Available online at www.sciencedirect.com
ScienceDirect
Parallel Computing 32 (2006) 415–440
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Efficient parallel LAN/WAN algorithms for optimization. The MALLBA project

E. Alba [c,*], F. Almeida [b], M. Blesa [a], C. Cotta [c], M. Díaz [c], I. Dorta [b], J. Gabarró [a], C. León [b], G. Luque [c], J. Petit [a], C. Rodríguez [b], A. Rojas [b], F. Xhafa [a]

[a] *ALBCOM, LSI, Universitat Politècnica de Catalunya, Campus Nord Ω, 08034 Barcelona, Spain*
[b] *EIOC, Universidad de La Laguna, Edificio Física/Matemáticas, 38271 La Laguna, Spain*
[c] *Dpto. de Lenguajes y Ciencias de la Computacion, LCC, Universidad de Málaga, E.T.S.I. Informática, Campus de Teatinos s/n, 29071 Málaga, Spain*

Received 15 December 2004; received in revised form 25 September 2005; accepted 30 June 2006

Abstract

The MALLBA project tackles the resolution of combinatorial optimization problems using generic algorithmic skeletons implemented in C++. A skeleton in the MALLBA library implements an optimization method in one of the three families of generic optimization techniques offered: exact, heuristic and hybrid. Moreover, for each of those methods, MALLBA provides three different implementations: sequential, parallel for Local Area Networks, and parallel for Wide Area Networks. This paper introduces the architecture of the MALLBA library, details some of the implemented skeletons, and offers computational results for some classical optimization problems to show the viability of our library. Among other conclusions, we claim that the design used to develop the optimization techniques included in the library is generic and efficient at the same time.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Combinatorial optimization; Software engineering; MALLBA library; Exact techniques; Metaheuristics; Hybridization; Local and wide area implementations; Parallel algorithms

1. Introduction

Combinatorial optimization problems arise naturally in many scientific areas such as Control Theory, Operations Research, Biology, Telecommunications and Computer Science. The MALLBA project is an effort to develop an integrated library of skeletons for combinatorial optimization including exact, heuristic and hybrid techniques. Sequential and parallel execution environments are supported in a user-friendly and, at the same time, efficient manner. Concerning parallel environments, both Local Area Networks (LANs) of workstations and Wide Area Networks (WANs) are considered.

Due to the growing park of available computers in the Internet, we include in our library the new scenario of WAN computing, in contrast to the traditional libraries targeted to parallel LAN environments. The WAN

* Corresponding author.
E-mail address: eat@lcc.uma.es (E. Alba).

0167-8191/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2006.06.007

environments have some characteristics different from LAN ones. The main features of WAN networks are: higher communication overhead, heterogeneity, unstable behavior of the links, security restrictions, and frequent faulty scenarios.

The three main features to highlight in the MALLBA library are the integration of all the skeletons under the same design principles, the facility to switch from sequential to parallel environments, and the cooperation among skeletons to provide new powerful hybrid skeletons.

In this work we describe the design, implementation and evaluation of a project that accomplishes all these goals. The resulting set of software techniques and numerical studies account for the results of the MALLBA project. The contributions of this paper are manifold. First, we test several algorithms provided by the MALLBA project. Second, we want to find out the expected and actual outcomes of solving optimization problems in LAN and WAN environments. Third, we are interested in showing significant results, and thus we include a mixed set of algorithms and optimization problems showing some difficulties usually found in real-world tasks. The current version of the MALLBA library works on clusters of PCs under Linux. However, from the software point of view, the architecture is flexible and extensible since new skeletons can be added, alternative communication layers can be used, etc. We refer the reader to [4] for a quick introduction into MALLBA.

Several tools offering parallel implementations for generic optimization techniques such as simulated annealing, branch-and-bound or genetic algorithms have been proposed in the past (see, e.g. [22,37,38,46]). Also, some existing frameworks, such as *Local++*, its successor *EasyLocal++* [27], *Bob++* [18], and the IBM COIN open source project [34] provide sequential and parallel generic implementations for some exact, heuristic and hybrid techniques (see [48] for a review). However, these frameworks and implementations do not integrate several optimization techniques together, and they do not offer the possibility of both sequential and parallel environments to coexist transparently together. Several frameworks, such as *OpenBeagle* [25] or DREAM project [6] have similar goals, although they are mainly oriented to evolutionary computation. *ParadisEO* [13] provides a more general framework for parallel metaheuristics and has an approach similar to the library that we present in this paper. Other libraries are out of this scope; for example *Sutherland* [39] and LEAP [23] are not parallel, while PISA [12] is too specific (multi-objective optimization).

To the best of our knowledge, there is no previous work considering all these optimization techniques and problems and, at the same time, extending their analysis to LAN and WAN environments.

The outline of this work is the following. We first present the architecture of MALLBA (Section 2) and its advantages for developing new algorithms and fast prototyping. In Sections 3–5 we provide working examples plus numerical and time analysis of exact, heuristic and hybrid algorithms, respectively; they all follow the basic MALLBA architecture for sequential, LAN and WAN platform execution. In Section 6 we summarize our work and present some conclusions and possible future work. Finally, we have added two appendixes to this paper, that list all the problems and pseudo-codes of the algorithms addressed in this study.

2. The MALLBA architecture

Concerning hardware, the MALLBA infrastructure is composed of computers and communication networks from the Universities of Málaga (UMA), La Laguna (ULL) and Barcelona (UPC), all of them in Spain. These three universities are connected through RedIRIS, the academic and scientific spanish computer network. This network is managed by CSIC (*Consejo Superior de Investigaciones Científicas*, the Spanish High Council of Scientific Research) that connects the main universities and research centers in Spain. RedIRIS is a WAN with ATM technology (ATM accesses of 34/155 Mbps). The current version of the MALLBA library works on clusters of PCs under Linux, which are located in each of the three participant universities.

Concerning software, all algorithms in the MALLBA library are implemented as *software skeletons* (similar to strategy pattern [26]) with a common internal and public interface. A skeleton is an algorithmic unit that, in a template-like manner, implements a generic algorithm. The algorithm will be made particular to solve a concrete problem by fulfilling the requirements specified in its interface. This permits fast prototyping and transparent access to parallel platforms. In the MALLBA library, every skeleton implements a resolution technique for optimization, taken from the fields of exact, heuristic and hybrid optimization.

All this software has been developed in C++. We chose this language because it provides a high-level oriented-object set of services and, at the same time, it generates efficient executable files what is an important

issue in this library. The skeleton design in MALLBA is based on the separation of two concepts: the features of the problem to be solved and the general resolution technique to be used. While the particular features related to the problem must be given by the user, the technique and the knowledge needed to parallelize the execution of the resolution technique is implemented in the skeleton itself, and is completely provided by the library. Thus the user does not program neither the resolution technique nor its parallelization. It is very common that the problem is represented by a complex function to be optimized and the details on how to manipulate tentative solutions (e.g. merge, cut, or interpret parts of them). Basically, the resolution technique is the algorithm defining the steps to proceed to the optimization of the problem. Almost every optimization technique exhibits a traditional three stage process, namely: (1) generating initial solutions (2) an improvement loop and (3) testing a stop condition. The way in which different skeletons do this work is really different and varied in the actual spectrum of optimization research.

Skeletons are implemented as a set of *required* and *provided* C++ classes which represent object abstractions of the entities participating in the resolution technique (see Fig. 1a). The *provided* classes implement internal aspects of the skeleton in a problem-independent way. We call this internal set of classes the *Solver part* of the skeleton. In general, for each algorithmic technique several sequential resolution patterns are provided, all of them grouped in the class `Solver_Seq` (for example, the iterative and recursive patterns showed in the figure). The parallel patterns are grouped in the classes `Solver_Lan` and `Solver_Wan`. In the figure are depicted resolutions patterns which uses the Master–Slave paradigm, independent runs and replicate data. Those classes are completely implemented and provided in the respective skeletons. The *required* classes specify information related to the problem. For the whole skeleton to work, it is required that these classes get completed with problem-dependent information. This conceptual separation allows us to define required classes with a fixed interface but without an implementation, so that provided classes can use required classes in a generic way. The fulfillment of the required classes would make the skeleton applicable to the problem specified.

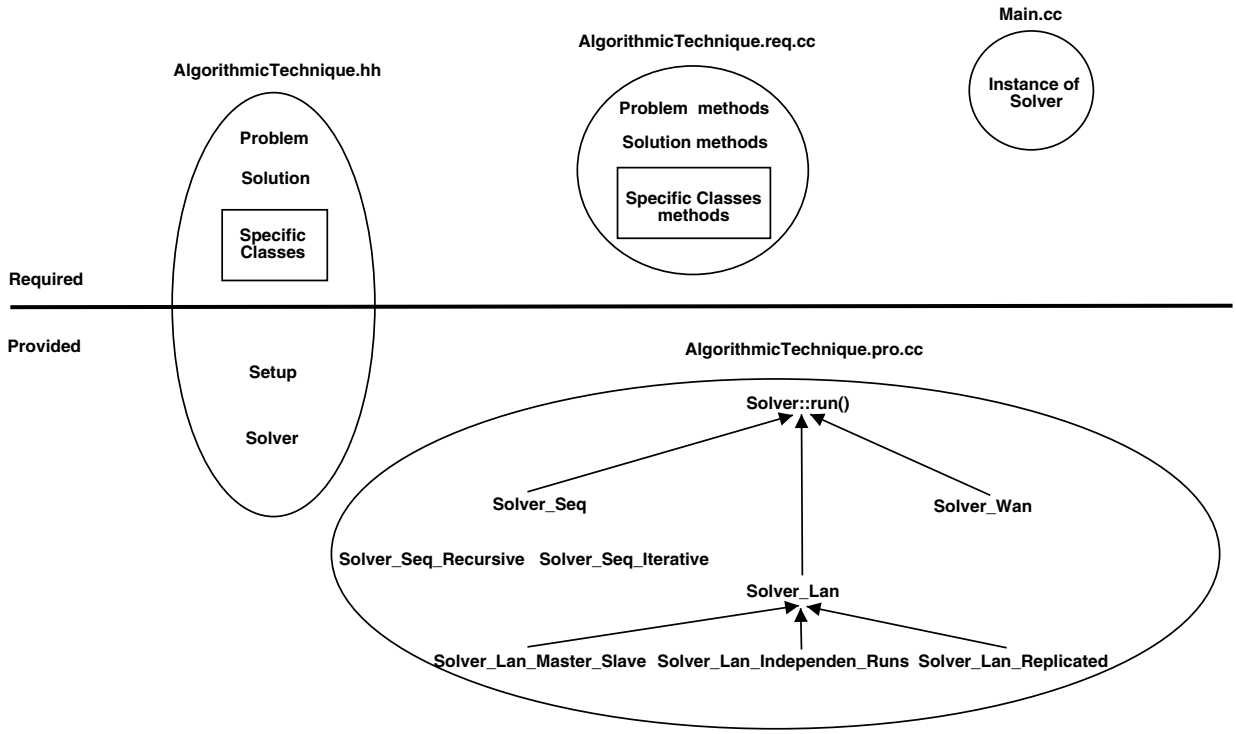
The fact that the user of a MALLBA skeleton only needs to implement the particular features of the problem to be solved, i.e., to fill in the required classes with an specific problem-dependent implementation, helps the creation of new programs with less effort.

Next, we will introduce and discuss the external interface of a skeleton. Two kinds of users work with this interface: the user who wants to instantiate a new problem, and the user who wants to implement a new skeleton and incorporate it to the MALLBA library. Also in the next subsections, we will discuss the communication and the hybridization interfaces. The aim is to explain firstly what a final user must consider, then what a skeleton programmer needs to know about the parallel issues and, finally, how to merge skeletons to yield new optimization procedures. Descriptions from these different points of view are needed since the potential users and researchers using the library will have a different level of interaction with our software (see Fig. 1b), depending on his goal (e.g., to instantiate a problem, to change the communication layer, or to create new skeletons for new techniques).

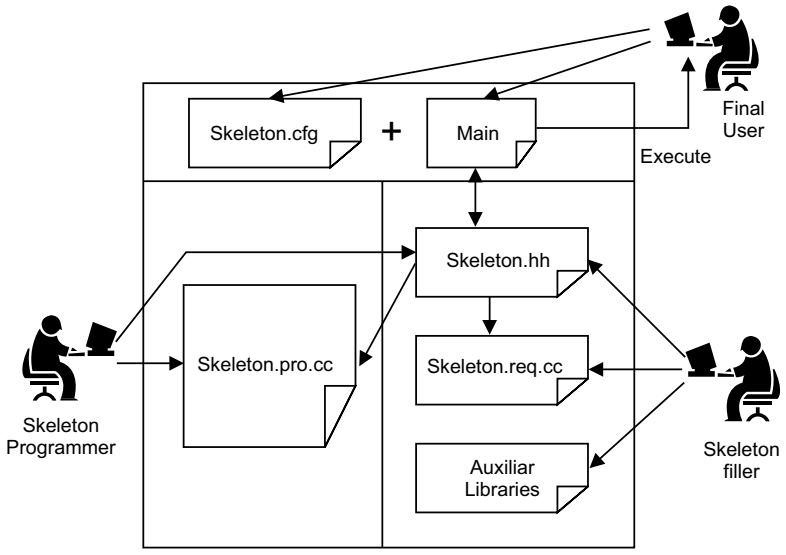
2.1. Skeleton interfaces

From the user's point of view, two major aspects must be considered: the problem to be solved, and the resolution technique to be used. The user will be responsible for adequately describing the former. As to the latter, rather complete descriptions are provided by the library. The user addresses these two aspects by selecting the skeleton and implementing its problem-dependent aspects. Later, since LAN and WAN implementations exist, the user will be able to execute the resulting program on sequential or parallel environments. Notice however that the same unique description made by the user will be used in any of the environments. Fig. 1b shows the interaction of different kinds of users (namely, final user, programmer and internal filler) with the different parts conforming a skeleton. Although there are three profiles, this does not mean that it is needed three different users to use MALLBA library. The same user can take care of the tasks at any profile level.

MALLBA already includes a large set of solvers ready for utilization. Extending them is quite direct, and creating new solvers is conceptually guided by the class hierarchy design. Parts of existing skeletons can be easily



(a) Architecture of a MALLBA skeleton. The horizontal line stands for the separation between the C++ classes the user must fulfill (upper part) and the classes that the skeleton already includes in a fully operational form (lower part).



(b) Interaction of the different user types and the files conforming a MALLBA skeleton. Usually, the skeleton filler and the final user are the same.

Fig. 1. MALLBA skeleton: structure and interaction.

reused to construct new ones. Each skeleton could have its own configuration file to avoid recompilation when parameters change.

Apart from some illustrative examples, MALLBA does not contain any complete implementation of specific problems. On the contrary, it provides the generic code the user has to customize. In this way, a single imple-

mentation – abstract yet efficient – can be reutilized in different contexts. The user do not need to have a deep knowledge about parallelism or distributed computing; these aspects are already included in the library. Let us get deeper in our understanding of the provided and required C++ classes conforming a skeleton.

Provided classes: they implement internal aspects of the skeleton in a problem-independent way. The most important *provided* classes are `Solver` (the algorithm) and `SetUpParams` (setup parameters). Provided classes are implemented in the files having the `.pro.cc` extension (see Fig. 1b).

Required classes: they specify information related to the problem. Each skeleton includes the `Problem` and `Solution` required classes that encapsulate the problem-dependent entities needed by the solver method. Depending on the skeleton other classes may be required. Required classes are implemented in the files having the `.req.cc` extension (see Fig. 1b).

2.2. Communication interface

Providing a parallel platform has been one of the central objectives in MALLBA. Local networks of computers are nowadays a very cheap and popular choice in laboratories and departments. Moreover, the available computational power of Internet is allowing the interconnection of these local networks, offering a plethora of possibilities for exploiting these resources.

To this end (i.e., using MALLBA onto a network of computers), it is necessary to have a communication mechanism allowing executing skeletons both in LAN and WAN. Since these skeletons are implemented in a high-level language, it is desirable for this communication mechanism to be also high-level; besides, some other network services could be needed in the future would be needed such as the management of parallel processes.

The needed set of services is generically termed *middleware*, and it is responsible for all basic communication facilities. Several steps were followed to construct this system: first, related existing systems were studied and evaluated; then, a service proposal was elaborated; finally, the middleware was implemented in C++.

The detailed review of existing tools included both systems based in the message-passing paradigm and systems for the execution and management of distributed objects and programs. We evaluated PVM, MPI, Java RMI, CORBA and Globus, as well as some other specific libraries [1]. Our main conclusion was the need for our own system, adapted to the necessities of our library, but based on an efficient standard, capable of being valid in the future.

Meeting all these criteria can be, almost exclusively, possible by using MPI as the base for developing a communication library. Efficiency was a major goal in this work, and hence this decision; besides, MPI (in both MPICH and LAM/MPI, the two well-known implementations of the standard) is becoming increasingly popular, and has been successfully integrated in new promising systems such as Globus.

Although there is no theoretical drawback in using MPI directly, we developed a light middleware layer termed `NetStream` (see Fig. 2). With this tool, a MALLBA programmer can avoid the large list of parameters and interact with the network in the form of *stream modificators*, that allows advanced input/output

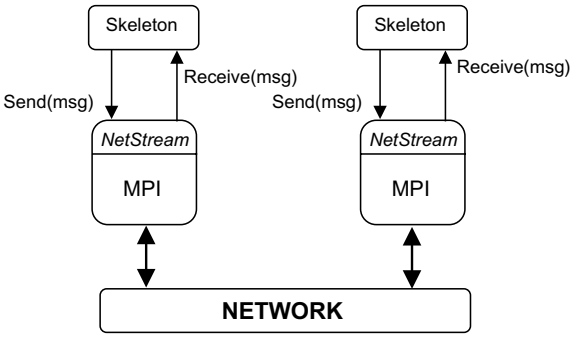


Fig. 2. NetStream communication layer on top of MPI.

operations to look like basic data exchanges with streams. By using << and >> operators the programmer can develop LAN and WAN skeletons by feeding data and net operations in a easy way.

Then, `NetStream` allows skeletons exchanging data structures efficiently, keeping a high abstraction level and ease of use. For this latter purpose, the number of parameters in the resulting methods has been minimized, and a large number of services has been implemented. These services can be classified into two groups: basic services, and advanced services. Among the basic ones we can mention:

- *Send-reception of primitive data types:* `int`, `double`, `char`, strings, etc. both in raw format and packed (for efficiency purposes when used on a WAN). This can be done using input/output streams from/to the network.
- *Synchronization services:* barriers, broadcasts, checking for pending messages, etc. As in the C++ standard, these services are available by means of manipulators, i.e., methods that alter the behavior of a stream, feeding it as if they were data.
- *Basic management of parallel processes:* querying a process ID or the number of processes, establishing and retrieving the IDs of processes at the ends of a stream, etc.
- *Miscellaneous:* starting and stopping the system (using static class methods (singleton pattern [26]) rather than instances methods), etc.

Among the advanced services implemented we can cite the following:

- *Management of groups of processes:* this allows skeletons to be arranged in parallel optimization regions. Available methods allow manipulating communicators and intercommunicators between groups, in the MPI sense. This organization could be important for certain distributed algorithms, especially in the case of hybrid algorithms.
- *Services to acquire the on-line state of the network:* this C++ methods are provided to allow working with a model of both communication links and the state of machines involved in the execution, all this under a real time basis, during the run of a skeleton. Basically, these services endow the skeleton programmer with C++ methods to check the delays in any link of the LAN or WAN for different packet sizes, plus the error rate (noise) in the link, and the load of a workstation in the net. Furthermore, independent clients in C, C++ and Java have been developed in addition to the mentioned one in order to make `NetStream` a stand alone communication layer for optimization and other applications at a minimum complexity and overhead.

All these services provide high-level programming and will ease taking on-line decisions in WAN algorithms, although we are still at the stage of developing ''intelligent'' algorithms that use this information to perform a more efficient search.

2.3. Hybridization interface

In this section we discuss the mechanisms available in MALLBA to foster combinations of skeletons in the quest for more efficient and accurate solvers. This raises the question of constructing efficient *hybrid algorithms*. In its broadest sense, hybridization [20] refers to the inclusion of problem-dependent knowledge in a general search algorithm in one of two ways [16]:

- *Strong hybridization*: problem knowledge is included as specific non-conventional problem-dependent representations and/or operators.
- *Weak hybridization*: several algorithms are combined in some manner to yield the new hybrid algorithm.

The term ''hybridization'' has been used with diverse meanings in different contexts. Here, we refer to the combination of different search algorithms (the so-called weak hybridization). As it has been shown in theory [49] and practice [20], hybridization is an essential mechanism for obtaining effective optimization algorithms for specific domains. For this reason, there exist in MALLBA several tools for building such hybrid skeletons.

This contrasts with other optimization libraries that let the programmer alone when building new algorithms from existing ones.

Due to the fact that the algorithmic skeletons will be reutilized and combined both by MALLBA end-users and by specialists, it is necessary to specify in a standard and unified fashion the way these skeletons can interact. For this reason, we propose the notion of a *skeleton state*. The state of skeleton is its connection point with the environment. By accessing this state, one can inspect the evolution of the search, and take decisions regarding future actions of the skeleton. For this latter reason, it is mandatory to have not only the means for inspecting the state, but also for modifying it on the fly. Thus, either a user or another skeleton can control the future direction of the search. This is done with independence of the actual implementation of the skeleton, a major advantage in any large-scale project.

The advantages of using a state is that combining skeletons has a low cost, despite the fact that uniformly defining the state is not trivial, and constitutes an open research topic [19]. Our proposal is articulated around the two basic classes we mentioned before: `StateVariable` and `StateCenter`. The `StateVariable` class allows defining and manipulating any information element within the algorithm skeleton. The latter is the connection point that provides access to the state itself.

On the basis of these classes, constructing a hybrid algorithm is very easy: one has to simply specify the behavior pattern by means of the appropriate manipulation of the states of the skeletons being combined. As an example of the flexibility of this model we have developed meta-algorithms that define the way in which n component skeletons interact each other. One simply has to specify the precise algorithm involved to instance this meta-algorithm to a concrete working hybrid skeleton; the behavior pattern is the same no matter which these component algorithms are. This philosophy of "make once instance many" can serve to produce different algorithms with the same underlying search pattern at a minimum cost.

3. Exact optimization techniques

This section is devoted to explain the structure of the exact skeletons provided in MALLBA, namely branch-and-bound and dynamic programming. Branch-and-bound and dynamic programming are two major enumerative methods used to solve combinatorial optimization problems. Both methods devise intensive searching on the state space representation and are, usually, very time consuming. We introduce the techniques using the problem definition established in Appendix A. Examples using the 0–1 Knapsack problem and the single resource allocation problem have been provided.

3.1. Branch-and-bound

During the branch-and-bound [32] computation, *subproblems* are continuously generated and tested. Given a *subproblem* Π_i, it can be decomposed into $\Pi_{i1}, \Pi_{i2}, \ldots, \Pi_{ik}$ by a branching operation where $S_i = \bigcup_{j=1}^{k} S_{ij}$. Thus any feasible solution $\sigma \in S_i$ belongs to some S_{ij} and conversely any $\sigma \in S_{ij}$ belongs to S_i. Let $\mathcal{Q}$ denote the set of *subproblems* currently generated. A *subproblem* $\Pi_i \in \mathcal{Q}$ that is neither decomposed or tested yet is called *alive*. The set of *alive subproblems* is denoted by $\mathcal{L}$. For each tested subproblem in $\mathcal{Q}$ its *lower bound* and *upper bound* are computed. The greatest lower bound obtained so far is called the *best solution value* and denoted by *bs*. The solution realizing *bs* is called the *best solution* and is stored in $\mathcal{T}$. Algorithm 1 in Appendix B shows the pseudo-code of the branch-and-bound algorithm for a maximization problem.

Fig. 3 shows the UML diagram of the classes that implement the branch-and-bound skeleton. The only required classes specific of this paradigm is the one which represents the subproblems described in the above paragraphs. Two different parallel implementations will be presented in the next subsection.

3.2. Dynamic programming

The underlying idea of dynamic programming (DP) [33] is to avoid duplicate calculations, usually by keeping a table of known results that fills up as the subproblems are solved. Branch-and-bound is a top-down method, when a problem is solved by branch-and-bound, it is divided into smaller and smaller subproblems as the algorithm progress. Dynamic programming on the other hand is a bottom-up technique. Usually it

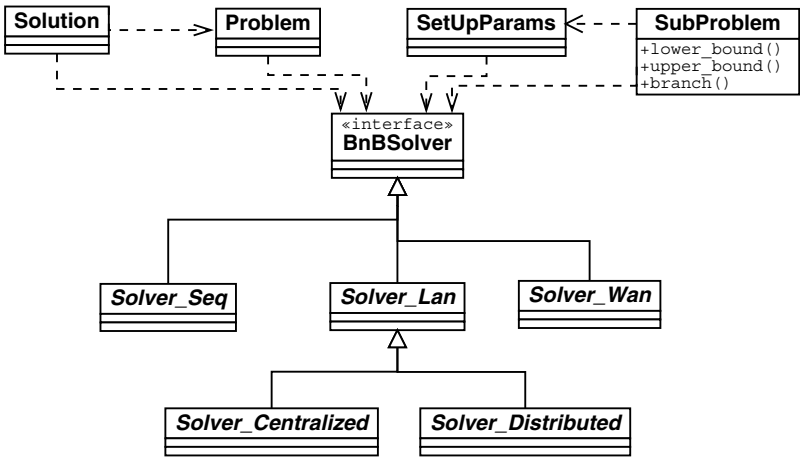


Fig. 3. UML diagram of the Branch-and-Bound skeleton.

starts with the smallest, and hence the simplest subproblems. By combining their solutions, the answers to subproblems of increasing size are obtained until finally the solution of the original instance is computed. Dynamic programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions. The method enumerates all decision sequences and pick out the best. The time and space requirements may be prohibitive. Dynamic programming often drastically reduces the amount of enumeration by avoiding the consideration of some decision sequences that cannot possibly be optimal. An optimal sequence of decisions is arrived by making explicit appeal to the principle of optimality. First, every subproblem that can be solved involving one decision $\{\Pi_{11},\ldots,\Pi_{1n(1)}\}$ is generated. Then the set of subproblems comprising two decisions $\{\Pi_{21},\ldots,\Pi_{2n(2)}\}$ is generated. To obtain the best solutions of the subproblems involving two decisions, is necessary the consideration of every feasible combination of subproblems involving one decision and to take the best decision on any subproblem. This approach admits a general recurrence formula where the optimal values for subproblems involving i decisions are computed in terms of subproblems involving $i-1$ decisions. Algorithm 2 in Appendix B shows a pseudo-code for a dynamic programming algorithm.

3.2.1. Parallel implementations

Concerning the branch-and-bound skeleton, two different parallel implementations are provided both of them based on the Master–Slave paradigm: one centralized and another distributed.

3.2.1.1. Centralized Master–Slave model. The Master has a local queue to store the generated subproblems. The Master extracts a subproblem from its queue and sends it to the first idle slave. The slave receives problems, best solutions, or an ending signal. If it receives a best solution value which improves the current one, then an update operation is performed. When a slave receives a problem, it calculates the new lower and the upper bounds. If the problem is not solved, then the slave branches it and sends the newly generated subproblems to the Master.

3.2.1.2. Distributed Master–Slave model. The Master sends the problems to the first idle slave and receives the slave request. The Master verifies the best value of the objective function has not been changed, and checks if there are free slaves to help in the search task. In this case it sends the number of idles slaves to the sender. When the number of idle slaves is equal to the initial value of slaves the search process finish. On the other hand, a slave works bounding the problem received. New subproblems are generated calling to the branch method. The slave asks for help to the master. If no free slaves are provided, the slave continues working locally. In other case, it removes subproblems from its local queue and sends them directly to other slave. The algorithm described is studied deeply in [21].

Concerning the dynamic programming skeleton, the MALLBA library provides a parallelization for multi-stage problems. The parallelization is obtained by assigning the computation of a column on the dynamic programming table (a *stage*) to a different processor to perform the computation in parallel. Since there are dependencies among consecutive *stages*, the parallel execution follows a pipeline scheme where the processors are activated as the computation in the former stages proceeds. The evaluation of a *state* by a processor involves the reception of information computed in previous processors in the pipe. After a *state* has been computed, it is sent to the next processor in the pipe. Typically, in a dynamic programming computation the number of stages involved is in the order of hundreds or even thousands. However, in practice, the number of physical processors available is not so big. Several strategies have been proposed to distribute the set of stages among the current set of processors [29]. The MALLBA engine for the parallel dynamic programming skeleton develops a cyclic assignment of the stages to the processors following a round robin scheme. The size of the buffer can be used to tune the application according to the characteristics of the network platform. A known drawback of the dynamic programming technique arises when the size of the problem is increased. The resulting algorithm may be very time and space consuming. The parallel approximation used not only reduces the running time of the sequential algorithm but it distributes the table amongst the set of processors also. This contributes to increase the size of the problem to be solved.

3.2.2. Instantiation examples and results

3.2.2.1. Branch-and-bound for the 0–1 Knapsack problem. The algorithm for the resolution of the classic Knapsack 0/1 problem described in Appendix A has been implemented using the branch-and-bound skeleton. In this section we analyze the experimental behavior for this algorithm on several sets of randomly generated test problems. Since the difficulty of such problems is affected by the correlation between profits and weights, we considered the strongly correlated ones. The experiments have been done on an heterogeneous cluster of PCs, which was configured with four 800 MHz AMD Duron Processors, seven 500 MHz AMD-K6 3D processors, 256 MB of memory each. The software used was Debian Linux version 2.2.19 (herbert@gondolin), the C++ compiler was GNU gcc version 2.7.2.3 and the MPICH version was 1.2.0.

Table 1 shows the speedup results of 10 executions of a randomly generated problem instance of size 1000. The first column contains the number of processors. The second column shows the average time in seconds. The column labeled `Speedup-Av` presents the speedup for the average times. The third and sixth columns give the minimum times (seconds) and the associated speedup, whereas the fourth and seventh columns show the maximum. The reason for the limited performance for a two processor system is that the Master/Slave paradigm is not suitable for a scenario where there is only a single slave. In this case, one of the processors is the Master and the others are the workers and furthermore, communications and work are not overlapped. Notice that there is no increase of the speed up with more than three processors, this is due to the problem has not enough grain and the inclusion of processors do not help to increase the performance only add overhead to it.

3.2.2.2. Dynamic programming for the resource allocation problem. We have instantiated the dynamic programming MALLBA skeleton for the single resource allocation problem described in Appendix A. We analyze the performance of the skeleton on a series of randomly generated problems. The serial algorithm considers

Table 1
Linux-PC cluster. Speedups for 10 executions of an instance of the 0/1 Knapsack problem randomly generated. The number of objects is 1000. The pure sequential time is 356.54

Procs	Average	Min	Max	SpeedUP-Av	SpeedUp-Max	SpeedUp-Min
2	686.86	680.88	712.57	0.52	0.52	0.5
3	223.86	186.62	284.02	1.59	1.91	1.26
4	166.11	147.94	192.67	2.15	2.41	1.85
5	171.91	151.48	185.03	2.07	2.35	1.93
6	165.79	140.06	205.79	2.15	2.55	1.73
7	158.16	146.15	174.89	2.25	2.44	2.04

Table 2
Results for the resource allocation problem using the DP skeleton, over a network of 13 PCs (4 AMD K6 750 MHz and 9 AMD K6 500 MHz) connected through a Fast Ethernet network

Stages	Seq. time (s) on fastest machine	SpeedUp			
States		2 procs. 750 MHz	4 procs. 750 MHz	4 procs. 750 MHz 4 procs. 500 MHz	4 procs. 750 MHz 9 procs. 500 MHz
1000–2000	457.79	1.97	3.92	4.12	6.01
1000–2500	714.87	1.98	3.94	4.30	6.02
1000–4000	1828.22	1.99	3.96	4.31	6.41
1000–5000	2854.04	1.99	3.97	4.24	6.42
1000–7000	5594.74	1.99	3.97	4.22	6.41
1000–10,000	11,422.60	1.98	3.97	4.18	6.38

instances with a fixed number of tasks ($N = 1000$) and different units of resources ($M = 2000$, $2500, 4000, 5000, 7000, 10{,}000$). Table 2 shows the speedup obtained from an instantiation of the dynamic programming skeleton for the resource allocation problem. We consider an heterogeneous cluster of 13 PCs, four of the processors are faster than the others. The parallel code shows a good scalability until four processors. Between four and eight processors performance decreases due to the slower machines introduced, however, it remains increasing when introducing more processors.

4. Heuristic optimization techniques

In this section we explain some heuristic techniques included in the MALLBA library. We consider two different families of heuristics: local search based methods and population based methods. These two families cover most of the spectrum of heuristics existing nowadays.

4.1. LS-based heuristic techniques

We address now the design and implementation of skeletons for several local search heuristics [41]. Local search (LS) heuristics make use of a basic local search method that iteratively improve a feasible solution by applying elementary perturbations or *moves* to it. In order to apply local search heuristics to a combinatorial optimization problem Π with set of feasible solutions S and cost function f (see Appendix A for a formal definition), any feasible solution $s \in S$ is associated with a set of neighborhood solutions $N(s)$. A solution s' is in the neighborhood of solution s if it can be obtained from s by applying a move to it. For example, if the solution is a permutation and the move the swap of two items in the permutation then the neighborhood of a permutation is the set of permutations obtained from it by swapping any two items. Note that, for efficiency reasons, the neighborhood of a solution s is usually maintained as the set of all elementary moves applied to s. The basic local search procedure can be described as in Algorithm 3 in Appendix B. Observe that this is a generic algorithm that must be particularized.

Starting from the basic local search, different local search heuristics have been defined by introducing other features such as *acceptance criteria* for a candidate solution, *stopping condition* for the procedure, *criteria to avoid cycling* between local optima, etc.

We consider the following local search heuristics: hill climbing, metropolis, simulated annealing (SA) and tabu search (TS). Notice that the first three heuristics generalize the local search method by introducing new acceptance criteria while the latter heuristic introduces criteria to avoid falling into local optima as well as new stopping condition criteria.

4.1.1. Hill climbing

Hill climbing heuristic is the simplest local search method. It only accepts solutions that improve the cost function. The heuristic ends in a locally optimal solution.

4.1.2. Metropolis

This local search heuristic was proposed by Metropolis et al. [40]. The basic idea is to accept, additionally, with low probability, solutions that worsen the objective function. To this end, the heuristic is parameterized by a temperature t where a move producing a gain $\delta = f(s') - f(s)$ in the cost function is accepted with probability $\min\{1, \exp(-\delta/t)\}$. Observe that hill climbing is a particular case of Metropolis by setting $t = 0$.

4.1.3. Simulated annealing

Simulated annealing (SA), proposed by Kirkpatrick et al. [36], is a generalization of the Metropolis heuristic. Indeed, simulated annealing consists of a sequence of executions of Metropolis with a progressive decrement of the temperature starting from a high temperature, where almost any move is accepted, to a low temperature, where the search resembles hill climbing. In fact, it can be seen as a hill-climber with an internal mechanism to escape local optima (see a pseudo-code in Algorithm 4 in Appendix B). In SA, the solution s' is accepted as the new current solution if $\delta \leqslant 0$ holds, where $\delta = f(s') - f(s)$. To allow escaping from a local optimum, moves that increase the energy function are accepted with a decreasing probability $\exp(-\delta/T)$ if $\delta > 0$, where T is a parameter called the "temperature". The decreasing values of T are controlled by a *cooling schedule*, which specifies the temperature values at each stage of the algorithm, what represents an important decision for its application (a typical option is to use a proportional method, like $T_k = \alpha \cdot T_{k-1}$). Simulated annealing usually gives better results in practice, but uses to be very slow. The most striking difficulty in applying SA is to choose and tune its parameters such as initial and final temperature, decrement of the temperature (cooling schedule), equilibrium detection, etc.

4.1.4. Tabu search

The tabu search heuristic [28] maintains some historical information related to the search process, that is, history on already visited solutions. This information is later used to guide the search in order to avoid cycling. Once a move is applied to a solution, it is considered *tabu* for some time, that means, it cannot be applied again during the exploration process for that time, unless it satisfies certain *aspiration condition*. Hence, a list of already applied moves, called the *tabu list*, is kept; before applying a move, the algorithm checks whether the move is tabu or not. Typically there are two kinds of tabu lists: the first, called long term memory, maintains history on the whole exploration process and, the second, called short term memory, maintains the history on most recently visited solutions. Additionally, the tabu search method incorporates two other features: *intensification* and *diversification*. Intensification is used to concentrate the search in regions of the solution space when there is evidence that the region contains good solutions and the latter is used to escape from local optima, that is, when the search gets trapped in a local optimum.

Regarding the stopping conditions, apart from standard stopping conditions, tabu search introduces other criteria based on the neighborhood structure. Note that by forbidding tabu moves from being applied, the neighborhood structure changes dynamically in the course of exploration process. Hence, we may derive other related stopping conditions. Fig. 4 depicts the UML diagram of the tabu search skeleton. The specific classes of this algorithm are `Movement` and `TabuStorage`, which were described in the precedent paragraphs. Also, there are several specializations of the `Solver_Lan` class which will be presented in Section 4.3.

4.2. Population-based heuristic techniques

Apart from LS-based heuristics, the MALLBA library also includes population-based heuristics. These techniques are inspired by the genetic mechanisms of natural species evolution, including the important concept of *population*. Three main natural processes are performed in a population to evolve: selection, mutation, and recombination of its individuals.

4.2.1. Evolutionary algorithms

Evolutionary algorithms (broadly called EAs) are stochastic search techniques that have been successfully applied in many real and complex applications (epistatic, multi-modal, multi-objective and highly constrained problems). Their success in solving difficult optimization tasks has promoted the research in the field known as *evolutionary computing* (EC) [7]. An EA is an iterative technique that applies stochastic operators on a pool of

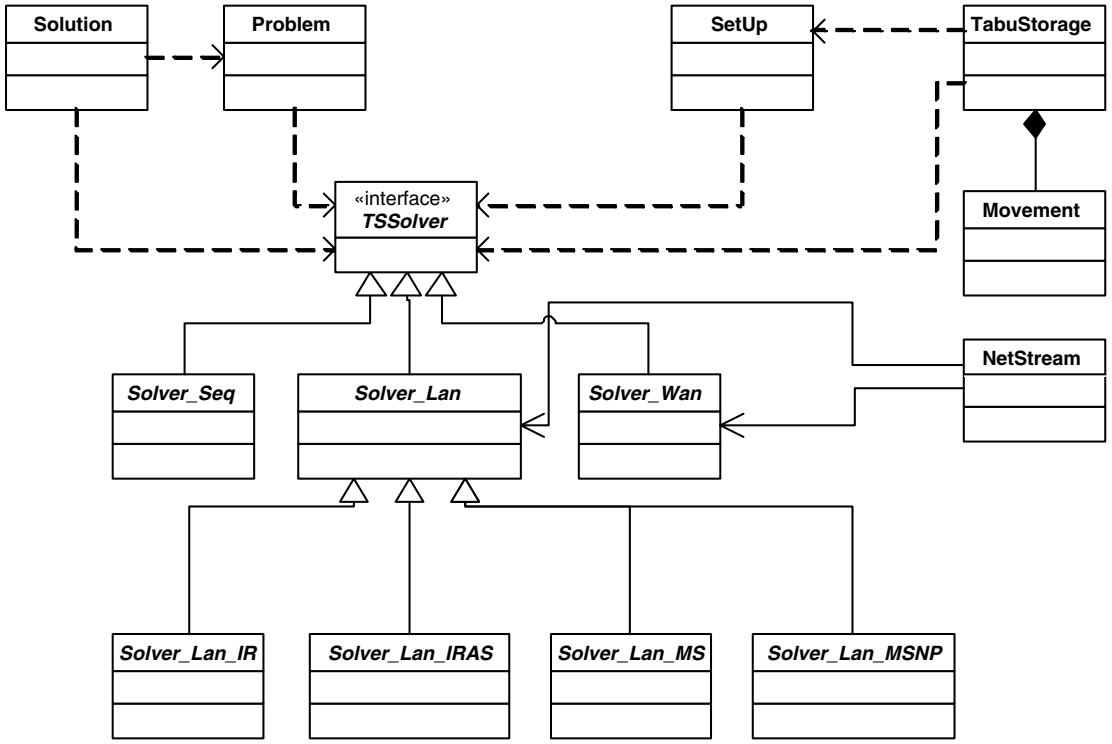


Fig. 4. UML diagram of the Tabu Search skeleton.

individuals (the population) (see Algorithm 5). Every individual in the population is the encoded version of a tentative solution. Initially, this population is generated randomly. An evaluation function associates a fitness value to every individual indicating its suitability to the problem.

The MALLBA library also includes genetic algorithms (GAs) [31]. GAs are a very popular class of EAs. Traditionally, GAs are associated to the use of a binary representation, but nowadays GAs use other types of representations too. A GA usually applies a recombination operator on two solutions, plus a mutation operator that randomly modifies the individual contents to promote diversity.

A CHC (Cross generational, elitist selection, Heterogeneous recombination and Cataclysmic mutation) [24] is a non-traditional GA which combines a conservative selection strategy (that always preserves the best individuals found so far) with a highly disruptive recombination (called *HUX*) that produces offsprings that are maximally different from their two parents. The traditional thought of preferring a recombination operator with a low disrupting properties may not hold when such a conservative selection strategy is used. On the contrary, certain highly disruptive crossover operator provide more effective search in many problems, which represents the core idea behind the CHC search technique (see Algorithm 6 in Appendix B). This algorithm introduce a bias against mating individuals who are too similar (*incest prevention*). Mutation is not performed, instead, a *restart* process re-introduces diversity whenever convergence is detected.

Another EA family we have studied in this work is an evolution strategy (ES, see Algorithm 7) [44]. This algorithm is very well suited for continuous optimization. In ES, the individual is made of the objective variables plus some other parameters guiding the search. Thus, a ES facilitates a kind of *self-adaption* by evolving the problem variables as well as the strategy parameters at the same time. Hence, the parameterization of an ES is highly customizable.

4.3. Parallel implementations

Finding good solution to combinatorial optimization problems by local search heuristics usually requires a considerable amount of computation time. Specifically, simulated annealing and tabu search are time

consuming heuristics; SA needs many iterations to converge to a good solution and TS needs extra time to efficiently manage the tabu list as well as to intensify and diversify the search. Parallelism can help with this respect, at least, in two directions: reducing the computation time and providing a better exploration of solution space.

4.3.1. Parallelization of LS-based heuristics

Local search heuristics can be parallelized in different ways [17], usually classified in: (a) low-level parallelism, that is, parallelization of computation within an iteration of the search, for example computing in parallel the cost function. This kind of parallelism aims to speed up the computations but the search process, and consequently, the solution found is as in the sequential case; (b) domain decomposition, that is, we try to split the original problem or its subproblems into smaller problems solved in parallel; for example, if we could partition the neighborhood of a solution into different parts then exploring them could be done in parallel by different processors; (c) multi-thread searches (or independent runs), that is, we compute in parallel the same task by different threads; for example, we may explore the same neighborhood by different threads and since the exploration is not deterministic, we are exploring different search paths leading to a better solution. Next, we may consider different degrees of synchronism/asynchronism yielding to other possibilities of performing the parallel computations.

Different parallel models can be considered to accomplish the parallel strategies mentioned above. We have considered two basic models: independent runs (IR) and Master–Slave (MS) and two implementations for each of them, namely, IR and IR with Autonomous Strategies, and MS and MS with neighborhood partition as explained next. In Fig. 4 we depicted how those parallel implementations of the core of the tabu search method are represented by classes and how they relate each other.

4.3.1.1. Independent runs model. The independent runs model (IR) consists of simultaneous and independent executions of the same program. In this model, there is a processor doing the coordination task that consists in, at the beginning, sending the problem instance as well as the values for the parameters to the rest of processors and receiving the results upon termination of all the processors execution. At the end, the coordinator processor computes the best solution and may show other relevant statistics. In this model, each processor runs the same instance of the program on the same input data and the communication time is almost irrelevant. Observe that this model makes sense as far as the program is non-deterministic, which is precisely the case of meta-heuristic implementations that take random decisions. Note that running the same implementation in different processors usually leads to exploring different areas of the solution space via different search paths. In general, running the parallel IR implementation on p processors is essentially equivalent to running the program p times sequentially since the overhead due to the parallelism (distributing the input and recollecting the results) is very small.

4.3.1.2. Independent runs with autonomous strategies. The independent runs with autonomous strategies model (IRAS) is a generalization of the IR. In the IRAS model, a processor is given, additionally, a strategy to be used for its own search. A strategy consists of an initial solution and values for parameters that control the algorithm. Now, the coordinator processor, at the beginning, sends to any processor a strategy and the problem instance and receives the results upon termination of all the processors execution. Again, at the end, the coordinator processor computes the best solution, the best corresponding strategy and may show other relevant statistics.

4.3.1.3. Master–Slave model. In the Master–Slave model (MS) there are two distinguished types of processors: a master processor and slave processors. The control is performed by the master and the slaves are subordinated to it. The master processor spawns slaves processors, initializes them, assigns subtasks and collects their results. Then, it computes a result from the results obtained by the slaves and uses it for its own work and so on. In our case, the master processor runs the main algorithm and uses slaves to choose the best movement that leads to the best solution in the neighborhood of the current one. To this end, each slave processor explores the neighborhood by its own and comes up with its best movement. Clearly, the task of exploring the neighborhood in parallel makes sense as far as the neighborhood exploration is not deterministic.

4.3.1.4. Master Slave with neighborhood partition. The Master Slave with neighborhood partition model (MSNP) is derived from the MS model by specifying the type of the task accomplished by the slave processors. In contrast to the MS, in this model each processor explores just a portion of the neighborhood. Thus, through this model, we can reduce the time needed to perform a complete neighborhood exploration.

4.3.2. Parallelization of population-based heuristics

Parallelism arise naturally when dealing with populations, since each of the individuals belonging to it is an independent unit [14]. Due to this, the performance of population-based algorithms is specially improved when running in parallel. Two parallelizing strategies are specially focused on population-based algorithms: (1) parallelization of computation, in which the operations commonly applied to each of the individuals are performed in parallel; and (2) parallelization of population, in which the population is split in different parts that can be simply exchanged or evolve separately and be joined later. MALLBA implements two well-known models applying this last strategy: the split and the island model.

4.3.2.1. Split model. In the split model the master process does not evolve a population through a generational loop but perform successive send/receive operations of parts of the population to its slaves. The slave processes receive their part of the population from a unique origin (the master), evolve it through a generational loop and after a few loops they send the resulting population back to the master. The master builds the next population from the parts received from the slaves.

4.3.2.2. Island model. In this model, the role of the master process is only to initialize the slaves, to let them know their neighbors processes and collect the post-execution informations before the slaves stop. Each slave process (or island) evolves a whole population and this includes to exchange, from time to time, parts of its population with its neighbors.

Other models have also been applied to parallelize the population-based algorithms included in the MALLBA library. We detail them in Section 5 since they are specially related and oriented to the hybrid algorithms we propose.

4.4. Instantiation examples and results

In the following we explain more in detail how the tabu search skeleton has been internally parallelized. Other heuristic skeletons follow a similar pattern. Tabu search can be parallelized in multiple ways. We have implemented four different parallel versions of the skeleton: (1) a direct parallelization by independent runs; (2) a classical Master–Slave model; (3) independent runs with different search strategies; and (4) a Master–Slave model in which the work performed by a slave is to search a portion of the neighborhood of the current solution. Fig. 4 shows the UML diagram of the classes that implement the tabu search skeleton. All these parallel schemes have been implemented and tested over an homogeneous PC cluster using Linux as Operating System and `NetStream` using the MPI communication library.

4.4.1. Tabu search for 0–1 multidimensional Knapsack problem (0–1 MKNP)

We have instantiated the tabu search skeleton for the 0–1 multidimensional Knapsack problem (see Appendix A for more details). We give in Table 3 some experimental results for the independent runs model with search strategies. These results are obtained for six middle-sized instances from the OR-Library [8]. Results for the same problem and the same instances for the Master–Slave model with neighborhood partition are given in Table 4 (for more detail, see [9,10]). The execution is done with different maximum execution times and different number of processors. In the table, instance refers to the instance name, n and m are respectively the number of variables (items) and the number of restrictions; best known cost column gives the best known cost in the literature [15] for the instance obtained by other specific implementations and best obtained cost indicates the best cost found by our parallel implementation. The column dev indicates the deviation of the best found cost w.r.t. the best known cost, computed as (best known cost-best obtained cost)/(best known cost).

The results we obtained did not improve the best results known so far, but they are a considerably accurate approach if we take into account the reduced time (900 s) invested in their resolution (in the literature, the

Table 3
Results for 0–1 MKNP under an independent runs model with Autonomous Strategies using 4 (up) and 8 (low) processors

Instance	n	m	Best cost known	Best obtained cost	Average cost obtained	dev. w.r.t. best known	Iterations (average)
OR5x250-00	250	5	59,312	58,417	57,698.8	0.0151	3192.0
OR5x250-29	250	5	154,662	153,666	153,297.0	0.0064	1360.6
OR10x250-00	250	10	59,187	56,376	55,757.0	0.0475	1566.4
OR10x250-29	250	10	149,704	14,8077	147,831.6	0.0109	1152.8
OR30x250-00	250	30	56,693	55,412	55,055.2	0.0226	1575.6
OR30x250-29	250	30	149,572	148,010	147,720.4	0.0104	398.0
OR5x250-00	250	5	59,312	58,106	57,713.2	0.0203	8922.8
OR5x250-29	250	5	154,662	154,188	153,972.0	0.0031	4407.8
OR10x250-00	250	10	59,187	56,496	56,198.8	0.0455	4435.4
OR10x250-29	250	10	149,704	149,030	148,840.2	0.0045	3762.8
OR30x250-00	250	30	56,693	55,725	55,494.6	0.0170	4764.8
OR30x250-29	250	30	149,572	149,034	148,969.6	0.0036	2231.4

Instances solved with $max\_time = 900$ s.

execution times vary roughly from 700 s to 2500 s, see [15]). By comparing parallel models (see Tables 3 and 4) we can observe that the Independent Run model gets slightly better results than the Master–Slave with neighborhood partition. The reason for that lays on the communication necessities. A Master–Slave model, especially when including a neighborhood partition and distribution, requires much more communication among processes than an independent runs approach, in which the processes work independently and do not exchange any information. Another reason for the better performance of the independent runs model in comparison to the Master–Slave model can be found in the fact that in the former much more work is done.

4.4.2. Simulated annealing for minimum linear arrangement

The parallel skeletons for simulated annealing have also been used to find good solutions for the minimum linear arrangement problem (see Appendix A for more details). In this work we study the `airfoil1` instance having a 2D-mesh graph typically used in FE methods. It is made up of 4253 nodes, 12,289 edges and has diameter 64. The best layout found has cost 285,31. In this case, two alternative parallel models have been considered (see [43] for details). In the *accurate* version, all the processors cooperate to maintain a correct value of the objective function. In the *chaotic* version, processors do not cooperate, and thus their individual estimation of the objective function may contain some errors; periodic synchronizations are performed to maintain these errors small. For the accurate version, important time reductions are observed but quality

Table 4
Results for 0–1 MKNP under a Master–Slave model with neighborhood partition using 4 (up) and 8 (low) processors

Instance	n	m	Best cost known	Best obtained cost	Average cost obtained	dev. w.r.t. best known	Iterations (average)
OR5x250-00	250	5	59,312	57,838	57,059.6	0.0249	747.2
OR5x250-29	250	5	154,662	153,518	153,079.8	0.0074	484.4
OR10x250-00	250	10	59,187	55,352	54,912.8	0.0648	278.4
OR10x250-29	250	10	149,704	147,865	147,771.0	0.0123	363.6
OR30x250-00	250	30	56,693	55,082	54,112.4	0.0284	142.8
OR30x250-29	250	30	149,572	148,142	148,019.2	0.0096	129.6
OR5x250-00	250	5	59,312	58,085	57,923.0	0.0207	1223.8
OR5x250-29	250	5	154,662	153,636	153,283.2	0.0066	746.4
OR10x250-00	250	10	59,187	55,374	54,941.8	0.0644	505.4
OR10x250-29	250	10	149,704	148,220	147,954.2	0.0099	669.2
OR30x250-00	250	30	56,693	54,951	54,386.4	0.0307	269.0
OR30x250-29	250	30	149,572	148,424	148,099.2	0.0077	221.4

Instances solved with $max\_time = 900$ s.

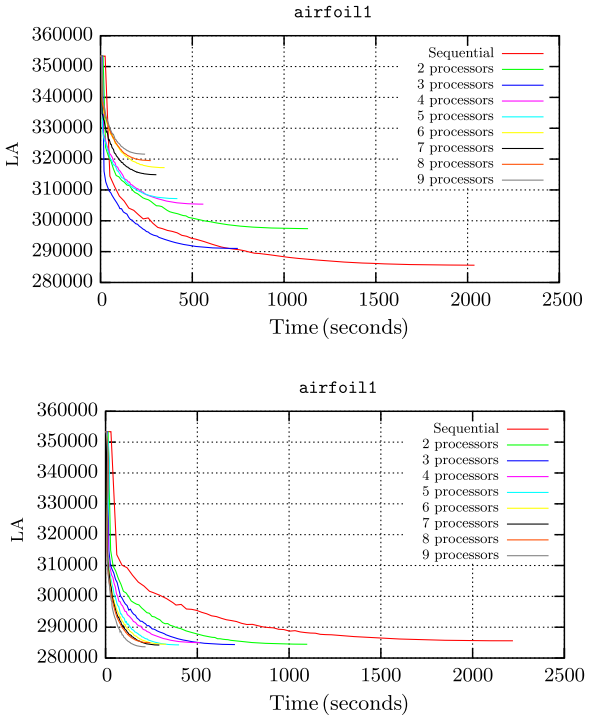


Fig. 5. Accurate (up) and chaotic (down) parallel SS+SA on the `airfoil1` graph: cost in function of time depending on the number of processors.

degrades as the number of processors increases. On the contrary, the chaotic version obtains an excellent speedup and maintains the solution quality (see Fig. 5).

5. Hybrid optimization techniques

Many studies have been done to design new techniques aimed at improving the quality of the results obtained with pure algorithms. One of these techniques consists in making several algorithms work together in order to profit from the best characteristics of each of them. The resulting algorithm is then called a *hybrid algorithm*. In the most general framework, algorithms can be hybridized with any other algorithm, even belonging to a quite different optimization family.

In our project, we naturally deal with several optimization techniques or solvers. We have built hybrid skeletons by combining evolutionary algorithms,[1] in particular genetic algorithms, a CHC algorithm, and an evolution strategy. Also, a local search technique like simulated annealing (SA) has been included. See Sections 4.1, 4.2 and Appendix B for a detailed definition these methods.

In this work we have implemented two hybrid algorithms, namely GASA and CHCES; they are two different instances of the *weak hybrid* scheme mentioned. The first of them (GASA) is made of a genetic algorithm and a simulated annealing; the second one uses this same scheme to combine CHC and ES. The rationale for this selection of algorithms is that, while the GA/CHC locates "good" regions of the search space (exploration), the SA/ES allows for exploitation in the best regions found by its partner.

We have implemented two main classes of hybrids in our research project:

[1] A hybrid algorithm that uses both traditional LS-based methods and evolutionary techniques is sometimes refereed to as a Memetic Algorithm (see [11,42]).

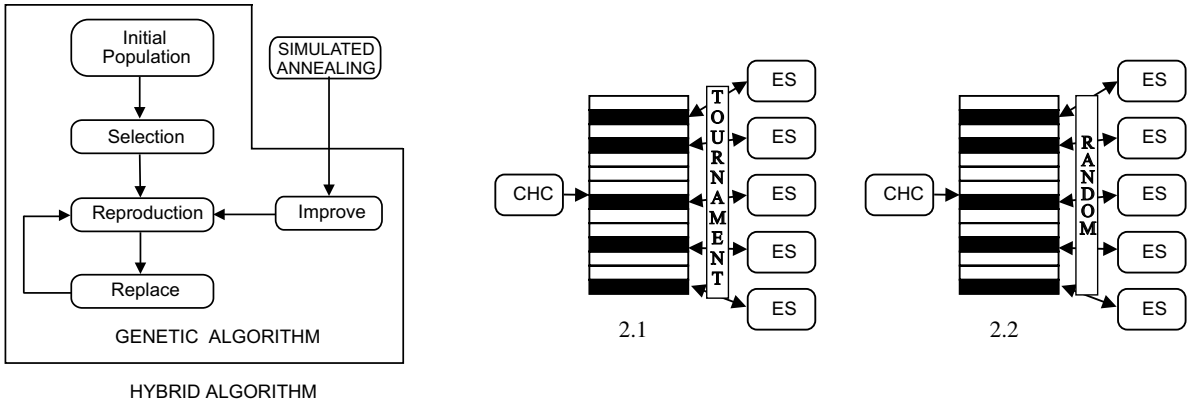


Fig. 6. Models of Hybridization: (left) Model of Hybridization 1 (GASA1) and (right) Model of Hybridization 2 (CHCES2/3).

- A first hybrid schema (GASA1/CHCES1) where a GA/CHC algorithm uses the other algorithm (SA/ES) as an evolutionary operator; the local search algorithm is applied in the main loop after the traditional recombination and mutation operators. See an example for GASA1 in Fig. 6 (left).
- A second hybrid schema executes a GA/CHC until the algorithm completely finishes. Then the hybrid selects some individuals from the final population and executes a SA/ES algorithm over them. We have implemented two variants whose only difference is the selection method. Concretely, we analyze a first version (GASA2/CHCES2) that uses a tournament selection (model 2.1 of Fig. 6 right), and another version (GASA3/CHCES3) that uses a random choice of individuals (model 2.2 of Fig. 6 right).

5.1. Parallel implementations

As we noted earlier, the parallelism does not only allow to reduce the execution time but it also allows to improve the quality of the solutions. Even when the parallel algorithms are executed in a single processor, they can improve the behavior of the serial ones. We will also improve the performance of our hybrid algorithms by making their components run in parallel. Thus all techniques composing our hybrid algorithms have been previously parallelized for LAN and WAN platforms. Moreover to explore the behavior of parallel hybrids is doubly interesting for the MALLBA project since we want to conduct our research in LAN and WAN platforms. For the present study we implemented three parallel distributed EAs, whose component sub-algorithm is a GA, an ES or a CHC. Different implementations can be obtained by creating separate subclasses of the `Solver` abstract class (see Fig. 1a). At present, we are using our own middleware layer `NetStream` implemented on top of MPI to ease communications.

5.1.1. Parallel EAs

A parallel EA (PEA) is an algorithm having multiple components EAs, regardless of their population structure. Each component (usually a canonical EA) sub-algorithm includes an additional phase of *communication* with a set of neighboring sub-algorithms [5]. Different parallel algorithms differ in the characteristics of their elementary skeletons, and in the communication details.

As an example parallel skeleton we have chosen a *distributed GA* (dGA) because of its popularity and because it can be readily implemented in clusters of machines. In such a distributed GA (or EA) there exists a small number of islands performing separate GAs, and periodically exchanging individuals after a number of isolated steps (*migration frequency*).

The migration policy must define the island topology, when migration occurs, which individuals are being exchanged, the synchronization among the sub-populations, and the kind of integration of exchanged individuals within the target sub-populations. Concretely, we use a static ring topology, select random migrants and include them in the target populations only if they are better than the worst-existing solutions.

5.1.2. Parallel SAs

For the parallel SA (PSA) there also exist multiple component SAs. Each component SA periodically exchanges the best solution found after a number of isolated steps (*cooperation* phase) with its neighbor in the ring.

5.1.3. Parallel hybrids

The hybrid versions apply, in the parallel case, SA (ES) as an operator in the basic loop of every island in the GA (CHC) algorithm. The result is a parallel version of what we will call GASA1 (CHCES1).

5.2. Instantiation examples and results

In this section we include the analysis of the performance of sequential, LAN and WAN hybrid skeletons for four problems: two of them have a combinatorial nature (MaxCut and MTTP) and the two others are representatives of the continuous optimization domain (RAS and FMS). See the details on these problems in the Appendix A at the end of the paper. Our goal with the upcoming results is to compare hybrid and pure search schemes in all these platforms and also to show that the underlying philosophy of MALLBA is efficient and accurate, as least as compared against the alternative of making separated and unstructured *ad hoc* implementations.

In Table 5 we provide the parameters used for the non-hybrid (basic) skeletons, while in Table 6 we include the parameters used for the incorporated operators. We tend towards a low-cost utilization of SA/ES in the hybrid skeletons to promote gradual exploitation of solutions during the search.

We show the results for the sequential, LAN and WAN platforms in Tables 7–9, respectively. Results are the average values of 30 independent runs for each problem, in each one of the three platforms. Since we want a fair comparison we begin with a canonical having one workstation in each of the three geographically separated sites.

Table 5
Parameters of the algorithms

Problem		Popsize	Cross. prob.	Mut. prob.	Others
MaxCut	(GA)	100	0.8	0.01	–
MTTP	(GA)	200	0.6	0.02	–
RAS, FMS	(CHC)	100	0.8	–	35% population restart

Table 6
Parameters of the hybrid operators

Algorithm	Probability	#Max iteration	Others
SA	0.1	100	MarkovChainLen = 10 Temperature decay factor = 0.99
ES	0.01	50	(1 + 10)-ES, mutation prob. = 0.8

Table 7
Average results in the sequential platform

Problem	Algorithm	Opt.	#Evals	Time (s)	Hits (%)
MaxCut	GA	1008	33,794	68.2	3.3
	GASA1	1030	48,823	96.3	9.9
MTTP	GA	219	42,017	5.8	66.6
	GASA1	200	38,297	5.5	100
RAS	CHC	0	9634	4.15	100
	CHCES1	0	14,413	4.82	100
FMS	CHC	3.095	32,270	19.65	100
	CHCES1	2.078	17,962	26.62	100

Table 8
Average results for the LAN platform

Problem	Algorithm	Opt.	#Evals	Time (s)	Hits (%)
MaxCut	GA	1031	23,580	49.1	16.6
	GASA1	1038	40,682	89.6	16.6
MTTP	GA	201	40,002	5.2	96.6
	GASA1	200	25,601	5.8	100
RAS	CHC	0	7591	3.33	100
	CHCES1	0	13,048	3.73	100
FMS	CHC	3.079	27,341	9.2	100
	CHCES1	1.692	16,558	11.45	100

Table 9
Average results for the WAN platform

Problem	Algorithm	Opt.	#Evals	Time (s)	Hits (%)
MaxCut	GA	1014	14,369	89.1	9.9
	GASA1	1031	28,956	298.5	9.9
MTTP	GA	200	32,546	25.7	100
	GASA1	200	34,952	45.62	100
RAS	CHC	0	7606	32.7	100
	CHCES1	0	13,681	10.0	100
FMS	CHC	2.929	29,743	17.25	100
	CHCES1	1.341	17,816	26.69	100

After observing these results, we can conclude that LAN and WAN enhance the percentage of hits (number of times locating an optimum) of the sequential platform, especially for the discrete problems. The LAN skeletons provide the best execution times for all the problems, but the speedup is sub-linear. An interesting result occurs for the FMS problem in which the WAN skeletons outperform the sequential and LAN ones in accuracy (**opt.** column) with similar times than the sequential time (LAN is faster), and with an equivalent number of evaluations than sequential and LAN. Therefore, although the reductions in time are important (especially for LAN skeletons), the most relevant conclusions focus in the numerical results, since the WAN skeletons are competitive in this sense with sequential and LAN versions. We have also obtained similar results for other problems (not only for FMS). For example, in [3], results for the VRP are showed where the WAN configuration outperform (in time and in solutions quality) the LAN one. These are great news for our intended future work on aggregating a high number of computers in WAN.

Now let us compare our results with results in the literature. Some up-to-date results on the same instances of MaxCut and MTTP can be found in [2], where the authors analyze three types of sequential and distributed EAs. Our results in MALLBA clearly outperform those of [2] for MaxCut, whose best percentage of hits is 5%, while ours are between around 10% and 16% in LAN and WAN, with an additional reduction in the whole search effort. For MTTP, we offer an almost constant 100% of hits with pure and hybrid skeletons with below 40,000 evaluations (with the exception of our 66% for sequential GA); however, in [2] the authors report similar hits percentages, but with a number of evaluations (specially for their LAN algorithms) well above 100,000. Similarly, all our hybrid versions outperform any of their evaluated pure algorithms in efficiency clearly.

For our two continuous optimization problems (RAS and FMS) the results reproduce other existing values within a lower run time (in LAN); RAS has been optimized quite accurately in all our skeleton versions, while FMS admits clear improvements in accuracy that are only possible with specialized operators just like the ones investigated in [30].

6. Concluding remarks and future work

We have sketched the architecture of the MALLBA library, including its design goals, skeleton implementation, available resources and communication issues for parallelizing then in LAN and WAN platforms. Also,

we have presented and evaluated a large set of exact, heuristic, and hybrid algorithms on a benchmark showing many different difficulties. Our goal in this paper has been to design, implement and evaluate most popular classes of skeletons for optimization targeted to the three platforms more readily accessible for researchers: sequential, LANs and WANs.

Our experience after conducting this work indicates that these skeletons can be easily instantiated for a large number of problems. Sequential instantiations provided by the users are ready to use in parallel; also, the parallel implementations are scalable, and the evaluated skeletons have provided solutions whose quality is comparable to *ad hoc* implementations for concrete problems. The architecture supports easy construction of hybrid algorithms and a fast prototyping phase.

Our future work will focus on offering a more complete set of skeletons for LAN and WAN, and to export the whole architecture to be utilized from foreign non-MALLBA environments.

Acknowledgements

This work has been partially supported by the Canarian Government Project PI/2000-60, by the Spanish projects CICYT TIC-1999-0754 (MALLBA) and MCYT TIC2002-04498-C05-02 (TRACER), and also by the European project of the IST program IST-2001-33116 (FLAGS).

Additionally, at the time when this research was done, C. León was partially supported by TRACS program at EPCC; M. Blesa was partially supported by the Catalan Research Council (grant 2001FI-00659) of the Generalitat de Catalunya, and G. Luque was partially supported by Andalusian 2002FPDI-43927 grant.

MALLBA is publicly available at the following websites: http://www.lsi.upc.edu/~mallba and http://neo.lcc.uma.es/mallba/easy-mallba/.

Appendix A. Problems

In this appendix, we present the optimization problems introduced along the paper. Globally stated, a combinatorial optimization problem is a tuple $\Pi = (I, S, f, g)$ where:

- I is the set of instances of Π. If $x \in I$ we say that x is an *instance* (or an input) of Π.
- Given an instance $x \in I$, $S(x)$ denotes the *set of feasible solutions* of x.
- For any instance $x \in I$ and any feasible solution $\sigma \in S(x)$, $f(x, \sigma)$ represents a real value, the measure (or cost or fitness) of σ with respect to Π and x. The function f is called the *objective function*.
- $g \in \{\max, \min\}$. The goal of Π is to find a feasible solution that optimizes f according to g: given an input $x \in I$, determine an *optimal solution* $\sigma' \in S(x)$ such that $f(x, \sigma') = g\{f(x, \sigma) | \sigma \in S(x)\}$.
- A *subproblem* Π_i is a tuple $\Pi_i = (I, S_i, f, g)$ where $S_i(x)$ is a subset of the underlying space I.

Approximating a problem Π for an instance $x \in I$ means finding a feasible solution $\sigma \in S(x)$ with a cost as close as possible to $f(x,\sigma')$.

A.1. The frequency modulation sounds problem (FMS)

The frequency modulation sounds [47] has been proposed as a hard real task consisting in adjusting a general model $y(t)$ to a basic sound function $y_0(t)$. The goal is to minimize the sum of square errors given by Eq. (1). The problem is to evolve six parameters $\vec{x} = (a_1, w_1, a_2, w_2, a_3, w_3)$ in order $y(t)$ to fit the target $y_0(t)$. The evolved and target models have the expressions shown in Eqs. (2) and (3). The resulting problem is a highly complex multimodal function having strong epistasis with minimum value $f^* = 0$. For the experiments, we consider as an optimum any solution with fitness below 3.1.

$$\mathrm{FMS}(\vec{x}) = \sum_{i=0}^{N} (y(t) - y_0(t))^2 \tag{1}$$

$$y(t) = a_1 \sin(w_1 t\theta + a_2 \sin(w_2 t\theta + a_2 \sin(w_3 t\theta))) \quad (2)$$

$$y_0(t) = 1.0 \sin(5.0 t\theta + 1.5 \sin(4.8 t\theta + 2.0 \sin(4.9 t\theta))) \quad (3)$$

$$\theta = 2\pi/100, \quad a_i, w_i \in [-6.4, 6.35]$$

A.2. The minimum linear arrangement problem (MINLA)

Given a graph $G = (V, E)$ with $n = |V|$ nodes, a linear arrangement is a bijection φ: $V \to \{1, \ldots, n\}$. The problem is to find a linear arrangement that minimizes the following objective function:

$$\sum_{(u,v)\in E} |\varphi(u) - \varphi(v)|,$$

which can be viewed as the minimal sum of the edges lengths of the graph.

A.3. The maximum cut problem (MaxCut)

The maximum cut problem [2] consists in partitioning the set of vertices of a weighted graph into two disjoint subsets such that the sum of the weights of edges with one endpoint in each subset is maximized. Thus, if $G = (V, E)$, denotes a weighted graph where V is the set of nodes and E the set of edges, then the maximum disjoint sets V_0 and V_1 such that the sum of weights of E that have one endpoint in V_0 and the other in V_1 is maximized. We use a binary string $(x_1, x_2, \ldots, x_n)$ of length n where each digit corresponds to a vertex. Each string encodes a partition of the vertices. If a digit is 1 then its corresponding vertex is in set V_1, if it is 0 then the corresponding vertex is in set V_0. The function to be maximized is

$$F(\vec{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij}\big(x_i(1 - x_j) + x_j(1 - x_i)\big)$$

For the experiments reported here, we use a scalable problem instance with a graph of size $n = 100$, "cut100" ($f^* = 1077$).

A.4. The minimum tardy task problem (MTTP)

The minimum tardy task problem is a task-scheduling problem [35]. Each task i from the set of tasks $T = 1, 2, \ldots, n$ has an associated length l_i, the time it takes for its execution, a deadline d_i before which the task must be scheduled and its execution completed, and a weight w_i. The weight is a penalty indicating the importance that a task remain unscheduled. Scheduling the tasks of a subset S of T consists in finding the starting time of each task in S, such that at most one task at a time is performed, and such that each task finishes before its deadline. The optimal solution is a feasible schedule S with the minimum tardy task weight W which is the sum of weights of unscheduled tasks.

$$\min W = \sum_{i \in T - S} w_i$$

A feasible solution must satisfy that no task is scheduled before the completion of an earlier scheduled one and all tasks are completed within its deadline.

For our experiments, we use a scalable problem instance [35] of size 100 task, "mttp100" ($f^* = 200$).

A.5. The 0–1 Knapsack problem

We consider the classical 0–1 Knapsack problem where a subset of N given items has to be introduced in a knapsack of capacity C. Each item has a profit p_i and a weight w_i and the problem is to select a subset of items whose total weight does not exceed C and whose total profit is a maximum. We assume that all input data are positive integers. Introducing the binary decision variables x_i, with $x_i = 1$ if item i is selected, and $x_i = 0$ otherwise, we can formulate the problem as follows:

$$\begin{aligned} \max \quad & \sum_{i=1}^{N} p_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{N} w_i x_i \leqslant C \\ & x_k \in \{0,1\}, \quad k \in \{1,\ldots,N\} \end{aligned}$$

A.6. The 0–1 multidimensional Knapsack (0–1 MKNP)

The NP-hard 0–1 Multidimensional Knapsack problem consists in selecting a subset of n given objects (or items) in such a way that the total profit of the selected objects is maximized while a set of knapsack constraints are satisfied. The 0–1 MKNP problem can be stated as $\max c_x$ subject to $Ax \leqslant b$ and $x \in \{0,1\}^n$, where $c \in \mathbb{N}^{*n}$, $A \in \mathbb{N}^{mxn}$ and $b \in \mathbb{N}^m$. The binary components x_j of x are decision variables: $x_j = 1$ if the object j is selected, 0 otherwise. The profit associated to j is denoted by c_j. Each of the m constraints $A_i x \leqslant b_i$ is called a knapsack constraint.

A.7. The resource allocation problem

The (single) resource allocation problem can be stated as follows [33]:

$$\begin{aligned} \max \quad & z = \sum_{j=1}^{N} f_j(x_j) \\ \text{s.t.} \quad & \sum_{j=1}^{N} x_j = M \end{aligned}$$

Namely, it is required to allocate M units of an indivisible resource to N tasks so that the sum of the effectiveness measured by $f_j(x_j)$ is maximized.

A.8. The Rastrigin function (RAS)

The generalized Rastrigin function (see below) is a problem with a large search space and a very large number of local optima [45]. This function is a non-epistatic function representing a typical test for EAs. For the experiments, we have used a problem instance of 20 variables (fitness values $f^* = 0$).

$$\mathrm{Ras}(x_i|_{i=1,\ldots,n}) = 10 \cdot n + \sum_{i=1}^{n} [x_i^2 - 10 \cdot \cos(2 \cdot \pi x_i)], \quad x_i \in [-5.12, 5.12].$$

Appendix B. Pseudo-codes of the methods

In this appendix, we show the pseudo-codes of the methods used along the paper.

Algorithm 1 (*Pseudo-code of branch-and-bound method* (*maximization case*)).

$\mathscr{L} := \{\Pi_0\}$; $\mathscr{Q} := \{\Pi_0\}$; $bs := -\infty$; $\mathscr{T} := \varnothing$
while $\mathscr{L} \neq \varnothing$ **do**
  $\Pi_i := s(\mathscr{L})$
  if *upper_bound*$(\Pi_i) > bs$ **then**
    if *lower_bound*$(\Pi_i) > bs$ **then**
      $bs :=$ *lower_bound*(Π_i)
      $\mathscr{T} := \{\sigma\}$ // σ satisfies $f(x,\sigma) =$ *lower_bound*(Π_i)

```
        else
          (branch) decompose Π_i into Π_i1,Π_i2,...,Π_ik
          L := L ∪ {Π_i1, Π_i2, ..., Π_ik} − {Π_i}
          Q :=Q ∪ {Π_i1, Π_i2, ..., Π_ik}
        end if
      end if
      L := L − Π_i
    end while
    return ⟨bs, T⟩ // T is the best solution and bs its value
```

Algorithm 2 (*Pseudo-code of dynamic programming* (*maximization case*)).

```
    F := ∅
    // Loop on the stages
    for i := 1;i ⩽ n do
      // Loop on the states
      for j := 1;j ⩽ n(i) do
        generate(Π_ij); bs_ij := −inf; d_ij := ∅// Initialize state
        // Loop on the decisions
        for all d' ∈ Σ do
          evaluate(Π_ij,s',d')
          if s' < bs_ij then
            d_ij := d'; bs_ij := s'
          end if
          F := F ∪ {Π_ij, bs_ij, d}
        end for
      end for
    end for
    return F // F stores the best solution and the best solution value
```

Algorithm 3 (*Pseudo-code of basic local search* (*maximization case*)).

```
    s := Initial_Solution()
    while there exists s' ∈ N(s) with f(s) < f(s') do
      m := Generate_Move()
      s' := Apply(m,s) // s' is candidate solution
      if f(s) < f(s') then
        s := s'
      end if
    end while
    return s // s is a locally optimal solution
```

Algorithm 4 (*Pseudo-code of simulated annealing* (*SA*)).

```
    t := 0
    Initialize T
    s0 := Initial_Solution()
    v0 := Evaluate(s0)
    while 'outer-loop stop criterion' is not satisfied do
```

```
    while t mod MarkovChainLen ≠ 0 do
      t := t + 1
      s1 := Generate(s0, T) //Move
      v1 := Evaluate(s1)
      if Accept(v0, v1, T) then
        s0 := s1
        v0 := v1
      end if
    end while
    T := Update(T)
  end while
  return s0
```

Algorithm 5 (*Pseudo-code of an evolutionary algorithm* (*EA*)).

```
  Generate(P(0))
  t := 0
  while not Termination_Criterion((P(t)) do
    Evaluate(P(t))
    P'(t) := Selection(P(t))
    P'(t) := Apply_Reproduction_Ops(P'(t))
    P(t + 1) := Replace(P(t), P'(t))
    t := t + 1
  end while
  return Best_Solution_Found
```

Algorithm 6 (*Pseudo-code of the CHC algorithm*).

```
  t := 0
  d := L/4 // L is the length of a tentative solution
  initialize P(t)
  evaluate structures in P(t)
  while not end do
    t := t + 1
    select C(t) from P(t − 1)
    C'(t) := HUX(C(t))
    evaluate structures in C'(t)
    replace P(t) from C'(t) and P(t − 1)
    if P(t) = P(t − 1) then
      d := d − 1
    end if
    if d < 0 then
      diverge P(t) // restart
      d := r * (1.0 − r) * L
    end if
  end while
  return Best_Solution_Found
```

Algorithm 7 (*Pseudo-code of an evolution strategy* (*ES*)).

```
  t := 0
  initialize P(t)
```

evaluate structures in $P(t)$ //*containing problem variables plus control parameters*
while not end **do**
  $t := t + 1$
  $C(t) :=$ select_best_from($P(t-1)$)
  mutate structures in $C(t)$ to yield $C'(t)$ // *crossover could also exist*
  evaluate structures in $C'(t)$
  replace $P(t)$ from $C'(t)$ and/or $P(t-1)$
end while
return Best_Solution_Found

References

[1] E. Alba, C. Cotta, M. Díaz, E. Soler, J.M. Troya. MALLBA: middleware for a geographically distributed optimization system, Technical report, Dpto. Lenguajes y Ciencias de la Computación, Universidad de Málaga (internal report), 2000.
[2] E. Alba, S. Khuri, Sequential and distributed evolutionary algorithms for combinatorial optimization problemsAdvances in Soft Computing – Hybrid Information Systems, vol. 113, Physica-Verlag, Heidelberg, 2002, pp. 211–233, Chapter10.
[3] E. Alba, G. Luque, J.M. Troya, Parallel LAN/WAN heuristics for optimization, Parallel Computing 30 (5–6) (2004) 611–628.
[4] E. Alba, the MALLBA Group, MALLBA: a library of skeletons for combinatorial optimisation, in: R.F.B. Monien (Ed.), Proceedings of the Euro-Par, Lecture Notes in Computer Science, vol. 2400, Springer-Verlag, Paderborn, GE, 2002, pp. 927–932.
[5] E. Alba, M. Tomassini, Parallelism and Evolutionary Algorithms, IEEE Transactions on Evolutionary Computation 6 (5, Oct.) (2002) 443–462.
[6] M.G. Arenas, P. Collet, A.E. Eiben, M. Jelasity, J.J. Merelo, B. Paechter, M. Preub, M. Schoenauer, A framework for distributed evolutionary algorithms, in: J.J. Merelo, P. Adamidis, H.G. Beyer, J.L. Fernández-Villacañas, H.P. Schwefel (Eds.), Seventh International Conference on Parallel Problem Solving from Nature, 2002, pp. 665–675.
[7] T. Bäck, D.B. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, Oxford University Press, 1997.
[8] J.E. Beasley, OR-Library: distributing test problems by electronic mail, Journal of Operational Research Society 41 (11) (1990) 1069–1072. Available from: <http://mscmga.ms.ic.ac.uk/info.html>.
[9] M. Blesa, L. Hernández, F. Xhafa, Parallel skeletons for Tabu Search Method, in: 8th International Conference on Parallel and Distributed Systems, IEEE Computer Society Press, 2001, pp. 23–28.
[10] M. Blesa, L. Hernández, F. Xhafa, Parallel skeletons for Tabu Search Method based on search strategies and neighborhood partition, in: M. Paprzycki, J. Dongarra, J. Wasniewski (Eds.), 4th International Conference on Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 2328, Springer–Verlag, 2002, pp. 185–193.
[11] M. Blesa, P. Moscato, F. Xhafa, A memetic algorithm for the minimum weighted *k*-cardinality tree subgraph problem, in: 4th Metaheuristics International Conference, vol. 1, Porto, Portugal, 2001, pp. 85–90.
[12] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, PISA – a platform and programming language independent interface for search algorithms.
[13] S. Cahon, E.-G. Talbi, N. Melab, PARADISEO: a framework for parallel and distributed biologically inspired heuristics, in: IPDPS-NIDISC'03, Nize, France, 2003, p.144.
[14] E. Cantú-Paz, Efficient and Accurate Parallel Genetic Algorithms, Kluwer, Academic Press, 2000.
[15] P.C. Chu, J.E. Beasley. A Genetic Algorithm for the multidimensional knapsack problem, working paper, 1997.
[16] C. Cotta, J.M. Troya, On decision-making in strong hybrid evolutionary algorithms, in: A.P.D. Pobil, J. Mira, M. Ali (Eds.), Tasks and Methods in Applied Artificial Intelligence, Lecture Notes in Computer Science, vol. 1416, Springer-Verlag, Berlin Heidelberg, 1998, pp. 418–427.
[17] T.G. Crainic, M. Toulouse, Parallel metaheuristics, Technical report, DTpt. des sciences administratives (UniversitT du QuTbec a MontrTal) and Centre de recherche sur les transports (UniversitT de MontrTal) and School of Computer Science (University of Oklahoma), 1997.
[18] B.L. Cun, Bob++ library illustrated by VRP, in: European Operational Research Conference (EURO'2001), Rotterdam, 2001, p. 157.
[19] J.M. Daida, S.J. Ross, B.C. Hannan, Biological symbiosis as a metaphor for computational hybridization, in: L.J. Eshelman (Ed.), Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, 1995, pp. 328–335.
[20] L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.
[21] I. Dorta, C. León, C. Rodrfguez, A. Rojas, Parallel skeletons for divide-and-conquer and branch-and-bound techniques, in: 11th Euromicro Conference on Parallel, Distributed and Network-based Processing, IEEE Computer Society Press, 2003, pp. 292–298.
[22] J. Eckstein, C.A. Phillips, W.E. Hart. PICO: an object-oriented framework for parallel branch and bound, Technical report, RUTCOR, 2000.
[23] J. Eggermont, A.E. Eiben, J.I. van Hemert, A comparison of genetic programming variants for data classification, in: D.J. Hand, J.N. Kok, M.R. Berthold (Eds.), Advances in Intelligent Data Analysis, Third International Symposium, IDA-99, 9–11 1999, Amsterdam, The Netherlands, vol. 1642, Springer-Verlag, 1999, pp. 281–290.
[24] L. Eshelman, The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination, in: Foundations of Genetic Algorithms, Morgan Kaufmann, 1991, pp. 265–283.

[25] C. Gagne, M. Parizeau. Open BEAGLE: a new C++ evolutionary computation framework, in: Proceeding of GECCO 2002, New York, NY, USA, 2002.
[26] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[27] L.D. Gaspero, A. Schaerf, EasyLocal++: an object-oriented framework for the flexible design of local search algorithms and metaheuristics, in: 4th Metaheuristics International Conference (MIC'2001), 2001, pp. 287–292.
[28] F. Glover, M. Laguna, Tabu Search, second ed., Kluwer, Boston, 1997.
[29] D. González, F. Almeida, L. Moreno, C. Rodrfguez, Towards the automatic optimal mapping of pipeline algorithms, Parallel Computing 29 (2003) 241–254.
[30] F. Herrera, M. Lozano, Gradual distributed real-coded genetic algorithms, IEEE Transactions on Evolutionary Computation 4 (1) (2000) 43–63.
[31] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor Michigan, 1975.
[32] T. Ibaraki, Enumerative approaches to combinatorial optimization, Part I, Annals of Operation Research 10 (1987).
[33] T. Ibaraki, Enumerative approaches to combinatorial optimization, Part II, Annals of Operation Research 11 (1988) 1–4.
[34] IBM, COIN: common optimization INterface for operations research, 2000. Available from: <http://oss.software.ibm.com/developerworks/opensource/coin/index.html>.
[35] S. Khuri, T. Bäck, J. Heitkötter, An evolutionary approach to combinatorial optimization problems, in: 22nd ACM Computer Science Conference, ACM Press, 1994, pp. 66–73.
[36] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[37] K. Klohs, Parallel simulated annealing library, 1998. Available from: <http://www.uni-paderborn.de/fachbereich/AG/monien/SOFTWARE/PARSA/>.
[38] D. Levine, PGAPack, parallel genetic algorithm library, 1996. Available from: <http://www.mcs.anl.gov/pgapack.html>.
[39] N.F. McPhee, N.J. Hopper, M.L. Reierson, Sutherland: an extensible object-oriented software framework for evolutionary computation, in: J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, R. Riolo (Eds.), Genetic Programming 1998: Proceedings of the Third Annual Conference, 22–25 1998, University of Wisconsin, Madison, WI, USA, Morgan Kaufmann, 1998, p. 241.
[40] W. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computing machines, The Journal of Chemical Physics 21 (6) (1953) 1087–1092.
[41] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer-Verlag, Berlin, 2000.
[42] P. Moscato, C. Cotta, A gentle introduction to memetic algorithms, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, MA, 2003, pp. 105–144.
[43] J. Petit, Combining spectral sequencing and parallel simulated annealing for the MinLA problem, Parallel Processing Letters 13 (1) (2003) 77–91.
[44] I. Rechenberg, Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Fromman-Holzboog Verlag, Stuttgart, 1973.
[45] A. Töorn, Ž. Antanas, Global optimizationLecture Notes in Computer Science, vol. 350, Springer, Berlin, Germany, 1989.
[46] S. Tschöke, T. Polzer. Portable parallel branch-and-bound library, 1997. Available from: <http://www.uni-paderborn.de/cs/ag-monien/SOFTWARE/PPBB/introduction.html>.
[47] S. Tsutsui, A. Ghosh, D. Corne, Y. Fujimoto, A real coded genetic algorithm with an explorer and an exploiter populations, in: T. Bäck et al. (Ed.), Proceedings of the Seventh International Conference on Genetic Algorithms, Morgan Kaufmann, 1997, pp. 238–245.
[48] S. Voss, D.L. Woodruff (Eds.), Optimization Software Class Libraries, Operations Research and Computer Science Interfaces, vol. 18, Kluwer Academic Publishers, Boston, 2002.
[49] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82.