareto front of the problem Golinski, with 100 partitions per variable

These results show the benefits of using grid computing technologies to solve problems that are infeasible to be computed in a monoprocessor system. We have used a pool of machines of a small size, but Condor allows to combine several pools of different organizations. Thus, building a grid computer with thousands of processors is nowadays possible. However, even when using such a system, the solution of problems such as Oszycka2 and Golinki with 1000 partitions per variable is intractable, so heuristic techniques are mandatory in these cases.

5 Conclusions and future work

In this paper, we have developed two parallel versions of an enumerative search algorithm for solving multi-objective

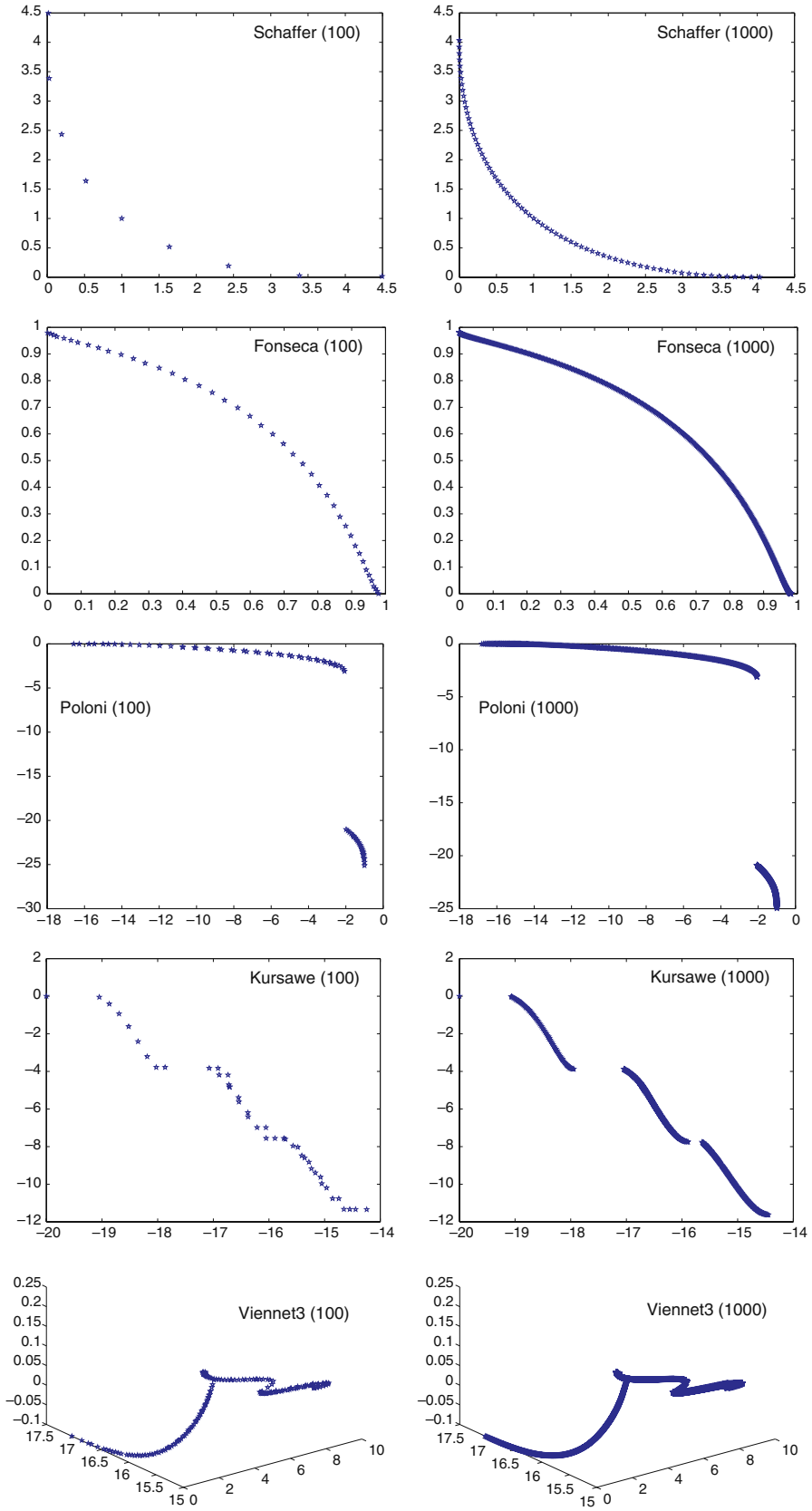

Fig. 6 Pareto fronts of the solved problems Schaffer, Fonseca, Poloni, Kursawe, and Viennet3 (the values between parenthesis indicate the precission)

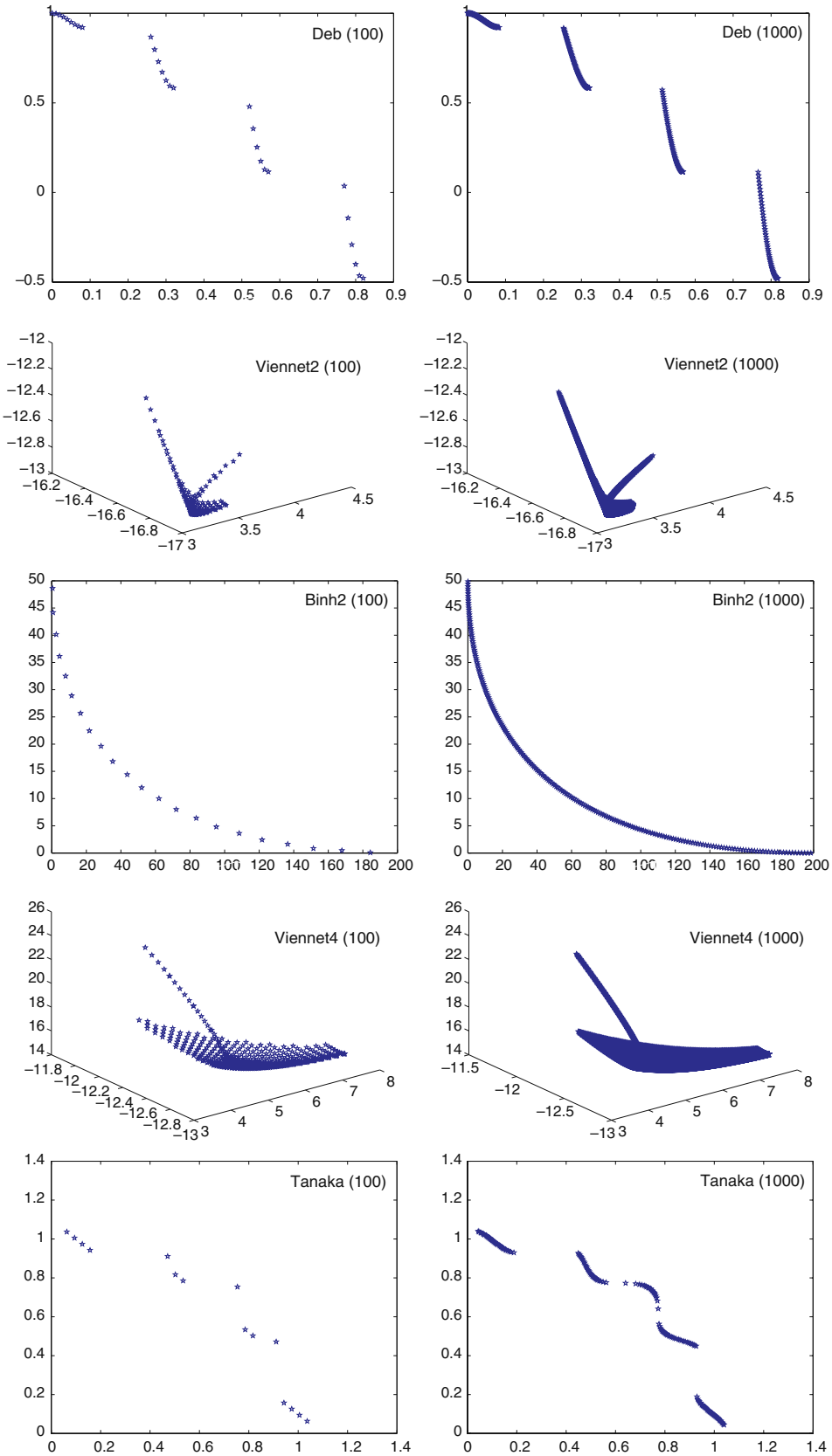

Fig. 7 Pareto fronts of problems Deb, Viennet2, Binh2, Viennet 4, and Tanaka (the values between parenthesis indicate the precission)

optimization problems: one suited to be used on shared-memory multiprocessors systems, and another one designed to run under Condor, a system that allows to build computational grids. Our main goal has been to analyze how these parallel systems can be used to obtain the Pareto front by enumeration of problems whose solution by using a single computer is infeasible.

We have found that the parallelization of the sequential enumerative algorithm is surprisingly easy. In fact, we use the same programs in the two parallel systems. The reason is twofold: first, there is no need of interprocess communication in the parallel enumerative algorithm and, second, the resulting program fits specially well to the requirements imposed by the Condor system.

Our experiments with the multiprocess version show that linear speed-ups can be obtained when using multiprocessors, what was an expected result because the processes are independent and they do not interact among them. The benefits of multiprocessors to solve hard problems depend directly on the number of processors, which is generally limited, even if we use a supercomputer; therefore, using a computational grid seems to be a more affordable and cost-effective solution to profit from a large number of processors. Our experiences using Condor to execute the distributed enumerative algorithm on a network of more than 100 processors indicate that it is possible to solve in a few days problems which would have otherwise required hundreds of days to be solved in a single computer.

We conclude that the multi-objective research community can take advantage of computational grids to solve difficult problems and to obtain the true Pareto fronts, thus allowing fair and meaningful exhaustive metrics, such as the distance to the Pareto front calculated with a grid computer.

As a future work, we intend to apply the experiences obtained in this work to the parallelization in the context of grid computing of heuristic techniques for multi-objective optimization.

Appendix: Test functions

Table 3 Test suite

Problem	Definition	Constraints
Schaffer	$\text{Min } F = (f_1(x), f_2(x))$ $f_1(x) = x^2$ $f_2(x) = (x-2)^2$	$-10^5 \leq x \leq 10^5$
Fonseca	$\text{Min } F = (f_1(\mathbf{x}), f_2(\mathbf{x}))$ $f_1(\mathbf{x}) = 1 - e^{-\sum_{i=1}^{n}(x_i - \frac{1}{\sqrt{n}})^2}$ $f_2(\mathbf{x}) = 1 - e^{-\sum_{i=1}^{n}(x_i + \frac{1}{\sqrt{n}})^2}$	$-4 \leq x_i \leq 4; i = 1, 2, 3$
Poloni	$\text{Max } F = (f_1(x, y), f_2(x, y))$ $f_1(x, y) = -[1 + (A_1 - B_1)^2 + (A_2 - B_2)^2]$ $f_2(x, y) = -[(x+3)^2 + (x+1)^2]$	$-\pi \leq x, y \leq \pi$ $A_1 = 0.5\sin 1 - 2\cos 1 + \sin 2 - 1.5\cos 2$ $A_2 = 1.5\sin 1 - \cos 1 + 2\sin 2 - 0.5\cos 2$ $A_3 = 0.5\sin x - 2\cos x + \sin y - 1.5\cos y$ $A_4 = 1.5\sin x - \cos x + 2\sin y - 0.5\cos y$
Kursawe	$\text{Min } F = (f_1(\mathbf{x}), f_2(\mathbf{x}))$ $f_1(\mathbf{x}) = \sum_{i=1}^{n-1}\left(-10e^{\left(-0.2*\sqrt{x_i^2 + x_{i+1}^2}\right)}\right)$ $f_2(\mathbf{x}) = \sum_{i=1}^{n}(\lvert x_i\rvert^a + 5\sin(x_i)^b)$	$-5 \leq x_i \leq 5$ $i = 1, 2, 3$ $a = 0.8,$ $b = 3$
Viennet3	$\text{Min } F = (f_1(x, y), f_2(x, y), f_3(x, y))$ $f_1(x, y) = 0.5 * (x^2 + y^2) + \sin(x^2 + y^2)$ $f_2(x, y) = \frac{(3x-2y+4)^2}{8} + \frac{(x-y+1)^2}{27} + 15$ $f_3(x, y) = \frac{1}{(x^2+y^2+1)} - 1.1e^{(-x^2-y^2)}$	$-3.0 \leq x, y \leq 3.0$
Deb	$\text{Min } F = (f_1(x, y), f_2(x, y))$ $f_1(x, y) = x$ $f_2(x, y) = (1 + 10y) * [1 - (\frac{x}{1+10y})^{\alpha} - \frac{x}{1+10y}\sin(2\pi q x)]$	$q = 4$ $\alpha = 2$ $0 \leq x, y \leq 1$
Viennet2	$\text{Min } F = (f_1(x, y), f_2(x, y), f_3(x, y))$ $f_1(x, y) = \frac{(x-2)^2}{2} + \frac{(y+1)^2}{13} + 3$ $f_2(x, y) = \frac{(x+y-3)^2}{36} + \frac{(-x+y+2)^2}{8} - 17$ $f_3(x, y) = \frac{(x+2y-1)^2}{175} + \frac{(2y-x)^2}{17} - 13$	$-4.0 \leq x, y \leq 4.0$
Bihn2	$\text{Min } F = (f_1(x, y), f_2(x, y))$ $f_1(x, y) = 4x^2 + 4y^2$ $f_2(x, y) = (x-5)^2 + (y-5)^2$	$0 \geq (x-5)^2 + y^2 - 25$ $0 \geq -(x-8)^2 - (y+3)^2 + 7.7$ $0 \leq x \leq 5$ $0 \leq y \leq 3$
Viennet4	$\text{Min } F = (f_1(x, y), f_2(x, y), f_3(x, y))$ $f_1(x, y) = \frac{(x-2)^2}{2} + \frac{(y+1)^2}{13} + 3$ $f_2(x, y) = \frac{(x+y-3)^2}{175} + \frac{(2y-x)^2}{17} - 13$ $f_3(x, y) = \frac{(3x-2y+4)^2}{8} + \frac{(x-y+1)^2}{27} + 15$	$y < -4x + 4$ $x > -1$ $y > x - 2$ $-4 \leq x, y \leq 4$
Tanaka	$\text{Min } F = (f_1(x, y), f_2(x, y))$ $f_1(x, y) = x$ $f_2(x, y) = y$	$0 \geq -x^2 - y^2 + 1 + a * \cos(b * \arctan(x/y))$ $0.5 \geq (x-0.5)^2 + (y-0.5)^2$ $a = 0.1$ $b = 16$ $-\pi \leq x, y \leq \pi$

Table 3 (Contd.)

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

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

References

1. Alba E, Tomassini M (2002) Parallelism and evolutionary algorithms. IEEE Trans Evol Comput 6(5):443–462
2. Alba E, Troya JM (1999) A survey of parallel distributed genetic algorithms. Complexity 4(4):31–52
3. Aloisio G, Blasi E, Cafaro M, Epicoco I, Fiore S, Mocavero S (2003) A grid environment for diesel engine chamber optimization. In: Proceedings of ParCo2003
4. Anstreicher K, Brixius N, Goux J-P, Linderoth J (2002) Solving large quadratic assignment problems on computational grids. Mathe Program 91:563–588
5. Baker M, Buyya R, Laforenza D (2002) Grids and grid technologies for wide area distributed computing. Soft Prac Exp 32:1437–1466
6. Berman F, Fox GC, Hey AJG (2003) Grid comptuing. making the global infrastructure a reality. In: Communications networking and distributed systems. Wiley New York
7. Coello CA (1999) A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowl Inf Sys Int J 1(3):269–308
8. Coello CA, Van Veldhuizen DA, Lamont GB (2002) Evolutionary algorithms for solving multi-objective problems. In: Genetic algorithms and evolutionary computation. Kluwer Dordrecht
9. Deb K (2001) Multi-objective optimization using evolutionary algorithms. Wiley, New York
10. Deb K, Zope P, Jain A (2002) Distributed computing of pareto-optimal solutions using multi-objective evolutionary algorithms. Technical Report 2002, KanGAL
11. Fonseca CM, Flemming PJ (1995) An overview of evolutionary algorithms in multiobjective optimization. Evol Comput 3(1):1–16
12. Foster I, Kesselman C (1997) Globus: a metacomputing infraestructure toolkit. Int J Supercomput Appl 11(2):115–128
13. Foster I, Kesselman C (1999) The grid: blueprint for a new computing infrastructure. Morgan-Kaufmann, San Francisco
14. Gendron B, Crainic TG (1994) Parallel branch and bound algorithms: survey and synthesis. Oper Res 42:1042–1066
15. Grama A, Kumar V (1999) State of the art in parallel search techniques for discrete optimization. IEEE Trans Knowl Data Eng 11(1):28–35
16. Grimshaw A, Wulf W (1997) The legion vision of a worldwide virtual computer. Commun ACM 40(1) 1995
17. Kurpati A, Azarm S, Wu J (2002) Constraint handling improvements for multi-objective genetic algorithms. Struct Multidisciplinary Optimization, 23(3):204–213
18. Migdalas A, Pardalos PM, Story S (1997) Parallel computing in optimization applied optimization, Vol. 7. Kluwer Dordrecht
19. Neary MO, Cappello P (2002) Advanced eager scheduling for Java-based adaptively parallel computing. In: Procedings of the ACM Java grande/ISCOPE conference, pp 56–65
20. Osyczka A (1985) Multicriteria optimization for engineering design. Academic, London
21. Tanimura Y, Hiroyasu T, Miki M, Aoi K (2002) The System for evolutionary computing on the computational grid. In: Parallel and distributed computing and systems (PDCS 2002)
22. Thain D, Tannenbaum T, Livny M (2002) Condor and the grid. In: Berman F, Fox G, Hey T (eds), Grid Computing: making the global infrastructure a reality. Wiley, New York
23. UEA CALMA Group (1995) CALMA project report 2.4: parallelism in combinatorial optimisation. Technical report, School of Information Systems, University of East Anglia, Norwch, UK
24. Van Veldhuizen DA, Lamont GB (2000) Multiobjective evolutionary algorithms: analyzing the state-of-the-art. Evol Comput 8(2):125–147
25. Van Veldhuizen DA, Zydallis JB, Lamont GB (2003) Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Trans Evol Comput 87(2):144 –173
26. Wright SJ (2000) Solving optimization problems on computational grids. Technical report, Mathematics and Computer Science Division, Argonne National Latoratory, Argonne, IL