Efficient Parallel LAN/WAN Algorithms for Optimization. The MALLBA Project

E. Alba[3] F. Almeida[2] M. Blesa[1] J. Cabeza[2] C. Cotta[3] M. Díaz[3]
I. Dorta[2] J. Gabarró[1] C. León[2] J. Luna[2] G. Luque[3] L. Moreno[2]
C. Pablos[2] J. Petit[1] A. Rojas[2] F. Xhafa[1]

[1] LSI – UPC. Campus Nord C6. 08034 Barcelona (Spain).
[2] EIOC – ULL. Edificio Física/Matemáticas. 38271 La Laguna (Spain).
[3] LCC – UMA. E.T.S.I. Informática. Campus de Teatinos. 29071 Málaga (Spain).

Abstract. The MALLBA project tackles the resolution of combinatorial optimization problems using algorithmic skeletons implemented in C++. MALLBA offers three families of generic optimization techniques: exact, heuristic and hybrid. Moreover, for each technique, MALLBA provides three different implementations: sequential, parallel for local area networks, and parallel for wide area networks. This paper explains the architecture of the MALLBA library, presents some of the implemented skeletons, and offers several computational results to show the viability of the approach. In our conclusions we claim that the design used to develop the optimization techniques is general and efficient at the same time, and also that the resulting skeletons can outperforms existing algorithms on a plethora of problems.

1 Introduction

The MALLBA project is an effort to develop an integrated library of skeletons for combinatorial optimization (including exact, heuristic and hybrid techniques) dealing with parallelism in a user-friendly and, at the same time, efficient manner. Its three target environments are sequential computers, LANs of workstations and WANs (local and wide area networks, respectively). The main features of MALLBA are: integration of all the skeletons under the same design principles, facility to switch from sequential to parallel optimization engines, and cooperation among solvers to provide more powerful hybrid skeletons, ready to use on commodity machines. Clusters of PCs under Linux are currently supported, and the resulting software architecture is flexible and extensible (new skeletons can be added, alternative communication layers can be used, etc.). See a quick introduction on MALLBA in [3].

Combinatorial optimization problems arise frequently in various fields such as Control Theory, Operations Research, Biology, Telecommunications and Computer Science. 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. [13, 18, 19, 21]). Also, Some existing frameworks, such as *Local++*, its successor *EasyLocal++* [12], *Bob++*

[8], and the IBM COIN open source project [16] provide sequential and parallel generic implementations for several exact, heuristic and hybrid techniques, but they lack features to integrate them. Furthermore, the DREAM project [5] is loosely related to our goals, although DREAM is targeted to distributed agent and simulations, not having a clear focus on optimization, and no evaluations exist showing its performance on a large set of problems.

In this work we describe the design, implementation and evaluation of a project that accomplishes for 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 manyfold. 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. Third, we are interested in showing really useful results, and thus we include a mixed set of algorithms and optimization problems showing some difficulties usually found in real-world tasks. Last, but not least, our conclusions are somewhat expected and somewhat surprising at the same time, since we do validate in practice some theoretical thoughts on WAN optimization, but also we are able to report competitive performance in WAN.

To our knowledge, there is no other work that considers, at the same time, all these classes of algorithms and problems, and that also extends the analysis to LAN and WAN environments.

In this paper we first present (Section 2) the architecture of MALLBA and its advantages for developing new algorithms and fast prototyping. In Sections 3, 4, and 5 we provide working examples plus a numerical and time analysis of exact, heuristic and hybrid algorithms; they all follow the basic MALLBA architecture for sequential, LAN and WAN platform execution, and all them are shown to be competitive when compared against existing results in literature. In Section 6 we include some summary conclusions and future work. Finally, we have added two appendices to this paper; the first one contains an example instantiation of an optimization skeleton in MALLBA to help the interested reader. The second one lists all the problems whose solution is addressed in the evaluation sub-sections for each one of the techniques.

2 The MALLBA Architecture

The MALLBA project is an effort to develop a library of algorithms for optimization that can deal with parallelism in a user-friendly and, at the same time, efficient manner. Its three target environments are sequential, LAN and WAN computer platforms. All the algorithms in this paper are implemented as *software skeletons* with a common internal and public interface. Every skeleton implements a resolution technique for optimization problems, taken from the fields of exact, heuristic or hybrid optimization. This permits fast prototyping and transparent access to parallel platforms.

MALLBA skeletons are based on the separation of two concepts: the concrete 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 to parallelize the execution of the resolution technique is implemented in the skeleton itself. The user does not program 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 manipulate tentative solutions (merge, cut, or interpret parts of a solution, for example). 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 by a set of *required* and *provided* C++ classes which represent object abstractions of the entities participating in the resolution technique. The *provided* classes implement internal aspects of the skeleton in a problem-independent way. Typically, these internal aspects refer to the implementation of the resolution technique. Those classes have been completely implemented in the respective skeletons. The *required* classes specify information and behavior 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. Fig. 1 depicts this architecture.

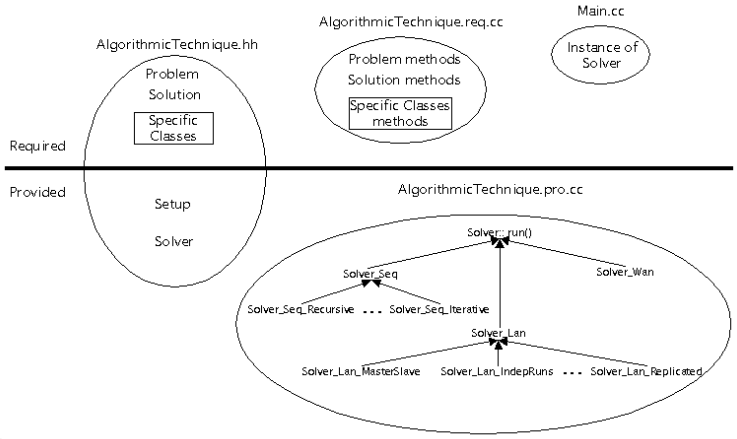

Fig. 1. Architecture of a MALLBA skeleton. The horizontal line stands for the separation in MALLBA between the C++ classes the user must code -upper part of the figure- and the classes that MALLBA already includes in a fully operational form -lower part-.

Therefore, the user of a MALLBA skeleton only needs to implement the particular features related to the problem, i.e., to fill ul the required classes with an specific problem-dependent implementation. This speeds the creation of new algorithms with a minimum effort considerably.

The MALLBA infrastructure is composed of computers and communication networks from the Universities of Málaga (UMA), La Laguna (ULL) and Barcelona (UPC). These three universities are connected through RedIRIS, the academic and scientific computer network, and managed by CSIC (*Consejo Superior de Investigaciones Científicas*), that connects the main universities and research centers in Spain. RedIRIS is a WAN with ATM technology (ATM accesses of 34/155 Mbps). There is a node in each administrative region of Spain.

In the next section we proceed to introduce and discuss the utilization interface in order a new user could program its own skeleton in MALLBA. Next, we will discuss the communication and the hybridization interfaces in the other two sub-sections. The aim is to explain first what a final user must consider, then what a really internal programmer needs to know about the parallel issues and finally to discuss how to merge skeletons to yield new optimization procedures. We need these three descriptions since what a "user" is depends on the level of interaction of a researcher with our software and its goal: using MALLBA for his/her problem, changing communications, or creating new techniques for optimization, respectively.

2.1 Utilization Interface

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. Since LAN and WAN skeletons exist, the user can now execute the resulting program on sequential or parallel environments.

Apart from some illustrative examples, MALLBA does not contain any actual code for solving specific problems. On the contrary, it provides the generic code the user has to customize. This way, a single implementation –abstract yet efficient– can be reutilized in different contexts. The user need not have a deep knowledge about parallelism or distributed computing; these aspects are already included in the library.

See the global picture of the system in Fig. 2. 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 provided by MALLBA and even by reusing some parts of existing skeletons. Each skeleton could have its own configuration file to avoid recompilation when parameters change. Also, the reader can appreciate the three types of users we envision, namely: final user, programmer and internal filler.

Let us now get deeper in our understanding of the provided and required C++ classes.

– **Provided classes**: Classes within this category are responsible for implementing the basic functionality of the corresponding skeleton. First of all, class `Solver` encapsulates the behavior of the algorithm under consideration.

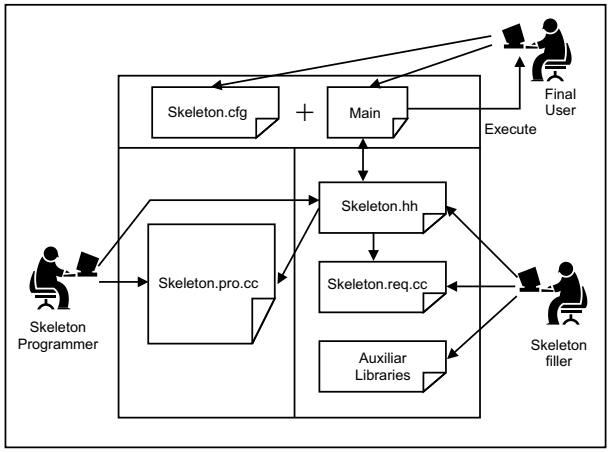

Fig. 2. Users and MALLBA.

This optimization engine is fully generic, and interacts with the problem using the classes required from the user (see below). In order to enable a skeleton to have different solver engines, the `Solver` class defines a unique interface and offers several subclasses that provide different sequential and parallel implementations (see Fig. 1). Secondly, class `SetupParams` contains all parameters describing the execution of the algorithm, e.g., the number of iterations, the population size in a genetic algorithm, the queue management policy in a branch-and-bound algorithm, etc. Another provided class is `Statistics`, whose purpose is collecting statistical information about the execution of the algorithm. Finally, there exist two classes `StateVariable` and `StateCenter` whose necessity and purpose will be described later (in the hybridization interface section).

– **Required Classes**. These classes are responsible for providing details about the problem being solved. It must be noted that despite this problem dependency, the interface of these classes is known and fixed; thus, provided classes can use them without specific knowledge of the problem-dependent features that have been implemented. Among the required classes, one can cite class `Problem` (that must provide all necessary methods for handling data of the problem instance), class `Solution` (that encapsulates the representation and manipulation of a solution for the problem), class `UserStatistics` (that allows the user gathering specific data not being considered by the provided class `Statistics`) as well as other classes that depend on the algorithmic skeleton chosen.

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 labs 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 (commonly infra-utilized in practice).

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 this communication mechanism to be also high-level; besides, maybe in the future would be needed a management of parallel processes (creation, destruction, etc.), and here the new MPI-2 standard could come to the rescue.

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

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 an large number of services has been implemented. These services can be classified into two groups: basic services, and advances services. Among the former one can cite:

- **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.

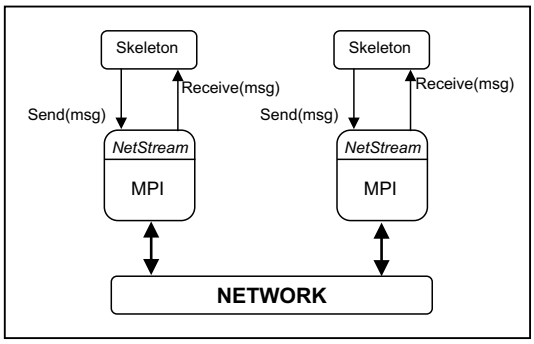

Fig. 3. NetStream communication layer on top of MPI.

- **Synchronization services:** barriers, broadcast, 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.
- **Misc.:** starting and stopping the system (using static (class) methods 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 demes. 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 net:** this C++ methods are provided to allow working with a model of both communication links and the estate 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. 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 [7]). As it has been shown in theory [23] and practice [10], hybridization is in a broad sense an essential mechanism for obtaining effective optimization algorithms for specific domains. For this reason, there exist in MALLBA some basic tools for building such hybrid skeletons. This contrast 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 using 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 estate, 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 estate is that combining skeletons has a low, despite the fact that uniformly defining the state is not trivial, and constitutes an open research topic [9]. Our proposal is articulated around the two basic classes we mentioned before: `StateVariable` and `StateCenter`. The former allows defining and manipulating any information element within the algorithm skeleton. This way, `StateVariable` provides the means for assigning an ID to such elements, as well as for inspecting and updating their value.

All `StateVariable` instances are subsumed within a `StateCenter`. The latter is the connection point that provides access to the state itself. Such access is not performed using specific method for each variable, but using a generic method and an ID as its parameter. The interface is then very generic (there is no dependency either to the problem or to the algorithm) and flexible (state variables can be very easily defined and accessed, even during the execution of the skeleton, and whatever their data types could happen to be).

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 defines the way in which n component skeletons interact each other. One simply has to specify the precise algorithm involved to instance this metha-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 product different algorithm with the same underlying search pattern at a minimum cost.

3 Exact Optimization Techniques

EXACT

4 Heuristic Optimization Techniques

HEURISTIC

5 Hybrid Optimization Techniques

In its broadest sense, hybridization refers to the inclusion of problem-dependent knowledge in a general search algorithm [11] in one of two ways:

- **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.

In this work we have implemented two hybrid algorithms, namely GASA and CHCES; they two are different instances of the *weak hybrid* scheme above 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 a CHC [14] and an evolution strategy (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.

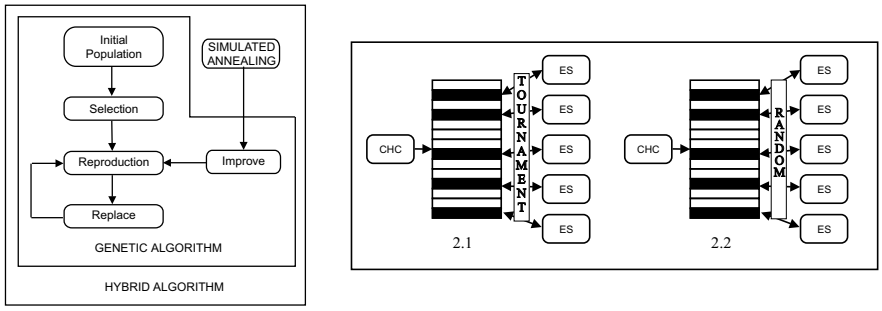

Fig. 4. Models of Hybridization; (left): Model of Hybridization 1 (CHCES1); (right): Model of Hybridization 2 (CHCES2/3).

We define two main classes of hybrids in this work:

– 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. 4 (left).
– A second hybrid schema executes a GA/CHC until the algorithm completely finishes. Then the hybrid selects some individuals from the last 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. 4 right), and another version (GASA3/CHCES3) that uses a random choice of individuals (model 2.2 of Fig. 4 right).

5.1 Hybrid Skeletons for Optimization

In our project, we naturally deal with several optimization techniques or solvers. We have selected to discuss in this work the skeletons for evolutionary algorithms (EAs) -in particular genetic algorithms (GAs)- a CHC algorithm, and an evolution strategy (ES). Also, a local search technique like simulated annealing (SA) has been included due to its widely recognized efficiency in optimization (see a description of all these algorithms in [6]). All techniques have been parallelized for LAN and WAN platforms; a more detailed presentation of them is included in the next sub-sections.

Evolutionary Algorithms Evolutionary algorithms (EAs) are stochastic search techniques that have been successfully applied in many real and complex applications (epistatic, multimodal, 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) [6]. An EA is an iterative technique that applies stochastic operators on a pool of individuals (the population) (see Fig. 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. For the present study we implemented three parallel distributed EAs, whose component sub-algorithm is a GA, an ES or a CHC.

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 also. 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 [14] 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 though 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

```
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
return Best_Solution_Found
```

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

disruptive crossover operator provide more effective search in many problems, which represents the core idea behind the CHC search technique (see a pseudo-code in Fig. 6. This algorithm introduce a new 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.

```
t := 0
d := L/4
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)
       d--
   if d < 0
      diverge P(t)   //restart
      d := r * (1.0 - r) * L
return Best_Solution_Found
```

Fig. 6. Pseudo-code of the CHC algorithm.

The last EA we include in our study is an ES (Fig. 7). This algorithm is suited for continuous values, usually with an elitist selection and a specific mutation (crossover is used rarely). 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.

Simulated Annealing The simulated annealing algorithm (SA) was first proposed in 1983. SA is a stochastic relaxation technique that can be seen as a

```
t := 0
initialize P(t)
evaluate structures in P(t)
while not end do
   t := t + 1
   C(t) := select_best_from(P(t-1))
   mutate structures in C(t) to yield C'(t)
   evaluate structures in C'(t)
   replace P(t) from C'(t) and/or P(t-1)
return Best_Solution_Found
```

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

hill-climber with an internal mechanism to escape local optima (see a pseudo-code in Fig. 8). In SA, the solution s' is accepted as the new current solution if $\delta \leq 0$ 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. Here, we are using a proportional method for updating the temperature ($T_k = \alpha \cdot T_{k-1}$).

```
t := 0
Initialize T
s0 := Initial_Solution()
v0 := Evaluate(s0)
repeat
  repeat
    t  := t+1
    s1 := Generate(s0,T)   //Move
    v1 := Evaluate(s1)
    if Accept(v0,v1,T)
      s0 := s1
      v0 := v1
  until t mod MarkovChainLen = 0
  T = Update(T)
until 'outer-loop stop criterion' satisfied
return s0
```

Fig. 8. Pseudo-code of simulated annealing (SA).

Parallel Hybrid Skeletons Since we want to conduct our research in LAN and WAN platforms, it seems natural to explore the behavior of parallel hybrids. 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 [4]. Different parallel algorithms differ in the characteristics of their elementary skeletons, and in the communication details.

As example parallel skeletons we have chosen a *parallel GA* (PGA) because of its popularity and because it can be readily implemented in clusters of machines. In distributed GAs 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.

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. The hybrid versions applied, in the parallel case, SA (CHC) as an operator in each island of a GA (CHC) algorithm.

These different implementations can be obtained by creating separate subclasses of the `Solver` abstract class (see Fig. 1). At present, we are using our own middleware layer `NetStream` implemented on top of MPI to ease communications. The result is a parallel version of what we will call GASA1 (CHCES1) (Section 5).

Analysis of the 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 second appendix at the end of the paper. We include such four problems in this section to show the really wide application of the hybrid skeletons developed within MALLBA. Our goal with the upcoming results is to compare hybrid versus 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 1 we provide the parameters used for the non-hybrid (basic) skeletons, while in Table 2 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 figures 3, 4 and 5, respectively. All they are the average values of 30 independent runs for each problem, in each one of the three platforms. Since we want a fair com-

Table 1. 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 2. Parameters of the hybrid operators.

Algorithm	Prob.	# max iter.	Others
SA	0.1	100	MarkovChainLen = 10
			Temp. decay factor = 0.99
ES	0.01	50	(1+10)-ES, Mutation prob. = 0.8

parison we begin with a canonical having one workstation in each of the three geographically separated sites.

After these three figures, 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. These are great news for our intended future work on aggregating a high number of computers in WAN.

Now let us compare our figures against existing 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 [15].

Table 3. Average results in the sequential platform.

Problem	Algorithm	opt.	#evaluations	time	hits
MaxCut	GA	1008	33794	68.2	3.3%
	GASA1	1030	48823	96.3	9.9%
MTTP	GA	219	42017	5.8	66.6%
	GASA1	200	38297	5.5	100%
RAS	CHC	0	9634	4.15	100%
	CHCES1	0	14413	4.82	100%
FMS	CHC	30.95	32270	19.65	100%
	CHCES1	20.78	17962	26.62	100%

Table 4. Average results for the LAN platform.

Problem	Algorithm	opt.	#evals	time	hits
MaxCut	GA	1031	23580	49.1	16.6%
	GASA1	1038	40682	89.6	16.6%
MTTP	GA	201	40002	5.2	96.6%
	GASA1	200	25601	5.8	100%
RAS	CHC	0	7591	3.33	100%
	CHCES1	0	13048	3.73	100%
FMS	CHC	30.79	27341	9.2	100%
	CHCES1	16.92	16558	11.45	100%

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.

Table 5. Average results for the WAN platform.

Problem	Algorithm	opt.	#evals	time	hits
MaxCut	GA	1014	14369	89.1	9.9%
	GASA1	1031	28956	298.5	9.9%
MTTP	GA	200	32546	25.7	100%
	GASA1	200	34952	45.62	100%
RAS	CHC	0	7606	133.45	100%
	CHCES1	0	13681	10.0	100%
FMS	CHC	29.29	29743	17.25	100%
	CHCES1	13.41	17816	26.69	100%

Our experience after conducting all 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 powerful hybrid algorithms and a remarkably 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: Spanish CICYT TIC-1999-0754 (MALLBA), MCYT TIC2002-04498-C05-02 (TRACER), EU IST program IST-2001-33116 (FLAGS), Future and Emerging Technologies of EU contract IST-1999-14186 (ALCOM-FT) and Canary Government Project PI/2000-60.

C. León is partially supported by TRACS program at EPCC, M. Blesa is partially supported by Catalan 2001FI-00659 pre-doctoral grant, and G. Luque is partially supported by Andalusian 2002FPDI-43927 grant.

MALLBA is publicly available at `http://www.lsi.upc.es/~mallba`.

A Appendix: Example of a Skeleton Instantiation

In this section we will highlight the main steps in a skeleton instantiation from the point of view of a final user. Such a user would probably want to include his/her problem in an existing skeleton to solve it. As an example let us consider a *Simulated Annealing* heuristic (SA). If such a SA were to be used it would be necessary to define class `Move` defining the neighborhood structure among solutions. In Fig. 9 we show the design for such a SA algorithm. In that design, it appears a set of new classes that are specifically included for the SA technique

(`Move` hierarchy). These new classes allow the user to generate new solutions from the current one in order to search in its neighborhood.

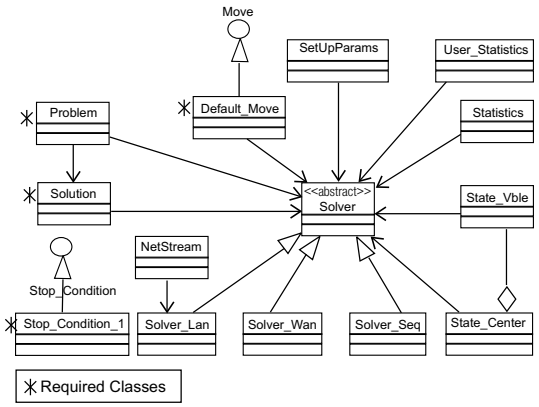

Fig. 9. UML Diagram describing the design of an optimization technique in MALLBA. Lines ending with a triangle denote extension of class `Solver` by inheritance; the other arrows points from the "part" to the "whole" expressing a whole-part aggregation relationship. A diamond means "one or more instances" included, e.g. of state variables in a state center. We mark with an asterisk the user required classes.

Hence, a user of MALLBA will have to implement some data structures to represent the problem considered, as well as providing an actual behavior for all methods included in the interfaces of required classes, according to the previously chosen representation.

A Appendix: Problems

In this section we present the optimization problems that will be used to test our hybrid skeletons. We made a benchmark of XXX optimization tasks, and included in it a complex instance for each one. For testing the algorithms in combinatorial optimization we consider the minimum tardy task problem (MTTP), the error correcting code design problem (ECC), and the maximum cut problem (MaxCut). Our representatives for continuous optimization are the Rastrigin function (RAS) and the frequency modulation sounds (FMS).

The first two problems were chosen because their continuous nature make them adequate for testing the ES algorithm. The first problem is of moderate difficulty. The second one is a highly complex multimodal problem having strong epistasis. The rest of problems represent a broad spectrum of challenging intractable tasks in the areas of scheduling, coding theory, graph theory and transportation. Almost all of them have a direct real world use.

We list the problems alphabetically for the reader's commodity.

The Frequency Modulation Sounds Problem (FMS) The Frequency Modulation Sounds [22] 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 $\boldsymbol{x} = (a_1, w_1, a_2, w_2, a_3, w_3)$ in order $y(t)$ to fit the target $y_0(t)$. The evolved and target models have the expressions shown in Eq. 2 and Eq. 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 value below 0.12.

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

$$y(t) = a_1 \sin\left(w_1 t\theta + a_2 \sin\left(w_2 t\theta + a_2 \sin(w_3 t\theta)\right)\right) \tag{2}$$

$$y_0(t) = 1.0 \sin\left(5.0 t\theta + 1.5 \sin\left(4.8 t\theta + 2.0 \sin(4.9 t\theta)\right)\right) \tag{3}$$

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

The Minimum Tardy Task Problem (MTTP) The minimum tardy task problem is a task-scheduling problem [17]. 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 (Eq. 4).

$$min\ W = \sum_{i \in T - S} w_i \tag{4}$$

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 ([17]) of size 100 task, "mttp100" ($f^* = 20$).

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.

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:

$$F(\boldsymbol{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij}\big(x_i(1-x_j) + x_j(1-x_i)\big) \tag{5}$$

For the experiments reported here, we use a a scalable problem instance ([2]) with a graph of size $n = 100$, "cut100" ($f^* = 1077$).

The Rastrigrin Function (RAS) The generalized Rastrigin function (6) is a problem with a large search space and a very large number of local optima ([20]). 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 $f^* = 0$).

$$Ras(x_i|_{i=1..n}) = 10 \cdot n + \sum_{i=1}^{n}[x_i^2 - 10 \cdot \cos(2 \cdot \pi x_i)] \quad x_i \in [-5.12, 5.12] \tag{6}$$

References

1. E. Alba, C. Cotta, M. Díaz, E. Soler, and J. 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 and S. Khuri. *Sequential and Distributed Evolutionary Algorithms for Combinatorial Optimization Problems*, volume 113 of *Advances in Soft Computing - Hybrid Information Systems*, chapter 10, pages 211–233. Physica-Verlag, Heidelberg, July 2002.
3. E. Alba and the MALLBA Group. MALLBA: A library of skeletons for combinatorial optimisation. In R. F. B. Monien, editor, *Proceedings of the Euro-Par*, volume 2400 of *Lecture Notes in Computer Science*, pages 927–932, Paderborn (GE), 2002. Springer-Verlag.
4. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.
5. M. Arenas, P. Collet, A. Eiben, M. Jelasity, J. Merelo, B. Paechter, M. Preub, and M. Schoenauer. A framework for distributed evolutionary algorithms. In J. J. Merelo, P. Adamidis, H. G. Beyer, J. L. Fernández-Villacañas, and H. P. Schwefel, editors, *Seventh International Conference on Parallel Problem Solving from Nature*, pages 665–675, 2002.
6. T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press, 1997.
7. C. Cotta and J. Troya. On decision-making in strong hybrid evolutionary algorithms. In A. Del Pobil, J. Mira, and M. Ali, editors, *Tasks and Methods in Applied Artificial Intelligence*, volume 1416 of *Lecture Notes in Computer Science*, pages 418–427. Springer-Verlag, Berlin Heidelberg, 1998.
8. B. L. Cun. Bob++ library illustrated by VRP. In *European Operational Research Conference (EURO'2001)*, page 157, Rotterdam, 2001.

9. J. M. Daida, S. J. Ross, and B. C. Hannan. Biological symbiosis as a metaphor for computational hybridization. In L. Eshelman, editor, *Sixth International Conference on Genetic Algorithms*, pages 328–335. Morgan Kaufmann, 1995.
10. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.
11. L. Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
12. L. Di Gaspero and A. Schaerf. EasyLocal++: an object-oriented framework for the flexible design of local search algorithms and metaheuristics. In *4th Metaheuristics International Conference (MIC'2001)*, pages 287–292, 2001.
13. J. Eckstein, C. A. Phillips, and W. E. Hart. Pico: An object-oriented framework for parallel branch and bound. Technical report, RUTCOR, 2000.
14. L. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, 1991.
15. F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–63, 2000.
16. IBM. COIN: Common Optimization INterface for operations research, 2000. http://oss.software.ibm.com/developerworks/opensource/coin/index.html.
17. S. Khuri, T. Bäck, and J. Heitkötter. An evolutionary approach to combinatorial optimization problems. In *22nd ACM Computer Science Conference*, pages 66–73. ACM Press, 1994.
18. K. Klohs. Parallel simulated annealing library. http://www.uni-paderborn.de/fachbereich/AG/monien/SOFTWARE/PARSA/, 1998.
19. D. Levine. PGAPack, parallel genetic algorithm library. http://www.mcs.anl.gov/pgapack.html, 1996.
20. A. Töorn and Ž. Antanas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1989.
21. S. Tschke and T. Polzer. Portable parallel branch-and-bound library, 1997. http://www.uni-paderborn.de/cs/ag-monien/SOFTWARE/PPBB/introduction.html.
22. S. Tsutsui, A. Ghosh, D. Corne, and Y. Fujimoto. A real coded genetic algorithm with an explorer and an exploiter populations. In T. B. et al., editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 238–245. Morgan Kaufmann, 1997.
23. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.