Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 30 (2004) 611–628
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Parallel LAN/WAN heuristics for optimization

E. Alba *, G. Luque, J.M. Troya

Departamento de Lenguajes y Ciencias de la Computación, E.T.S. Ingeniería Informática, Campus de Teatinos (3.2.12), 29071 Málaga, Spain

Received 10 November 2003; accepted 15 December 2003
Available online 18 May 2004

Abstract

We present in this work a wide spectrum of results on analyzing the behavior of parallel heuristics (both pure and hybrid) for solving optimization problems. We focus on several evolutionary algorithms as well as on simulated annealing. Our goal is to offer a first study on the possible changes in the search mechanics that the algorithms suffer when shifting from a LAN network to a WAN environment. We will address six optimization tasks of considerable complexity. The results show that, despite their expected slower execution time, the WAN versions of our algorithms consistently solve the problems. We report also some interesting results in which WAN algorithms outperform LAN ones. Those results are further extended to analyze the behavior of the heuristics in WAN with a larger number of processors and different connectivities.

Keywords: Evolutionary algorithm; Hybridization; Parallel heuristic; LAN; WAN

1. Introduction

Solving complex problems means designing new algorithms that improve in some manner the computational effort and the execution time for an acceptable solution. Analyzing and designing parallel algorithms is a healthy activity since it pursuits some of the most promising objectives in optimization, namely reducing the

* Corresponding author.
E-mail addresses: eat@lcc.uma.es (E. Alba), gabriel@lcc.uma.es (G. Luque), troya@lcc.uma.es (J.M. Troya).

0167-8191/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2003.12.007

wall-clock time, allowing cooperation of different algorithms (hybridization), and even modifying the search pattern to yield new methods.

In this work, we study the way in which several parallel algorithms change their behavior when executed in a LAN with respect to their execution in a WAN. Due to the growing park of available computers in the Internet, our aim is to study the new scenario of WAN computing, in contrast to the traditional LAN analysis.

The MALLBA project [1] is an effort of research in this direction. In the MALLBA project we intend to design and analyze exact, heuristic and hybrid techniques in sequential, LAN and WAN environments. MALLBA source code and latest achievements are used in this work.

The contributions of this paper are manifold. First, we test the efficiency of some algorithms developed in the MALLBA project, since the achievements mentioned in the last paragraph represent an ambitious goal that must be validated in practice. Second, we want to put aside the expected and actual outcomes of computing in LAN and WAN. Third, we are interested in showing really useful results, and this is why we check the algorithms on six quite different problems, accounting for epistasis and multimodality, both in continuous and discrete optimization. Our claims and conclusions are somewhat expected and surprising at the same time, since we do validate in practice some theoretical thoughts on the induced WAN overhead, but, at the same time, we report competitive performance in WAN in some cases.

The organization of the paper is as follows. Next section (Section 2) introduces the MALLBA project. Section 3 discusses the search models considered in our study. Section 4 presents the details on the problems being solved. We then turn in Section 5 to comparatively analyze the algorithms in different LAN/WAN scenarios. Finally, we provide in Section 6 some concluding remarks, and point out the future work we envision after our conclusions.

2. The MALLBA project

The MALLBA [1] research project is aimed at developing 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 described in the next section are implemented as *software skeletons* (similar to the concept of software pattern [2]) with a common internal and public interface. This permits fast prototyping and transparent access to parallel platforms.

MALLBA skeletons distinguish between the concrete problem to be solved and the solver technique. Skeletons are generic templates to be instantiated by the user with the features of the problem. All the knowledge related to the solver method (e.g., parallel considerations) and its interactions with the problem are implemented by the skeleton and offered to the user. Skeletons are implemented by a set of

[1] The MALLBA library is publicly available at http://neo.lcc.uma.es/mallba/easy-mallba/index.html.

required and *provided* C++ classes that represent an abstraction of the entities participating in the solver method:

- *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).
- *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.

Therefore, the user of a MALLBA skeleton only needs to implement the particular features related to the problem. This speeds considerably the creation of new algorithms with a minimum effort, especially if they are built up as combinations of existing skeletons (*hybrids*).

For example, in Fig. 1 we show the design for a simulated annealing (SA). In that design, we define a set of new classes that are specifically included for the SA method (`Move` hierarchy). These new classes allow the user to generate new solutions from the current one in order to search in its neighborhood.

The infrastructure used in the MALLBA project is made of communication networks and clusters of computers located at the spanish universities of Málaga, La Laguna and UPC in Barcelona. These nodes are interconnected by a chain of Fast Ethernet and ATM circuits.

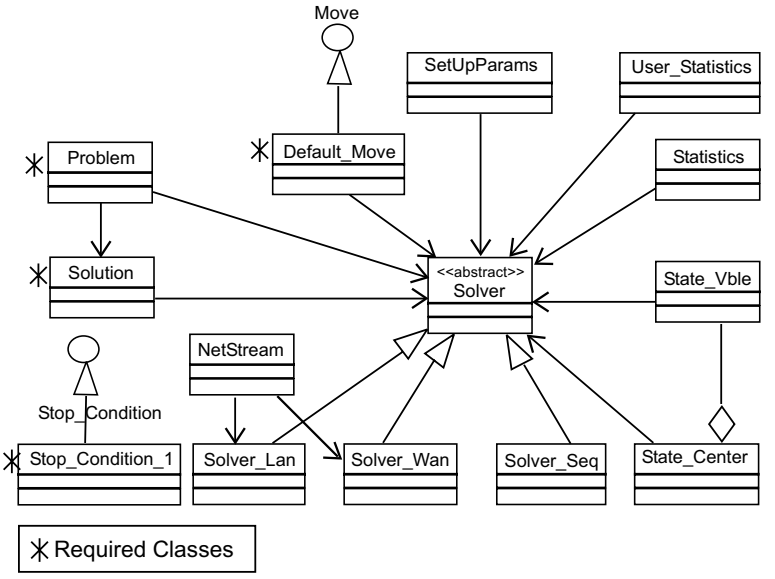

Fig. 1. UML Diagram describing the design of the SA Method.

3. Algorithms

In this paper we deal with several evolutionary algorithms, in particular with genetic algorithms (GAs), a CHC algorithm, and an evolution strategy (ES). Local search methods are also considered in this work, like simulated annealing (SA) (see a detailed description of these techniques in [3]). All methods have been parallelized for LAN and WAN platforms. Finally, we will also include some hybrid algorithms in our LAN/WAN study.

Let us now proceed to explain the optimization techniques considered here.

3.1. Evolutionary algorithms

Evolutionary algorithms (EAs) are stochastic search methods that have been successfully applied in many real applications of high complexity. An EA is an iterative technique that applies stochastic operators on a pool of individuals (the population) in order to improve the average fitness ($\bar{f}_i < \bar{f}_{i+1}$). Every individual in the population is the encoded version of a tentative solution. Initially, this population is randomly generated. An evaluation function associates a fitness value to every individual indicating its suitability to the problem. We have implemented three separate parallel distributed EAs, whose component subalgorithm 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 other types of representations can be found. A GA usually applies a recombination operator on two tentative solutions, plus a mutation operator that randomly modifies the individual contents to promote diversity.

A CHC [4] is a non-canonical GA which combines a conservative selection strategy (that always preserves the best individuals found so far) with a highly disruptive recombination (HUX), that produces offsprings that are maximally different from their two parents. The traditional thought of preferring a recombination operator with low disrupting properties may not hold when such a conservative selection strategy is used. On the contrary, certain highly disruptive crossover operator provides more effective search in many problems, which represents the core idea behind the CHC search method. This algorithm introduces a bias against mating individuals who are too similar (*incest prevention*). Mutation is not performed; instead, a *restart* process reintroduces diversity whenever convergence is detected.

The last EA that we consider is an ES. This algorithm is suited for continuous optimization, usually with an elitist selection and a specific mutation (crossover is used rarely). In ES, the individual is composed of the objective float variables plus some other parameters guiding the search. Thus, an 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.

3.2. Simulated annealing

The simulated annealing algorithm (SA) was first proposed in 1983. SA is a stochastic search technique that can be seen as a hill-climber with an internal mecha-

nism to escape from local optima. 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. Here, we are using a proportional method for updating the temperature ($T_k = \alpha \cdot T_{k-1}$). The α factor indicates the decrease speed of the temperature.

3.3. Hybrid algorithms

In its broadest sense, hybridization [5] refers to the inclusion of problem-dependent knowledge in a general search algorithm in one of two ways:

- *Strong hybrids*: problem-knowledge is included as specific non-conventional problem-dependent representations and/or operators.
- *Weak hybrids*: 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. The first of them (GASA) is made of a genetic algorithm and a simulated annealing; also, this scheme has been used to combine a CHC and an 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 deal with two main subclasses of weak hybrids in this work:

- A first hybrid (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. 2 (left).
- The 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. 2 (right)), and another version (GASA3/CHCES3) that uses a random choice of individuals (model 2.2 of Fig. 2 (right)).

3.4. Parallel heuristics

Since we want to conduct our research in LAN and WAN platforms it seems natural to explore the behavior of parallel heuristics. A parallel EA (PEA) is an algorithm having multiple component EAs, regardless of their population structure.

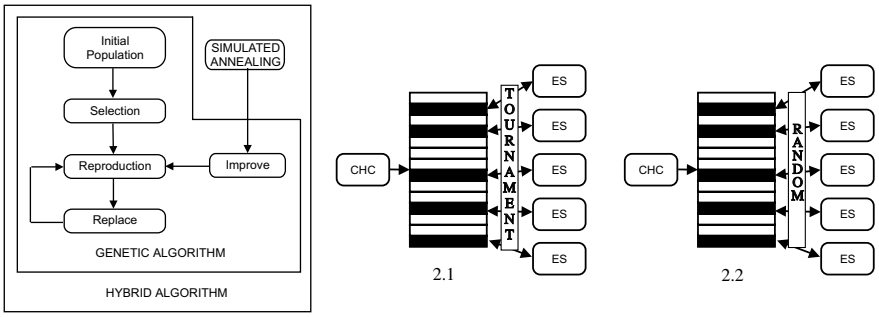

Fig. 2. Models of hybridization: (left) model of hybridization 1 (GASA1) and (right) two variants of model of hybridization 2 (CHCES2/3).

Each component (usually a traditional (single population) EA) subalgorithm includes an additional phase of *communication* with a set of subalgorithms [6].

Different parallel algorithms differ in the characteristics of their elementary heuristics and in the communication details. In this work, we have chosen a *distributed EA* (dEA) because of its popularity and because it can be easily implemented in clusters of machines. In distributed EAs there exists a small number of islands performing separate EAs, 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 subpopulations, and the kind of integration of exchanged individuals within the target subpopulations. Concretely, we use a static ring topology, select random migrants, and include them asynchronously in the target populations only if they are better than the local worst-existing solutions.

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

4. Problems

In this section, we discuss the optimization tasks that will be used to test our parallel heuristics. We made a benchmark of six optimization tasks, considering a complex instance for each one. We have selected problems both from discrete and continuous domains of research. Our representatives for continuous optimization are the Rastrigin function (RAS) and the frequency modulation sounds problem (FMS). For testing the algorithms in combinatorial optimization we consider the minimum tardy task problem (MTTP), the error correcting code design problem (ECC), the maximum cut problem (MaxCut), and the vehicle routing problem (VRP). For the VRP we include two instances.

The first two problems were chosen because their continuous nature makes 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.

4.1. The Rastrigin function (RAS)

The generalized Rastrigin function (Eq. (1)) is a problem with a large search space and a very large number of local optima [7]. 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 (optimum at $f_{\mathrm{RAS}}(x^*) = 0$).

$$f_{\mathrm{RAS}}(\vec{x}) = 10 \cdot n + \sum_{i=1}^{n} x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i), \quad x_i \in [-5.12, +5.12] \tag{1}$$

4.2. The frequency modulation sounds problem (FMS)

The frequency modulation sounds problem [8] (or FMS) 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. (2).

$$f_{\mathrm{FMS}}(\vec{x}) = \sum_{i=0}^{100} (y(t) - y_0(t))^2 \tag{2}$$

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. (3) and (4).

$$y(t) = a_1 \cdot \sin(w_1 \cdot t \cdot \theta + a_2 \cdot \sin(w_2 \cdot t \cdot \theta + a_3 \cdot \sin(w_3 \cdot t \cdot \theta))) \tag{3}$$

$$y_0(t) = 1.0 \cdot \sin(5.0 \cdot t \cdot \theta - 1.5 \cdot \sin(4.8 \cdot t \cdot \theta + 2.0 \cdot \sin(4.9 \cdot t \cdot \theta))) \tag{4}$$

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

The resulting problem is a highly complex multimodal function having strong epistasis with minimum value $f_{\mathrm{FMS}}(x^*) = 0$. For the experiments, we consider as an optimum any solution with fitness value below 0.12.

4.3. The maximum cut problem (MaxCut)

The maximum cut problem [9] (MaxCut) 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 cut problem consists in partitioning V into two disjoint sets V_0 and V_1 such that the sum of weights of edges from E that have one endpoint in V_0 and the other in V_1 is maximized. The function to be maximized is

$$f_{\text{MaxCut}} = \sum_{i \in V_0} \sum_{j \in V_1} w_{ij} \tag{5}$$

where w_{ij} is the weight of edge (i, j). For the experiments reported here, we use a scalable problem instance [9] with a graph of size $n = 100$, "cut100" (optimum at $f_{\text{MaxCut}}(x^*) = 1077$).

4.4. The minimum tardy task problem (MTTP)

The minimum tardy task problem is a task-scheduling problem [9]. 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 remains 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, which is the sum of weights of incompleted tasks (Eq. (6)).

$$f_{\text{MTTP}} = \sum_{i \in T - S} w_i \tag{6}$$

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

For our experiments, we use a scalable problem instance [9] of size 100 tasks, "mttp100" (optimum at $f_{\text{MTTP}}(x^*) = 200$).

4.5. The error correcting code design problem (ECC)

The error correcting code design problem [10] (ECC) consists in assigning codewords to an alphabet that minimizes the length of transmitted messages, and that provides maximal correction of single uncorrelated bit errors when the messages are transmitted over noisy channels. This problem can be formally represented by a three-tuple (n, M, d), where n is the length of each codeword, M is the number of codewords and d is the minimum Hamming distance between any pair of codewords. The function to be maximized is

$$f_{\text{ECC}}(C) = \frac{1}{\sum_{i=1}^{M} \sum_{j=1, i \neq j}^{M} d_{ij}^{-2}} \tag{7}$$

where d_{ij} represents the Hamming distance between codewords i and j.

In this study, we consider a problem instance, where $n = 12$ and $M = 24$. The optimum is achieved at fitness of $f_{\text{ECC}}(x^*) = 0.0674$.

4.6. The vehicle routing problem (VRP)

The vehicle routing problem [11] (VRP) consists in determining minimum cost routes for a fleet of vehicles originating and terminating in a depot. The fleet of vehi-

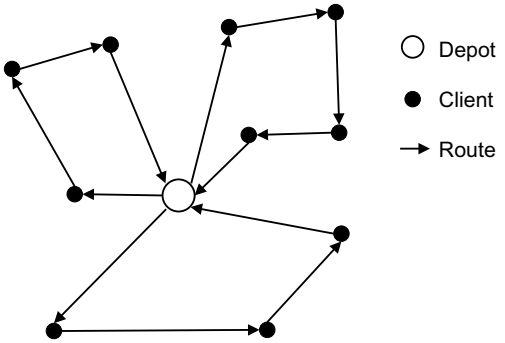

Fig. 3. Example of the vehicle routing problem.

cles gives service to a set of customers with a known set of constrains. All customers must be assigned to vehicles such that each customer is serviced exactly once, taking account for the limited capacity of each vehicle (Fig. 3).

The goal is to minimize the sum of route costs given by

$$f_{\mathrm{VRP}}(\vec{x}) = \sum_{k=0}^{K-1} \alpha(W_k(\vec{x})) \tag{8}$$

$$W_k(\vec{x}) = \sum_{i=0}^{N} \sum_{j=0}^{N} c_{ij} \cdot x_{ijk} \tag{9}$$

In these equations, x_{ijk} is a binary variable that is 1 if route k has an arc between nodes i and j, c_{ij} is the cost to go from node i to node j, α is the function that calculates the route cost depending on W_k, N is the number of clients (0 is the depot), and K is the number of routes.

A feasible solution must satisfy that all clients must be visited (exactly once), routes begin and finish in depot and have no cycles, and routes cannot exceed the vehicle capacity.

For the experiments, we use the two first instances of the classical benchmark after Christofides [12]. We consider as an optimum any solution with $f_1^* < 800$ for the first instance, and $f_2^* < 1200$ for the second one. We must notice that the two instances of VRP are very difficult to solve without including specific operations. See a complete and detailed on-line description of VRP variants at `http://neo.lcc.uma.es/radi-aeb/WebVRP/index.html`.

5. Analysis of the results

In this section we describe and discuss the results after running the algorithms over the whole test suite. We will proceed in three phases. First, we analyze the

behavior of a parallel GA in LAN and WAN environments. In this phase check the performance of a parallel GA to solve combinatorial optimization problems (MaxCut, MTTP, ECC and VRP). Later, we include a second phase of study that addresses the continuous optimization domain. In this second phase we analyze a parallel evolution strategy on our two continuous problems (RAS and FMS). The final third phase has the goal of validating the achieved conclusions, but this time by considering hybrid parallel algorithms (CHCES variants on RAS and GASA variants on MaxCut). We applied these phases in two cases; first, we compare the algorithms in LAN and WAN environments, and later, we use these stages to study the behavior of the heuristics in two different WAN platforms.

The algorithm parameters used in the experiments are described in Table 1. Table 2 shows the hardware and software configuration. All the tests perform 30 independent runs. We report average values of successful executions, i.e., executions that reach the target fitness. We show values for the average best solution fitness found (*avg opt*), average number of evaluations to get an optimal solution (#*evals*), average time (*time*, in seconds), the percentage of hits (*hits*), i.e., the relative number of runs finding an optimal solution, and the statistical *t*-test confidence value for evaluations and times (*p-value*).

We have organized this section into two subsections. In the first one we analyze the changes in the search mechanics that the algorithms suffer when shifting from

Table 1
Parameters of algorithms

Problem	Popsize	P_c	P_m	Others
MaxCut, VRP (GA)	100	0.8	0.01	–
MTTP, ECC (GA)	200	0.8	0.02	–
RAS, FMS (ES)	100	0.3	0.80	–
MaxCut (GASA)	100	0.8	0.01	SA operator applied with prob 0.1, MarkovChainLen = 10, 100 iterations
RAS (CHCES)	100	0.8	–	35% population restart, (1 + 10)-ES operator (prob 0.01)

Table 2
Hardware and software configuration of the WAN

	MA-cluster	LL-cluster	BA-cluster
Processor	PIII, 500 MHz	AMD Duron, 800 Mhz	AMD K6-2, 450 Mhz
Main Memory (MB)	128	256	256
Hard Disc (GB)	4	18	4
Linux Kernel	2.4.19-4	2.2.19	2.4.19
g++ version	3.2	2.95.4	2.95.4
MPICH version	1.2.2.3	1.2.2.3	1.2.2.3

a LAN network to a canonical WAN environment, while in the second one we study the behavior of the heuristics in two different combined LAN/WAN platforms (as an extension to the work available at [13]).

5.1. LAN and WAN analysis

In this section we perform a comparative analysis of the behavior of the canonical *LAN* and *WAN* versions of every algorithm. For the *WAN* experiments, we use one machine from each MALLBA site (three in total) as it is shown in Fig. 4. For the LAN tests, these three machines belong to the MA-cluster.

In Table 3 we include the results of applying a parallel GA on the four combinatorial problems of our benchmark. If we compare *LAN* and *WAN* executions of the parallel GA we can detect a clear trend of the *WAN* execution to spend more time in finding a solution for the three first problems with respect to the *LAN* times.

An interesting detail is that the parallel GA seems to increase the number of hits when executed in *WAN* for MTTP and ECC, although, as expected, the wall-clock time for these problems and for MaxCut is higher in *WAN* due to the communications overhead. Numerically speaking, this means that the *WAN* environment retards the migrations, and this effect is somewhat perceptible in some problems. In fact, *WAN* reduces the number of evaluations for three out of five instances, but

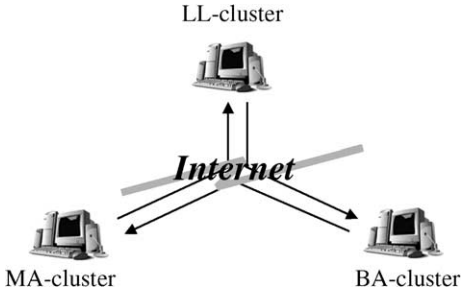

Fig. 4. *WAN* configuration.

Table 3
Average results for a parallel GA

	LAN				*WAN*				*p*-value	
	Avg. opt.	#Evals	Time	Hits (%)	Avg. opt.	#Evals	Time	Hits (%)	Eval	Time
MaxCut	1031	23580	49.1	17	1014	14369	89.1	10	0.91	8.13e-9
MTTP	201	40002	5.2	97	200	32546	25.7	100	0.09	3.7e-10
ECC	0.0642	18279	9.1	7	0.0657	26041	238.4	10	0.43	1.1e-14
VRP 1	696.15	12843	78.9	100	690.693	7168	68.5	100	0.92	4.78e-4
VRP 2	1080.67	5873	391.1	100	1077.83	8104	358.7	100	0.73	4.99e-6

the t-tests (0.05 significance level) reveal that the differences in the numerical effort is not significant. The conclusion is then that *LAN* and *WAN* are similar in terms of effort, for all the problems in the benchmark. Since larger isolation is not always appropriate for an arbitrary problem, we could think about a poorer *WAN* numerical results for other problems, as it effectively happens for the MaxCut problem, because its percentage of hits decays from 17% in *LAN* to a smaller 10% in *WAN*. Clearly, the large isolation times induced by the *WAN* in our asynchronous parallel GA is beneficial for some problems and harmful for others.

However, in the case of VRP, although *LAN* and *WAN* work out numerically similar results, the *WAN* execution times are smaller for the two instances. We did got statistical significance for these results, what effectively makes us to claim that the *WAN* execution is faster than the *LAN* one in this problem. This is a very promising result; it can be explained because of the considerably larger grain of the VRP problem with respect to the other ones, and it is due to the longer actual isolation provided by the WAN for the parallel GA islands, which results advantageous for VRP. These two reasons (larger grain and large isolation) change the search pattern followed by the parallel GA from *LAN* with respect to *WAN*, with an improved time in this last platform.

We now proceed to the second phase (continuous optimization) in which we analyze the behavior of a ring of parallel evolution strategies running in parallel to solve RAS and FMS. The results are shown in Table 4. We can see that the conclusions we made for the parallel GA also hold in the case of the parallel ES. This agreement can be noticed in that the execution times of the *WAN* version of parallel ES are higher than the *LAN* ones. Again, like with the parallel GA, the final fitness (*avg opt* column) are approximately the same for *LAN* and *WAN*. Besides, we show that the numerical effort is almost the same in the *LAN* and *WAN* cases, what is an indicator of correct implementation, and validates our claim of similarity in their numerical *LAN*/*WAN* behavior.

Finally, in the third phase, we want to check our findings in a new arena, i.e., on hybrid parallel algorithms. In Table 5 we include the results for RAS (three upper rows) and MaxCut (three lower rows). We selected them as representatives of the class of continuous and combinatorial problems. This is why we use the CHCES hybrid for RAS and the GASA hybrid for MaxCut. After these results, we can state that the precedent conclusions we got for pure algorithms (parallel GA and parallel ES) hold also for hybrid techniques. It seems that all the algorithms present a higher time when running in *WAN* and need a statistically similar number of iterations to

Table 4
Average best results for a parallel ES Algorithm

	LAN				*WAN*				*p*-value	
	Avg. opt.	#Evals	Time	Hits (%)	Avg. opt.	#Evals	Time	Hits (%)	Eval	Time
RAS	0	10763	5.7	97	0	11546	67.5	93	0.171	2.20e-16
FMS	0.099	10904	4.7	100	0.093	11372	13.0	100	0.450	3.26e-5

Table 5
Average best results of all experimental runs performed by the hybrid algorithms

	LAN				*WAN*				*p*-value	
	Avg. opt.	#Evals	Time	Hits (%)	Avg. opt.	#Evals	Time	Hits (%)	Eval	Time
CHCES1	0	13048	3.7	100	0	13681	10.0	100	0.09	8.62e-6
CHCES2	0	8093	3.4	100	0	8593	7.2	100	0.85	3.97e-5
CHCES3	0	8182	3.4	100	0	9635	7.9	100	0.13	7.9e-11
GASA1	1038	40682	89.6	17	1031	28956	298.5	10	0.53	2.97e-5
GASA2	1031	19477	45.7	7	1024	17412	123.5	8	0.22	0.0029
GASA3	1038	21651	43.8	8	1021	12162	202.9	7	0.75	0.0344

yield the optimum. The *LAN* behavior can be assumed then to fit the *WAN* environment from a numerical point of view for most problems.

We conclude this section with a summary of the results. We have shown that, whatever the basic search method is used (GA, ES or hybrid), an execution in a *WAN* environment presents an equivalent numerical result, while showing a natural larger time to achieve a solution with respect its *LAN* execution. Of course, in case of really high number of processes this could not hold; this question remains open, and it has a wide relevance in the case of grid computing platforms, but not for a moderate or low number of subalgorithms (as many researchers use in practice).

The surprising result appears when analyzing the VRP problem, because the *WAN* execution reduces the search time with respect to the *LAN* execution, what seems a counter-intuitive conclusion. We explain this scenario because of the considerable computation time needed by the fitness function (which makes more similar the *LAN* and *WAN* search). Also, because of the larger isolation time induced in practice in the *WAN* platform (although we set the migration frequency to be equal in *LAN* and *WAN*) that, for this problem, represents a more efficient parameterization. This is, of course, a unique feature of asynchronous heuristics, since migration frequency in synchronous heuristics does not admit any numerical advantage in the *WAN*; only the time would be affected in LAN/WAN studies of synchronous heuristics.

5.2. Extensions to the previous WAN analysis

In this section we continue on our study by extending the results of the previous section in two new scenarios. Since our previous *WAN* platform only included three processors separated by the Internet to provide a canonical baseline of comparison, one can wonder about the influence of the number of processors and their connectivity. Therefore, we analyze in this section the behavior of the heuristics in two new environments; *WAN1* (Fig. 5a), and *WAN2* (Fig. 5b). In *WAN1* we maximize the number of links to the Internet in our logical ring of machines (continuous neighbors in separate sites), thus, every link lies in the WAN. In *WAN2* we split the ring into two parts, each one executed in a separate LAN, with only two links through the

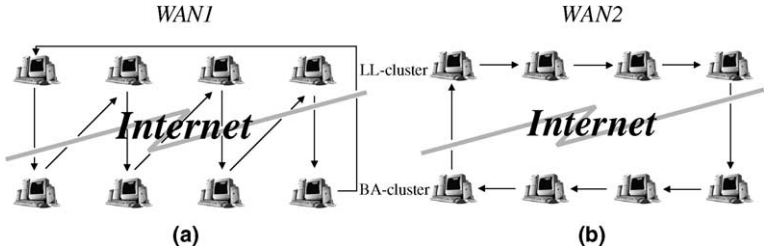

Fig. 5. WAN configurations used: (a) *WAN1* configuration and (b) *WAN2* configuration.

WAN. *WAN1* and *WAN2* use eight machines, four machines from the LL-cluster and other four ones from the BA-cluster.

We follow the same organization that in Section 5.1. First, we analyze the behavior of parallel GA; later, we study the parallel ES results, and, finally, we examine hybrid versions.

Table 6 shows the results when using a parallel GA. A clear first conclusion that may be drawn from this table is that the *WAN2* execution is most times much more efficient than the *WAN1*. *WAN2* minimizes the number of messages over the slowest communication link (i.e., over the Internet) and therefore, it reduces the communication overhead provoking a smaller overall runtime.

The two platforms induce a similar number of hits, and only in ECC we have noticed a significant increase in the hits rate when executed in the *WAN2*. Consequently, this table shows that the numerical effort is quite similar in the *WAN1* and *WAN2* cases.

If we check these results against the initial *LAN* and *WAN* configurations, we can observe that *WAN1* and *WAN2* improve on the previous ones numerically (Table 3) for MaxCut and VRP problems, since *WAN1* and *WAN2* executions obtain a better final average fitness than in the *WAN* case. If we compare the execution times, we see that the experiments of this subsection improve the runtimes (except for ECC instance). This improvement is due to the increased number of processors; overall, results on heterogeneous hardware and software platforms are difficult to analyze and compare, but we must deal with heterogeneity, since it is the intrinsic nature of the Internet.

Table 6
Average results for a parallel GA

	WAN1				*WAN2*				*p*-value	
	Avg. opt.	#Evals	Time	Hits (%)	Avg. opt.	#Evals	Time	Hits (%)	Eval	Time
MaxCut	1041	14246	37.1	17	1040	12283	21.0	17	0.14	1.52e-4
MTTP	200	32886	21.7	100	201	35326	7.1	97	0.47	4.31e-7
ECC	0.0653	26603	423.1	10	0.0659	23243	117.2	17	0.22	2.7e-10
VRP 1	662	13160	55.5	100	657	17066	32.1	100	0.16	0.246
VRP 2	1017	7166	186.3	100	1011.1	10604	144.9	100	0.17	0.0243

Now, in the second phase we analyze the results of parallel ES (Table 7). The experiments with the parallel ES confirm the conclusions obtained in the previous parallel GA case of this subsection. That is, *WAN2* is clearly faster than *WAN1*, and they two obtain the same number of hits with a similar number of evaluations. As we expected, these results reduce the execution time with respect to the *WAN* times obtained in Section 5.1 for parallel ES since we have increased the number of processors. From a numerical point of view, these executions do not present significant differences with the previous ones. The statistical analysis shows that in general, the *WAN2* times are smaller than the *WAN1* ones, while the differences in numerical effort show a trend to present *WAN2* as numerically better (just a trend, since none of the *eval* *t*-tests are positive).

Finally, in the last phase we study the results with hybrid algorithms (Table 8). These results also confirm the conclusions obtained in previous experiments; that is, *WAN2* is faster than *WAN1*, and the difference in numerical effort is not significant in any case (*t*-tests are negative).

The results of this table improve the ones showed in Table 5 (Section 5.1). The execution times of the experiments in this subsection are smaller than the canonical *LAN* and *WAN*. In addition, in the MaxCut instance (three lower rows), these results improve the average best fitness and the hit rates with respect to the canonical *WAN* and *LAN* ones.

Let us now summarize the results. Here, we have checked the behavior of the parallel heuristics in two extended WAN scenarios using eight processors. The Fig. 6 condenses the runtime executions of all problems and scenarios. The split ring

Table 7
Average best results for a parallel ES algorithm

	WAN1				*WAN2*				*p*-value	
	Avg. opt.	#Evals	Time	Hits (%)	Avg. opt.	#Evals	Time	Hits (%)	Eval	Time
RAS	0	9213	38.3	100	0	9407	3.1	100	0.887	2.2e-16
FMS	0.089	13772	11.6	100	0.09	10727	7.2	100	0.089	1.5e-4

Table 8
Average best results of all experimental runs performed by the hybrid algorithms

	WAN1				*WAN2*				*p*-value	
	Avg. opt.	#Evals	Time	Hits (%)	Avg. opt.	#Evals	Time	Hits (%)	Eval	Time
CHCES1	0	14587	6.2	100	0	12091	1.9	100	0.23	0.0039
CHCES2	0	10126	4.8	100	0	8520	2.9	100	0.12	2.2e-11
CHCES3	0	8826	4.1	100	0	9013	1.8	100	0.83	2.5e-12
GASA1	1046.5	37873	92.9	17	1046.6	39753	57.4	17	0.15	0.036
GASA2	1046.8	24066	68.6	29	1052.4	25446	24.4	24	0.32	3.1e-09
GASA3	1045.7	25213	76.2	24	1049	23820	36.4	24	0.19	6.4e-05

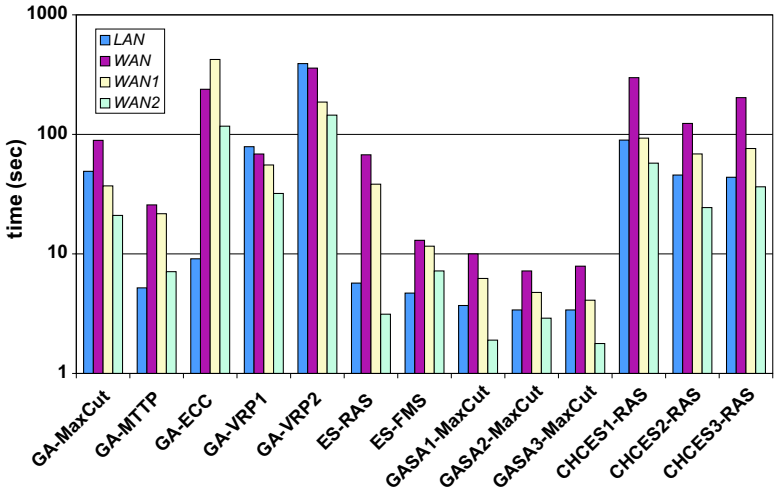

Fig. 6. Runtimes of all problems and scenarios.

topology of *WAN2* represents a configuration which reduce the communications overhead and, consequently, decreases the global runtime with respect to the alternated execution featured in *WAN1*. Also, we have shown that *WAN1* and *WAN2* produce similar numerical results than a canonical *WAN* environment, but with an important reduction in the runtime. The *WAN2* configuration allows us to take advantage of geographically distributed clusters without inducing important overheads in the runtime.

6. Concluding remarks and future work

In this work we have provided a clear seminal evidence to create a body of knowledge that will lead us from our understanding of LAN parallel optimization heuristics to their WAN execution. Our aim is to point out some important facts and to explain them. We have selected a non-trivial benchmark for which a WAN execution comes as a natural research scenario.

The conclusions of this work can be summarized attending to different criteria. With respect to the numerical behavior, *LAN* and *WAN* results seem to be very similar, and this similarity is readily stable. This is very interesting since this will allow us to reduce our discussion to another criterion: wall-clock time execution. With this goal in mind, we have found that *WAN* versions are consistently slower than *LAN* ones, as one could expect. However, the harder the problem, the more similar *LAN* and *WAN* algorithms become.

In the limit (i.e., in our harder instances) we have found that *WAN* executions could be even more efficient than *LAN* ones. This is by no means a question of faster execution, since *WAN* has always higher latencies and delays; it is an effect of isola-

tion. We hypothesize that running these algorithms in WAN is somewhat similar to running them in a LAN cluster but with higher isolation times among the subalgorithm components. We confirmed this hypothesis on some of the problem instances. For some complex problems, large isolation times promote alternating phases of exploration and exploitation in a natural manner, which yield an algorithm of higher efficiency, since isolation time is a well-known influent parameter in distributed parallel algorithms (e.g. see [14] for multipopulation GAs).

We have also extended the whole study to consider more processors and different mappings of logical-to-physical links between neighbor algorithms. The *WAN2* configuration allows us to reduce the execution times in WAN environments and, at the same time, maintains the same numerical behavior than *WAN* and *WAN1* scenarios.

As a future work we plan to quantify the relationship between the isolation time imposed by the WAN network and the isolation time in a LAN cluster, and to check this result on additional problems and algorithms.

Acknowledgements

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

References

[1] E. Alba, The MALLBA Group, MALLBA: A Library of Skeletons for Combinatorial Optimisation, in: B. Monien, R. Feldmann (Eds.), Proceedings of the Euro-Par, in: LNCS, vol. 2400, Springer-Verlag, Paderborn (GE), 2002, pp. 927–932.
[2] T. Römke, J.P. i Silvestre, Programming Frames for the Efficient Use of Parallel Systems, Technical Report LSI-97-9-R, Universidad Politécnica de Barcelona (1997).
[3] Z. Michalewicz, D. Fogel (Eds.), How to Solve It: Modern Heuristics, Springer, 2000.
[4] 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.
[5] L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991.
[6] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (5) (2002) 443–462.
[7] A. Töorn, A. Žilinskas, Global Optimization, in: Lecture Notes in Computer Science, 350, Springer, Berlin, Germany, 1989.
[8] S. Tsutsui, Y. Fujimoto, Forking genetic algorithms with blocking and shrinking modes, in: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann, 1993, pp. 206–213.
[9] S. Khuri, T. Bäck, J. Heitkötter, An evolutionary approach to combinatorial optimization problems, in: Proceedings of the 22nd ACM Computer Science Conference, ACM Press, Phoenix, Arizona, 1994, pp. 66–73.
[10] A.E. Gamal, L. Hemachandra, I. Shperling, V. Wei, Using simulated annealing to design good codes, IEEE Transactions on Information Theory 33 (1) (1987) 116–123.
[11] G. Laporte, M. Gendreau, J.-Y. Potvin, F. Semet, Classical and modern heuristics for the vehicle routing problem, International Transactions in Operational Research 7 (2000) 285–300.

[12] N. Christofides, A. Mingozzi, P. Toth, The Vehicle Route Problem in Combinatorial Optimization, John Wiley & Sons, 1979 (Chapter 11).
[13] E. Alba, G. Luque, Parallel LAN/WAN heuristics for optimization, in: Proceedings of the IPDPS-NIDISC'03, Nize, France, 2003, p. 147.
[14] E. Alba, J. Troya, Influence of the migration policy in parallel distributed GAs with structured and panmictic populations, Applied Intelligence 12 (3) (2000) 163–181.