MALLBA: A library of skeletons for combinatorial optimisation*

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] 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 resolution methods: exact, heuristic and hybrid. Moreover, for each resolution method, MALLBA provides three different implementations: sequential, parallel for local area networks, and parallel for wide area networks (currently under development). This paper shows the architecture of the MALLBA library, presents some of its skeletons and offers several computational results to show the viability of the approach.

1 Introduction

Combinatorial optimization problems arise in various fields such as control theory, operational research, biology and computer science. Exact methods like *divide and conquer*, *branch and bound* and *dynamic programming* have been traditionally used to solve these problems, however, the computational requirements of these methods may be prohibitive. As an alternative, heuristic methods usually provide good solutions in reasonable running times. *Local search*, *spectral techniques* and *evolutionary algorithms* are well known heuristic methods.

A key feature of all the previously mentioned methods is their genericity: they can be easily adapted to solve different problems because they can be described as search *skeletons*, which are general optimization procedures. In this case, the quality of the obtained solutions depends both on the running time and on the amount of problem-dependent knowledge inserted into the algorithm (hybridization) [4, 17].

Parallel computing systems offer a way to provide the large computing power necessary for handling the increasing complexity of combinatorial optimization

* http://www.lsi.upc.es/∼mallba. Work partially supported by: Spanish CICYT TIC-1999-0754 (MALLBA), EU IST program IST-2001-33116 (FLAGS) and Canary Goverment Project PI/2000-60. C. León partially supported by TRACS program at EPCC. M. Blesa partially supported by Catalan 2001FI-00659 pre-doctoral grant.

problems. Parallelism also plays an important role in developing hybrid algorithms. Clusters of existing commodity workstations are a low-cost hardware alternative to run parallel programs although, in this case, issues as heterogeneity and work load appear.

Application frameworks are a way to reduce the development difficulties by focusing on the reuse of parallel code. Several frameworks offering parallel implementations for generic optimization techniques such as *simulated annealing*, *branch and bound*, or *genetic algorithms* have been proposed (see, e.g. [13, 14, 16, 6]). In these frameworks, the user just describes the elements defining her problem and then instantiates the procedures that parameterize the selected generic method. In the parallel case, these frameworks contribute to reduce the gap between users and parallel architectures, because they hide the implementation details and allow executing parallel programs by just writing sequential code.

Some existing frameworks, such as *Local++* [1], its successor *EasyLocal++* [5], *Abacus* [10] and *Bob++* [3], provide sequential and parallel generic implementations for several exact, heuristic and hybrid methods, but lack features to integrate them.

The MALLBA project is an effort to develop, in an integrated way, a library of skeletons for combinatorial optimization (including exact, heuristic and hybrid methods) that can deal 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. 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. By providing sequential implementations users obtain parallel implementations.
- Cooperation between engines makes possible to provide more powerful hybrid engines.
- Ready to use on commodity machines: Clusters of PCs under Linux are currently supported.
- Flexible and extensible software architecture. New skeletons can easily be added, alternative communication layers can be used, etc.

In MALLBA, each resolution method is encapsulated into a skeleton. At present, the following skeletons for exact techniques are available: Divide and Conquer (DC), Branch and Bound (BnB) and Dynamic Programming (DP). Also the following skeletons for heuristic techniques are available: Hill Climbing, Metropolis, Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithms (GA). Moreover hybrid techniques have been implemented combining the previous skeletons, e.g., GA+TS, GA+SA, BnB+SA.

The paper is organized as follows. Section 2 describes the design of the MALLBA library. Section 3 presents some of the MALLBA skeletons, discusses their implementation for a LAN of workstations, and offers several computational results of the approach. Finally, Section 4 considers some issues related to the WAN implementation currently under development.

2 The MALLBA Architecture

MALLBA skeletons are based on the separation of two concepts: the concrete problem to be solved and the general resolution method to be used. They can be seen as generic templates that just need to be instantiated with the features of a problem in order to solve it. All features related to the selected generic resolution method and its interaction with the concrete problem are implemented by the skeleton. While the particular features related to the problem must be given by the user, the knowledge to parallelize the execution of the resolution method is implemented in the skeleton, so that users do not need to deal with parallelism issues.

The design of the MALLBA library focuses on easy to use skeletons and general and efficient implementations. To achieve both objectives, the C++ programming language was selected due to its high level, modularity, flexibility and efficiency features. We have reduced to a minimum the use of inheritance and virtual methods in order to provide better efficiency and ease of use. To instantiate most problems, a basic knowledge of C++ is enough, and only sequential code without side effects is needed.

Skeletons are implemented by a set of *required* and *provided* C++ classes that represent an abstraction of the entities participating in the resolution method. The *provided* classes implement internal aspects of the skeleton in a problem-independent way. The *required* classes specify information and behavior related to the problem. This conceptual separation allows us to define required classes with a fixed interface but without any implementation, so that provided classes can use required classes in a generic way. Fig. 1 depicts this architecture.

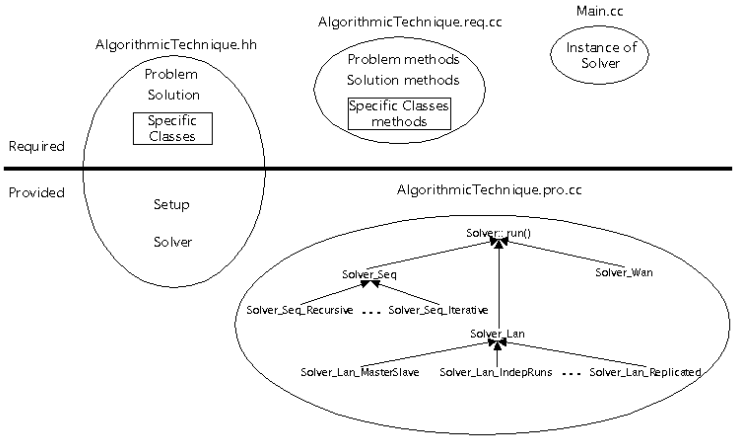

Fig. 1. Architecture of a MALLBA skeleton

More specifically, each skeleton includes the `Problem` and `Solution` required classes, that encapsulate the problem-dependent entities needed by the resolution method. The `Problem` class abstracts the features of the problem that are relevant to the selected optimization method. The `Solution` class abstracts

the features of the feasible solutions that are relevant to the selected resolution method. Depending on the skeleton, other classes may be required. On the other hand, each skeleton offers two provided classes: `Solver` and `Setup`. The former abstracts the selected resolution method. The later contains the setup parameters needed to perform the execution (e.g. number of iterations, number of independent runs, parameters guiding the search, etc.). The `Solver` class provides methods to run the resolution scheme and methods to consult its progress or change its state. The only information the solver needs is an instance of the problem to solve and the setup parameters. In order to enable an skeleton to have different solver engines, the `Solver` class defines a unique interface and provides several subclasses that provide different sequential and parallel implementations (`Solver_Seq`, `Solver_Par`, ...).

Different granularities for the execution of a skeleton are supported: (1) the *phase level*, consisting in obtaining the next solution of the search space from the current solution, (2) the *independent run level*, consisting in executing the whole algorithm once and, (3) the *global level*, consisting in executing several independent runs. This granularity, together with the possibility of consulting the current state of the exploration performed by the solver, allows us to build hybrid skeletons from existing skeletons. Also, this granularity enables us to implement different parallel versions for each skeleton.

3 Parallel Implementations

The skeletons on the MALLBA library are currently implemented for two target environments: sequential and LAN. The user will be able to use different parallelization engines just by easily extending the sequential instantiations. These different implementations can be obtained by creating separate subclasses of the `Solver` abstract class (see Fig. 1). In the following, we detail aspects related to some of the parallel engines implemented for an assorted set of skeletons in the library. Several well known problems have already been instantiated using the MALLBA skeletons as case-studies to check the adaptability of the library. Each of the parallel implementations treated on the following paragraphs reports results for one of these instantiated problems.

3.1 Dynamic Programming. The MALLBA library follows Ibaraki's discrete Dynamic Programming approach for Multistage Problems to represent DP problems [9] and the general parallelization scheme described in [8]. Besides the `Problem` and `Solution` classes common to all the skeletons, the DP Skeleton requires from the user the `Stage`, `State` and `Decision` classes. The user is also required to implement the first stage in `Evalua_First`, the last stage in `Evalua_Last` and the general stage in `Evalua` of the Multistage problem (Fig. 2).

Under this scheme, the DP problem is solved in parallel using a virtually infinite pipeline with as many processors as stages, by allocating the computation of the optimal values on each stage to each one of the virtual processors. Once the user instantiates the required part of the skeleton, then the same instantiation

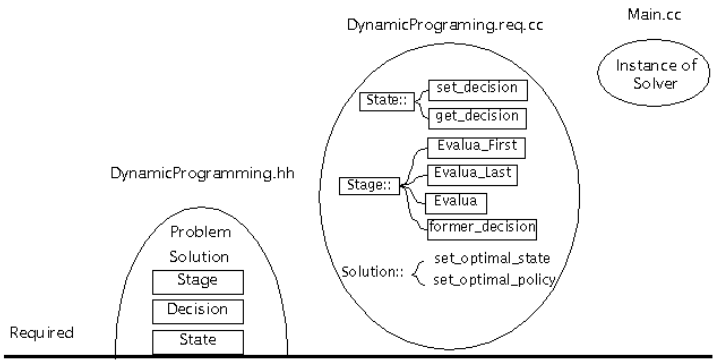

Fig. 2. Required classes for Dynamic Programming skeleton.

runs in sequential and parallel. The `run` method of the `Solver_Seq` class performs a sequential simulation of the stages and the same method on the `Solver_Lan` class enrolls the virtual pipeline into a simulation loop following a cyclic mapping over a ring of processors.

Our current implementation allows the user to introduce as a setup parameter the size of the buffer that will be used in communications in the parallel engine. Table 1 shows the speedup obtained from an instantiation of the Dynamic Programming skeleton for the Resource Allocation Problem [9]. The parallel engine shows a good scalability until four processors. Between four and eight processors the performance decreases due to the slower machines introduced, but it remains increasing when introducing more processors.

Table 1. Results for the Resource Allocation Problem using the Dynamic Programming skeleton, over a network of 13 PCs (4 AMD K6 700 MHz and 9 AMD K6 500 MHz).

Stages-States	Sequential time (s) on fastest machine	Speed-up			
		2 procs. 700 MHz	4 procs. 700 MHz	4 procs. 700 MHz 4 procs. 500 MHz	4 procs. 700 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
1000-10000	11422.60	1.98	3.97	4.18	6.38

3.2 Divide and Conquer and Branch and Bound. The required classes of Divide and Conquer skeleton are `Problem`, `SubProblem`, `Auxiliar` and `Solution`. In particular, the `SubProblem` includes the `easy`, `solve` and `divide` methods. The `Solution` class should define the `combine` method. For the Branch and Bound skeleton, the `Auxiliar` class is not required. `SubProblem` is the only class for which the user must implement some methods: `solved`, `upper_bound`, `lower_bound` and `branch`.

Both skeletons have been parallelized using a farm (master-slave) strategy. While a queue of tasks suffices for the BnB, the DC requires of a hierarchy of queues. For the DC, every time new subproblems are generated (`divide`), a new sub-queue is started, that keeps a child-parent relationship with the queue of the

subproblem being divided. This structure gives support to the required synchronizations, since the corresponding combination phase has to occur after all the children have been solved. To keep a trade between load balancing (i.e. guaranteeing that enough subproblems are generated) and the overhead produced by queue maintenance and work stealing is a difficult issue, since it depends on the nature of the problem and the underlying architecture. Factors to take into account are the relation between the number of available processors and the number of generated subproblems, the depth of the generated subproblems and the communication and computation capabilities of the hosting computer. The user can choose among the several strategies provided by the skeletons.

3.3 Tabu Search. Tabu Search is a meta-heuristic in which the search is guided in order to overcome the local optima [7]. While exploring the solution space, TS tries to avoid cycling by forbidding moves which lead to previously visited solutions. Fundamental ideas to design parallel strategies for meta-heuristics are already well-known (see [2] for a taxonomy of parallel strategies for TS). Here we present several parallel implementations for TS included in the MALLBA library:

- *Independent Runs (IR):* Consists in simultaneous and independent executions of the sequential TS program,
- *Independent Runs with Search Strategies:* A "coordinator" processor generates and sends different search "strategies" to the rest of processors and then each of them executes the TS program according to the given strategy,
- *Master-Slave (MS):* The master processor runs the TS method and uses slaves to choose the "best" move in the neighborhood of the current solution,
- *Master-Slave with Neighborhood Partition.* The neighborhood of the current solution is partitioned and explored in parallel by the slaves. To this aim, a feasible solution is viewed as a collection of items. The master processor partitions the current solution into parts (each part implicitly defines a portion of its neighborhood) and each slave explores the portion of the neighborhood corresponding to the received part.

We give in Table 2 some computational results obtained from the IR with Strategies and the MS with Neighborhood Partition for the 0-1 Multidimensional Knapsack problem. These results are obtained for instances from the standard OR-Library. In the IR with Strategies the communication time is almost irrelevant (at the beginning a "coordinator" processor sends the instance and strategies to the processors and, at the end, receives the best solution found by them) while in the MS with Neighborhood Partition there is a considerable communication time (slave processors report their best solution after each iteration). This explains that, for some instances, the solution found by IR with Strategies is better then the one found by the MS with Neighborhood Partition.

We have also tested our implementations with instances from other known benchmarks and compared our results with those obtained from the *ad hoc* parallel tabu search implementation of Niar et al. [15]; see Table 3. Notice that in both cases the results obtained by our generic implementations are very close to those obtained by other specific implementations for the problem.

Table 2. Results from IR with Strategies and MS with Neighborhood Partition over a network of 9 AMD K6-2 450 MHz. Maximum execution time fixed to 900s. An instance name like OR5x250-00 is an instance of 5 constraints and 250 variables. Averages calculated over 100 executions.

instance	best cost known	avg. deviation % from the best known					
		IR with Startegies			MS with Neighb. Part.		
		2 proc.	4 proc.	8 proc.	2 proc.	4 proc.	8 proc.
OR5x250-00	59312	0.028	0.015	0.020	0.051	0.024	0.020
OR5x250-29	154662	0.005	0.006	0.003	0.012	0.007	0.006
OR10x250-00	59187	0.045	0.047	0.045	0.079	0.064	0.064
OR10x250-29	149704	0.012	0.010	0.004	0.012	0.012	0.009
OR30x250-00	56693	0.023	0.022	0.017	0.041	0.028	0.030
OR30x250-29	149572	0.009	0.010	0.003	0.016	0.009	0.007

Table 3. Results from Niar et al., IR with Strategies and MS with Neighborhood Partition over a network of 9 AMD K6-2 450 MHz. Maximum execution time for our implementations fixed to 900s/#procs.

instance	Number of constraints	Number of variables	best cost known	Niar & Freville Impl.	IR with Strategies		MS with Neighb. Part.	
					4 proc.	8 proc.	4 proc.	8 proc.
Pet7	5	50	16537	16537	16388	16430	16338	16471
Weish10	5	50	6339	6339	6339	6338	6338	6339
Weish30	5	90	11191	11191	11148	11155	11175	11187
Sento1	30	60	7772	7772	7772	7772	7772	7772
Sento2	30	60	8722	8722	8722	8714	8721	8722
Weing7	2	105	1095445	—	1095260	1095260	1095140	1095380
Weing8	2	105	624319	624319	624319	618980	617072	624116
FHP2	4	34	3186	3186	3186	3168	3168	3143
FHP5	10	20	2139	2139	2139	2139	2139	2122

3.1 Parallel Genetic Annealing

We consider here the term *hybridization* as the combination of different optimization skeletons to yield a more controllable or efficient algorithm. In order to have skeletons that can be merged either by skeleton designers or by end users, some kind of *state* information must be offered by any skeleton. The state of the skeleton is both a way of inspecting its actual behavior and a way of controlling its future behavior by a higher-level hybrid strategy. Thus, we have incorporated in the `Solver` class of every skeleton state variables manipulated (read or written) through a *state* in order to be potentially merged.

By controlling the state of skeletons we have developed standard versions of many algorithms, such as parallel genetic algorithms and parallel simulated annealing. GA's are well known meta-heuristics using a set of tentative solutions whose mean fitness is iteratively improved towards better regions of the search space. GA's apply stochastic operators that merge and internally changes these tentative solutions to more promising regions of the problem space. SA is also a well-known heuristic searching in every step the neighborhood of the present solution, and perturbing it to get better solutions following a Boltzmann-like annealing scheme.

The software architecture of MALLBA has allowed us a fast prototyping of the following parallel hybrid skeletons:

- A parallel GA with collaboration among sub-algorithms.
- A parallel SA without collaboration among elementary SA's (SA1).

– A parallel SA with collaboration among elementary SA's (SA2).
– A hybrid skeleton GASA1 where parallel GA's are applying SA as a local search operator inside their sequential main loop.
– Two hybrid skeletons where parallel GA's are run under different population of solutions and then parallel SA's are applied. each one selecting (by tournament -GASA2- or randomly -GASA3-) some strings drawn from the final GA's population to improve them.

In all cases, it must be stressed that the construction of these hybrid algorithms has been extremely easy by using the MALLBA architecture.

We have applied the hybrid skeletons to Maximum Cut and Frequency Assignment. For the former, two instances have been considered: a high edge-density 20-vertex instance (`cut20-0.9`) and a 100-vertex instance (`cut100`). We refer to [12] for the details of these problem instances. As to the Frequency Assignment, two instances from the `CELAR` database have been considered: `CELAR01` (916 radio-links demanding frequencies, for a total number of 5548 interference constraints), and `CELAR02` (200 radio-links, for 1235 interference constraints). Again, we refer to [11] for a description of the problem. A total number of 50 runs has been performed for each algorithm and problem instance.

Table 4. Results for Maximum Cut. The experiments are done using 6 PCs (Pentium III, 700 MHz, 128 Mb under Linux) connected through a Fast Ethernet network. The "iterations" and "time" columns refer to the mean values of successful runs (those in which an optimal solution is found).

	Cut20-0.9			Cut100		
Algorithm	iterations	time(s)	successful runs	iterations	time (s)	successful runs
SA1	1742	0.11	92%	4048	3.96	78%
SA2	856	0.08	100%	2660	2.66	20%
GA	36	0.36	86%	399	78.66	6%
GASA1	5	0.50	100%	54	104.54	36%
GASA2	56	0.59	100%	178	43.84	4%
GASA3	71	0.73	94%	171	33.65	8%

The results for the Maximum Cut are shown in Table 4. In the hybrid parallel models, SA is used at each reproduction event with probability 0.1 for 100 iterations. In the parallel-cooperation modes (all but SA1), the algorithms are arranged in ring topology, asynchronously migrating individuals each 200 iterations. Regarding the 100-vertex graph, the best result is provided by SA1, a parallel SA without cooperation in about 4 s (78% success in finding the optimum). The cooperative parallel version (SA2) is faster yet less effective (just 20% success); the reason for this result strives in that there exists a large number of local optima that SA2 repeatedly visits, while SA1 (without collaboration) focuses the search faster to an optimum in every parallel sub-algorithm, thus avoiding this "oscillation" effect and visiting a larger number of different search regions. The hybrid model GASA1 achieves an intermediate 36%-success value, but it is much more computationally expensive.

The results for Frequency Assigment problem are shown in Table 5. In this case, the parameters of the algorithms are the same as in the previous problem

Table 5. Results for the Frequency Assignment Problem. The "time" column indicates the mean time for finding the best solution.

	CELAR01 (opt.=16)		CELAR02 (opt.=14)	
Algorithm	best solution found	time (s)	best solution found	time (s)
SA1	18	23.00	14	1.39
SA2	20	16.48	14	1.32
GA	18	313.05	14	16.79
GASA1	18	100.74	14	2.94

(with the exception of using 0.01 as the probability of applying SA in the hybrid models). The `CELAR02` instance has been always solved in roughly 2 s. The `CELAR01` instance is a harder instance, demanding more computational effort. SA2 is the faster algorithm, although it does not manage to find the optimum. The most efficient and effective algorithm is SA1, finding a 18-frequency solution in 23 s on average, versus 100 s for GASA1 or 313 s for GA.

Overall, SA1 worked out the best results since it is slightly slower than SA1 but it found the optimum in the two problem instances while SA1 failed in `CELAR01`. Hybrid GASA1 outperformed pure GA improving convergence time and percentage of hits in both set of problem instances.

4 Concluding Remarks and Future Work

We have presented the goals and achievements of MALLBA: we have discussed the architecture of the library, have shown selected skeletons, and have given some computational results obtained over a cluster of heterogeneous PCs using Linux. Our results indicate that: skeletons can be instantiated for a large number of problems, sequential instantiations provided by users are ready to use in parallel, parallel implementations are scalable, general heuristic skeletons can provide solutions whose quality is comparable to *ad hoc* implementations for concrete problems and the architecture supports easy construction of powerful hybrid algorithms.

At present, the library offers two implementations for each skeleton: one for sequential machines and another for a LAN environment. We plan to simplify the parallel executions so that by simply checking the underlying system where the execution will take place, the library will be expected to automatically and adaptively configure the parallel execution.

Our future work will be devoted to provide a third implementation for WAN environments to our skeletons. However, the experience of running applications in WAN environments is still limited. The complexity of this new environment represents a challenge for the developers of parallel applications. The efficiency of Internet applications will be strongly dependent of dynamical network parameters (latency, bandwidth, network overhead) and the implementations tools. To do so, we have started to study the behavior of RedIRIS, the Spanish research network that connects our PC clusters in Barcelona, La Laguna and Málaga.

References

1. M. Cadoli, M. Lenzerini, and A. Schaerf. Local++: A C++ framework for local search algorithms. Technical Report DIS 11-99, University of Rome "La Sapienza", 1999.
2. T. Crainic, M. Toulouse, and M. Gendreau. Towards a taxonomy of parallel tabu search heuristics. *INFORMS Journal on Computing*, 9(1):61–72, 1997.
3. B. L. Cun. Bob++ library illustrated by VRP. In *European Operational Research Conference (EURO'2001)*, page 157, Rotterdam, 2001.
4. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.
5. 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.
6. J. Eckstein, C. A. Phillips, and W. E. Hart. Pico: An object-oriented framework for parallel branch and bound. Technical report, RUTCOR, 2000.
7. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
8. D. González, F. Almeida, J. Roda, and C. Rodríguez. From the theory to the tools: Parallel dynamic programming. *Concurrency: practice and experience*, (12):21–34, 2000.
9. T. Ibaraki. *Enumerative Approaches to Combinatorial Optimization, Part II*. Annals of Operations Research. Volume 11, 1-4, 1988.
10. M. Jünger and S. Thienel. Introduction to ABACUS—a branch-and-cut system. *Operations Research Letters*, 22(2-3):83–95, 1998.
11. A. Kapsalis, V. J. Rayward-Smith, and G. D. Smith. Using genetic algorithms to solve the radio link frequency assigment problem. In D. W. Pearson, N. C. Steele, and R. F. Albretch, editors, *International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 37–40. Springer-Verlag, 1995.
12. 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.
13. K. Klohs. Parallel simulated annealing library. http://www.uni-paderborn.de/fachbereich/AG/monien/SOFTWARE/PARSA/, 1998.
14. D. Levine. PGAPack, parallel genetic algorithm library. http://www.mcs.anl.gov/pgapack.html, 1996.
15. S. Niar and A. Freville. A Parallel Tabu Search Algorithm for the 0-1 Multidimensional Knapsack Problem. In *11th Int. Parallel Proc. Symposium*, 1997.
16. S. Tschöke and T. Polzer. Portable parallel branch-and-bound library, 1997. http://www.uni-paderborn.de/cs/ag-monien/SOFTWARE/PPBB/introduction.html.
17. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.