Engineering Optimization
Vol. 00, No. 00, January 2005, 1–26

Cultural Algorithms, an Alternative Heuristic to Solve the Job Shop Scheduling Problem

Daniel Cortés Rivera, Ricardo Landa Becerra, Carlos A. Coello Coello*
CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Ingeniería Eléctrica
Sección de Computación
Av. Instituto Politécnico Nacional No. 2508
Col. San Pedro Zacatenco
México D.F. 07300, México
Tel. +52 55 5061 3800 x 6564
Fax +52 55 5061 3757
email: {dcortes, rlanda}@computacion.cs.cinvestav.mx
ccoello@cs.cinvestav.mx
(February 2006)

In this work, an approach for solving the job shop scheduling problem using a cultural algorithm is proposed. Cultural algorithms are evolutionary computation methods that extract domain knowledge during the evolutionary process. Additional to this extracted knowledge, the proposed approach also uses domain knowledge given 'a priori' (based on specific domain knowledge available for the job shop scheduling problem). The proposed approach is compared with respect to a Greedy Randomized Adaptive Search Procedure (GRASP), a Parallel GRASP, a Genetic Algorithm, a Hybrid Genetic Algorithm, and a deterministic method called *shifting bottleneck*. The cultural algorithm proposed in this paper is able to produce competitive results with respect to the two approaches previously indicated at a significantly lower computational cost than at least one of them and without using any sort of parallel processing.

Keywords: job shop scheduling; cultural algorithms; metaheuristics; evolutionary algorithms; evolutionary programming

* Corresponding author

1 Introduction

Scheduling problems constitute a very important class within combinatorial optimization because of their complexity and their frequency in real-world applications. The purpose of scheduling (in general) is to allocate a set of (limited) resources to tasks over time (Pinedo 1995). Scheduling has been a very active research area during several years, both in the operations research and in the computer science literature (Baker 1974, Bagchi 1999, Coffman 1976). Research on scheduling basically focuses on finding ways of assigning tasks (or jobs) to machines (i.e. the resources) such that certain criteria are met and certain objective (or objectives) function is optimized.

In the particular case of job shop scheduling, the tasks are jobs and the resources are the machines used to perform such jobs. Each job has a technological sequence and therefore requires to be processed in the machines following a certain order, which is fixed for that problem. The machines cannot process more than one job at a time, and once a machine has started a certain job, it cannot be interrupted before the job is finished. The objectives to be optimized in the case of job shop scheduling can be several, but the most common are either minimizing the maximum makespan or the total makespan.

Several heuristics have been used for different types of scheduling problems (e.g. job shop, flowshop, production, etc.): evolutionary algorithms (Yamada and Nakano 1997, Cheng et al. 1996, 1999), tabu search (Barnes and Chambers 1995, Taillard 1989), simulated annealing (Laarhoven et al. 1992, Catoni 1998), the ant system (Colorni et al. 1994), and artificial immune systems (Hart et al. 1998, Hart and Ross 1999, Cui et al. 2001), among others.

Note however, that this paper presents the first attempt (to the authors' best knowledge) to use cultural algorithms to solve job shop scheduling problems. Cultural algorithms (Reynolds 1994) are a particular class of evolutionary algorithm that use domain knowledge extracted during the evolutionary process in order to improve the performance of the search engine (i.e. the evolutionary algorithm) adopted. What we explore in this paper is the use of a combination of knowledge extracted during the evolutionary search with some knowledge that is inserted *a priori* because it is known to be useful in the job scheduling problem. The main hypothesis in this regard was that the incorporation of knowledge into an evolutionary algorithm would increase its performance as to make it competitive with other approaches whose computational cost is significantly higher.

The proposed approach is compared with respect to GRASP (Greedy Randomized Adaptive Search Procedure), a parallel version of GRASP, a genetic algorithm and a hybrid genetic algorithm with local search, in several test problems taken from the specialized literature. The obtained results indicate that the proposed approach is a viable alternative for solving efficiently job

shop scheduling problems.

The remainder of this paper is organized as follows: in Section 2 a brief description of the statement of the problem is provided. Section 3 contains an introduction to cultural algorithms which includes a description of their main components and the main motivation to use them. Section 4 contains the details of the proposed approach to solve job shop scheduling problems using a cultural algorithm. As part of this section, a description of the representation of solutions adopted in this work is included, as well as the mechanisms implemented to add domain knowledge to the evolutionary algorithm both before and during the search process. Section 5 provides a comparative study. Finally, Section 6 presents the general conclusions and some possible paths for future research.

2 Problem Statement

The job shop scheduling problem (JSSP) can be defined in the following way: let's consider a set of jobs, $j_1, j_2, \ldots, j_n$ that need to be processed in a set of machines, $\mu_1, \mu_2, \ldots, \mu_m$. The processing of job j_q in the machine μ_r is an operation that requires of a time $\tau_{q,r}$. Each job has a technological sequence (i.e. an order for the machines in which the job should be processed). Other important constraints are that the processing of a job requires the exclusive use of the machine in which it is located at that time. Additionally, the processing of a job cannot be interrupted in a machine once started.

Since the number of jobs will be represented by n and the number of machines will be represented by m, the problem is denoted as $n \times m$ JSSP. A schedule is then a set of duration times for each operation $\{o_{q,r}\}_{1 \leq q \leq n, 1 \leq r \leq m}$ that satisfies the previously indicated conditions. The total duration time required to complete all the jobs (makespan) will be called L. The goal is then to minimize L. For the purposes of the work reported in this paper, the objective considered will be the minimization of the makespan (i.e. the time taken to finish the last job available). In other words, the goal is to find a schedule that has the minimum duration required to complete all the jobs (Baker 1974).

Garey and Johnson (1979) showed that the JSSP is an **NP-hard** problem and within its class it is one of the least tractable problems (Bagchi 1999). Several enumerative algorithms based on *Branch & Bound* have been applied to JSSP. However, due to the high computational cost of these enumerative algorithms, some approximation approaches have also been developed. The most popular practical algorithm to date is the one based on *priority rules* and *active schedule generation* (Jones and Rabelo 1998). However, other algorithms, such as an approach called *shifting bottleneck* have been found to be very effective in practice (Adams et al. 1988). Furthermore, a number of

heuristics have also been used in the JSSP, (e.g. genetic algorithms (Bagchi 1999, Yamada and Nakano 1997), tabu search (Barnes and Chambers 1995), simulated annealing (Catoni 1998), and artificial immune systems (Hart and Ross 1999), among others), as indicated before.

An instance of the JSSP can be formulated in tabular form as indicated in Table 1, where a 3×3 problem is shown. The jobs are listed in the first column. Each table entry indicates the machine in which a job must be processed (based on its corresponding technological sequence) followed by a number in parentheses that represents the time $\tau_{q,r}$ that takes to the job to be processed in that machine.

3 Cultural Algorithms

Cultural algorithms were developed by Reynolds (1994) as a complement to the metaphor used by evolutionary algorithms (Fogel 1995), which had focused mainly on genetic and natural selection concepts.

Cultural algorithms are based on some theories originated in sociology and archaeology which try to model cultural evolution (see for example (Renfrew 1994, Durham 1994)). Such theories indicate that cultural evolution can be seen as an inheritance process operating at two levels: (1) a micro-evolutionary level, which consists of the genetic material that an offspring inherits from its parents, and (2) a macro-evolutionary level, which consists of the knowledge acquired by individuals through generations. This knowledge, once encoded and stored, is used to guide the behavior of the individuals that belong to a certain population.

Culture can be seen as a set of ideological phenomena shared by a population. Through these phenomena, an individual can interpret its experiences and decide its behavior. In these models, it can be clearly appreciated the part of the system that is shared by the population: the knowledge, acquired by members of a society, but encoded in such a way that such knowledge can be accessed by every other member of the society. And then there is an individual part, which consists of the interpretation of such knowledge encoded in the form of symbols. This interpretation will produce new behaviors as a consequence of the assimilation of the corresponding knowledge acquired combined with the experiences lived by the individual itself.

Reynolds (1994) attempts to capture this double inheritance phenomenon through his proposal of cultural algorithms. The main goal of such algorithms is to increase the learning or convergence rates of an evolutionary algorithm such that the system can respond better to a wide variety of problems (Franklin and Bergerman 2000).

Cultural algorithms operate in two spaces. First, there is the population

space, which consists of (as in all evolutionary algorithms) a set of individuals. Each individual has a set of independent features that are used to determine its fitness. Through time, such individuals can be replaced by some of their descendants, which are obtained through the application of a set of operators from the population.

The second space is the belief space, which is where the knowledge, acquired by individuals through generations, is stored. The information contained in this space must be accessible to each individual, so that they can use it to modify their behavior. In order to join the two spaces, it is necessary to provide a communication link, which dictates the rules regarding the type of information that must be exchanged between the two spaces.

The following is the pseudo-code of a cultural algorithm.

```
Generate the initial population
Initialize the belief space
Evaluate the initial population
repeat
   Update the belief space (with the individuals accepted)
   Apply the variation operators (under the influence of the belief space)
   Evaluate each child
   Perform selection
until the end condition is satisfied
```

Most of the steps of a cultural algorithm correspond with the steps of a traditional evolutionary algorithm. It can be clearly seen that the main difference lies in the fact that cultural algorithms use a belief space. In the main loop of the algorithm, the belief space must be updated. It is at this point in which the belief space incorporates the individual experiences of a select group of members of the population. Such a group is obtained with the function *accept*, which is applied to the entire population.

On the other hand, the variation operators (such as recombination or mutation) are modified by the function *influence*. This function applies some pressure such that the children resulting from the variation operators can exhibit behaviors closer to the desirable ones and farther away from the undesirable ones, according to the information stored in the belief space.

These two functions (*accept* and *influence*) constitute the communication link between the population space and the belief space. Such interactions can be appreciated in Figure 1 (Reynolds 1999). The implementation details for these functions in the current proposal are given in the next section.

In (Reynolds 1994), it is proposed the use of genetic algorithms (Goldberg 1989) to model the micro-evolutionary process, and Version Spaces (Mitchell

1978) to model the macro-evolutionary process of a cultural algorithm. This sort of algorithm was called the *Version Space guided Genetic Algorithm* (VGA). The main idea behind this approach is to preserve beliefs that are socially accepted and discard (or prune) unacceptable beliefs. Therefore, if a cultural algorithm for global optimization is applied, the acceptable beliefs can be seen as constraints that direct the population at the micro-evolutionary level (Michalewicz 1995).

In genetic algorithms' theory, there is an expression, called *schema theorem* (Holland 1975) that represents a bound on the speed at which the best schemata of the population are propagated. Reynolds (1994) provided a brief discussion regarding how could the belief space affect the schema theorem. His conclusion is that by adding a belief space to an evolutionary algorithm, the performance of such algorithm can be improved by increasing its convergence rate. That constitutes the main motivation to use cultural algorithms. Despite the lack of a formal mathematical proof of this efficiency improvement, there is empirical evidence of such performance gains reported in the literature (see for example (Chung and Reynolds 1998, Coello Coello and Landa Becerra 2002)).

4 Proposed Approach

The approach proposed in this paper uses, as its population space, the population adopted by evolutionary programming (Fogel 1999), together with its selection and variation operators. The pseudo-code of this evolutionary programming-based algorithm is shown next. The details about the culture influenced mutation and the update of the belief space are discussed later in this section.

Generate s random schedules (initial population)
Compute the makespan of each individual in the initial population
Initialize the belief space (copying the best individual to the belief space)
repeat
  for each individual in current population **do**
    Apply (culture influenced) mutation
    Compute the makespan of the new individual
  end for
  for each individual in the populations of parents and offspring **do**
    Randomly select c opponents from the union of parents and children
    Compare the individual with the c opponents, and store its number of victories
  end for

Select the s individuals with the largest number of victories to conform the new population
Update the belief space (with the individuals accepted)
until the end condition is satisfied

In evolutionary programming, there are s individuals in the original population (such individuals are randomly generated). In the main loop of the evolutionary programming algorithm only mutation is applied besides selection. This is because this approach simulates evolution at the species level, and different species do not recombine among themselves (Fogel 1995).

The mutation operator obtains a child from each of the individuals in the population (i.e. s children are obtained). In the case of continuous optimization, the mutation operator consists of adding Gaussian noise to each variable (Coello Coello and Landa Becerra 2002). In this case, since a combinatorial optimization problem is addressed, a set of exchanges in the sequence of the operations is used as the mutation operator.

Selection in evolutionary programming consists of a set of tournaments. For each individual in the population (including both parents and offspring), a random sample of size c is chosen, and each individual is compared with respect to each member of this sample through c binary tournaments. The number of wins accumulated by each individual is stored. At the end of all the tournaments, the s individuals with the largest number of victories are selected to constitute the population at the next generation.

4.1 *Representation*

The representation adopted to encode the solutions plays a very important role when applying an evolutionary computation technique (Rothlauf 2002, Ronald 1997), and this issue plays a crucial role when specifically dealing with the JSSP (Yamada and Nakano 1997). This is due to the fact that a solution to the JSSP cannot be represented as a permutation as normally done in many combinatorial optimization problems (Ronald 1995, Michalewicz 1996).

The use of a permutation-based representation would only be possible in job shop scheduling if the problem to be solved only had one machine. In such case, the n jobs would require to be processed in the only existing machine and the different solutions would consist of the ordering in which the jobs would be processed (this is precisely the ordering that could be represented using a permutation). However, regardless of the processing order of the jobs, the time taken to complete the last job (i.e. the makespan) is always the same, as long as there are no pauses in the schedule.

For the general JSSP of size $n \times m$, several types of possible encodings have

been proposed in the literature. Some examples are the use of binary encoding (Nakano and Yamada 1991), the use of disjunctive graphs (Colorni et al. 1994), and the permutations with repetitions (Bierwirth 1995).

Most of the existing encodings can generate invalid schedules and thus require a repair mechanism (Michalewicz and Schoenauer 1996). These repair mechanisms tend to bias solutions towards a certain region of the search space and are, therefore, not always advisable (Coello Coello 2002).

The permutation with repetitions has the advantage of never generating invalid schedules and that was precisely the main reason for which it was decided to adopt it for this approach. This representation consists of a permutation in which each component is repeated m times. The components of the permutation represent jobs as in the previous example of a single machine. However, in this case, the k-th occurrence of a job indicates the k-th operation in the technological sequence of such job. In Figure 2, an example of the decoding of a permutation with repetitions for the problem described in Table 1 is shown.

The main disadvantage of this representation is that different permutations can encode the same schedule. In the experiments performed, we found that this redundancy in the encoding is not a serious drawback when the search space is properly explored. However, this remains as an issue that must be considered when adopting this representation.

4.2 *Domain Knowledge Added* a priori *to the Algorithm*

In order to incorporate domain knowledge into the proposed algorithm, two mechanisms were used. The first of them involves adding knowledge *a priori* (i.e., before actually running the algorithm), whereas the second one involves extracting information during the execution of the algorithm following the traditional model of a cultural algorithm.

Next, we describe the addition of *a priori* domain knowledge. This mechanism is integrated during the evaluation of a new individual. First, a *semi-active* schedule is defined as that in which the operations are performed as soon as possible, but without changing the ordering of the schedule. For example, in Figure 2, the operation of job 1 in machine 2 can start at the same time as the operation of the job 2 in machine 1, since machine 2 is not busy at the moment and the technological sequence of job 1 is accomplished.

However, it may be the case that a semi-active schedule has long pauses in the use of some of the machines. In some cases, it is possible that some of the jobs can be traversed to fill up that pause, thus reducing the makespan of the schedule. This traversal movements are generically called *permissible left shifts*. A schedule to which it is not possible to apply more permissible left shifts is called *active*. The search of active schedules through permissible left shifts considerably reduces the search space and it is, therefore, advisable.

In the algorithm proposed in this paper, permissible left shifts are applied during the evaluation of an individual. The individual to be evaluated is applied all the permissible left shifts possible. When all the permissible left shifts are applied to the decoded schedule, the permutation with repetitions string is reordered, in order to obtain a new string where no left shifts need to be applied.

In Figure 3, we show an example of the aplication of permissible left shitfs to obtain an active schedule for the problem depicted in Table 1.

This process is called insertion of *a priori* domain knowledge, because these modifications are *ad hoc* to the JSSP, and they are encoded in the algorithm prior to its execution,

4.3 *Domain Knowledge Extracted During the Search Process*

This second knowledge insertion mechanism is integrated to the mutation operator. The identification of the best individual found so far, and the best individuals in a current generation, in addition to the use of a mechanism to enforce that new individuals generated are similar to the best ones, are used in this case to accelerate convergence.

This is precisely the main idea on which the design of the belief space of this approach is based. The belief space contains a part called situational knowledge, which has previously been used for continuous optimization (Chung and Reynolds 1998, Jin and Reynolds 1999). Situational knowledge consists of storing the best individuals found during the evolutionary process, and use them as leaders that other solutions must follow.

In this case, the best individual found so far is stored, and the individual in the current generation which has the larger number of tournament victories (in case of a tie, all the tied individuals are stored). The idea is that the best individuals found during the search process provide us information about the patterns that the sequences of operations in the machines should follow in order to decrease the makespan value of the solutions generated.

The mutation operator is modified in order to cause that the other solutions have a greater 'resemblance' (i.e. that their values are more similar) with respect to the individuals stored in the situational part. The mutation operator proposed is based on swaps of components in the permutation with repetitions.

The *influence* function over the variation operator is only applied the p_{cult} of the time, while the rest of the time, only a variation operator is applied. The influence consists of choosing a reference individual at random, from any of the individuals in the situational part, and replacing the original individual to be mutated.

In the individual to be mutated, the algorithm selects an operation randomly, then it identifies another operation being processed in the same ma-

chine, and exchanges those two operations.

This mutation operator is applied the $1/(m \times n)$ of the time. Thus, the operator is expected to be applied only once in a given string (of size $m \times n$).

An example of this process is shown in Figure 4. This problem is based in the problem described in Table 2. In the line labeled with (a), we show the input string, and the operations to be exchanged are also indicated, both in machine 4. In line (b), the string with the exchange of the selected operations is shown. In line (c), we show the string that maps to the same solution, but reordered to avoid the application of permissible left shifts in its descendants.

The update of the belief space consists only of replacing the individuals stored in the situational knowledge with the accepted individuals. The *accept* function selects the best individual found in the current population, and the individual with the largest number of victories.

5 Comparison of Results

The Cultural Algorithm (CULT) is compared with respect to 4 different approaches: a Greedy Randomized Adaptive Search Procedure (GRASP) approach reported in (Binato et al. 2002), a parallel version of GRASP reported in (Aiex et al. 2003), a genetic algorithm (GA) reported in (Gonçalves and Beirao 1999), and a hybrid genetic algorithm with local search (HGA) reported in (Gonçalves et al. 2002). These references were chosen for three main reasons: (1) they provide enough information (e.g. numerical results) as to allow a comparison, (2) these algorithms have been found to be very powerful in the JSSP studied in this paper, and (3) the comparison with respect to another evolutionary algorithm that does not use domain knowledge was an important issue for us. In addition to these algorithms, as a reference point, a comparison against a deterministic method, the shifting bottleneck (SB) (Adams et al. 1988), is also provided.

The benchmark adopted for the experiments is a subset of the JSSP instances contained in the OR-library (Beasley 1990). The OR-library is a set of test problems for different types of problems of interest in operations research. Over the years, the OR-library has become a standard benchmark to validate new approaches to solve such problems. The OR-library contains problems of different degrees of difficulty and reports the best known solution in each instance contained within. In this particular case, 3 problems of (Fischer and Thompson 1963), labeled FT06, FT10 and FT20, and the 40 problems of (Lawrence 1984), labeled from LA01 to LA40 were adopted.

The proposed approach was implemented in the C++ programming language and was compiled using the GNU g++ compiler.

Table 3 shows the overall comparison of results. In the first column are the

algorithms with respect to which the comparison is made, together with their corresponding reference. In the second column is shown the average deviation of the best results obtained by each algorithm with respect to the best known solution for the 43 test problems adopted in this study. The last column indicates the improvement achieved by the cultural algorithm with respect to each of the other algorithms compared. From Table 3, it can be seen that the proposed approach was able to improve on the results produced by the five algorithms compared.

In Table 4, the overall performance of the cultural algorithm is shown, with respect to the 5 other algorithms against which it was compared. The column labeled **Win** shows the number of problems in which each algorithm beat the cultural algorithm. The column labeled **Tie** indicates ties between the cultural algorithm and the other algorithms. Finally, the column labeled **Lose** indicates the number of problems in which each algorithm lost with respect to the cultural algorithm. Results indicate that SB and GRASP never beat the cultural algorithm. Parallel GRASP beat the cultural algorithm only in 4 problems and lost in 7. Note however, that the proposed approach is implemented sequentially and not in parallel as the Parallel GRASP. Regarding the GA, it beat the cultural algorithm in 1 problem and lost in 20. Finally, HGA beat the cultural algorithm in 4 problems and lost in 8.

Table 5 compares the best results found by CULT, SB, GRASP, the Parallel GRASP, GA and HGA. **Boldface** is used to indicate both the best known solutions (BKS) and when an algorithm reached such result.

Table 6 compares the number of evaluations required by CULT, and the iterations required by GRASP and Parallel GRASP.

In all the examples, 10 independent runs of the proposed algorithm were performed. The criterion adopted to stop the algorithm was to detect when no changes in the result were reported after a certain (normally large) number of consecutive iterations.

The parameters of the cultural algorithm that remained without changes are the following:

$$
\begin{aligned}
s &= 20 \\
c &= \frac{p}{2} = 10 \\
p_{cult} &= 0.1 \\
p_m &= \frac{1}{m \times n}
\end{aligned}
$$

where s is the population size, c is the number of binary confrontations to be performed by each individual during the tournament selection, p_{cult} is the

probability to apply the situational knowledge, and p_m is the mutation ratio.

Some of the most remarkable examples are the following:

- **LA16**: In this problem, the cultural algorithm, Parallel GRASP and HGA can reach the best known solution. However, GRASP requires 50.1 million iterations, Parallel GRASP requires 1.3 million iterations, and our proposed approach only requires 10 000 evaluations.

  Regarding CPU times, GRASP required 155 310 s on a SGI R10000 with a 196 MHz processor, Parallel GRASP (a sequential version) required 2 951 s on a SGI R10000 running at 196 MHz, HGA required 232 s on an AMD Thunderbird with a 1.333 GHz processor, and CULT required 2.23 s on an AMD K6-2 with a 550 MHz processor.

  The systems on which the algorithms ran remained the same for the rest of the instances discussed here; note that the processor on which CULT ran is almost three times faster than those of GRASP and Parallel GRASP, but has less than half of the speed of the one in which the HGA ran.
- **LA24**: In this problem, none of the 5 approaches converged to the best known solution, but the cultural algorithm produced the closest approximation. The cultural algorithm required 250 000 evaluations, whereas Parallel GRASP required 125 million iterations, and GRASP required 10.1 million iterations.

  The times required by each algorithm were: 64 640 s for GRASP, 407 500 s for Parallel GRASP, 578 s for HGA and 95.5 s for CULT.
- **LA30**: Parallel GRASP, HGA and the cultural algorithm found the best known solution, while the two other apporaches cannot. However, Parallel GRASP required 3 million iterations, and the proposed approach required only 50 000 evaluations.

  106 050 s were necessary for the results reported by GRASP, 22 830 s for the results of Parallel GRASP, 1 260 s for the results of HGA and 30.9 s for the results of CULT.
- **LA22**: The cultural algorithm converges to a solution that is only 3 units away from the best known solution. Parallel GRASP, in contrast, converges to the best known solution. However, Parallel GRASP required 26 million evaluations, whereas the cultural algorithm only performed 200 000 evaluations.

  The CPU times required by the different algorithms were: 315 630 s for GRASP, 89 700 s for Parallel GRASP, 629 s for HGA and 75.3 s for CULT.

For these 4 instances, convergence graphs of typical executions can be found in Figure 5.

Looking for a compromise to setup *a priori* a maximum number of objective function (or fitness) evaluations, a population size of 10 individuals, and a maximum number of generations of 20 000 are suggested. This will produce a

total of 200 000 fitness function evaluations, which is a good compromise for solving both the 'easy' and the 'difficult' problems included in the benchmark adopted.

The results obtained in this case are shown in Table 7. Note that in this case the median and worst results found by the proposed approach in each instance are also reported. As expected, results are poorer in this case, because some problems require a significantly larger number of evaluations. However, this alternative provides an alternative to setup *a priori* a maximum number of fitness function evaluations in the cultural algorithm. In any case, more work in this direction is desirable as to improve the search capabilities of the proposed approach while maintaining a relatively low number of fitness function evaluations.

6 Conclusions and Future Work

A new approach based on a cultural algorithm to solve JSSPs is introduced. The approach uses both knowledge introduced *a priori* (i.e. a heuristic to perform local rearrangements which we know beforehand that can reduce the makespan) and extracted during the evolutionary search. The proposed approach adopts a permutation representation that allows repetitions. The comparison of results indicated that the proposed approach is competitive with respect to other heuristics, even improving on their results in some cases.

In terms of computational efficiency, the proposed approach performs a number of evaluations that is (on average) considerably lower than those performed by Parallel GRASP, while producing similar results. Results are also competitive with respect to HGA, despite the fact that the results were obtained by the cultural algorithm without any local search.

As part of our future work, we plan to improve the heuristics adopted to perform local moves. It is also intended to introduce a backtracking mechanism to recover from movements towards local attractors, and it is also planned to incorporate into the algorithm certain mechanisms from tabu search (Glover and Laguna 1998).

It is also desirable to find a set of parameters that can be fixed for a larger family of problems as to eliminate the variability of iterations that is currently reported for the proposed algorithm.

Finally, it is also planned to work on a multiobjective version of the JSSP in which 3 objectives would be considered (Bagchi 1999): 1) makespan, 2) mean flowtime and 3) mean tardiness. This would allow us to generate trade-offs that the user could evaluate in order to decide what solution to choose.

Acknowledgments

The authors thank the anonymous reviewers for their comments and suggestions which greatly improved the contents of this paper.

The authors acknowledge partial support from CONACyT Mexico-CONICYT Chile through project No. J110.331/2005.

The second author acknowledges support from CONACyT through a scholarship to pursue graduate studies in Computer Science at the Sección de Computación of the Electrical Engineering Department at CINVESTAV-IPN.

The third author acknowledges support from CONACyT through project No. 42435-Y.

REFERENCES

Adams, J., Balas, E. and Zawack, D., The shifting bottleneck procedure for job shop scheduling. *Management Science*, 1988, **34**(3), 391–401.

Aiex, R. M., Binato, S. and Resende, M. G., Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 2003, **29**(4), 393–430.

Bagchi, T. P., *Multiobjective Scheduling by Genetic Algorithms*, 1999 (New York: Kluwer Academic Publishers).

Baker, K. R., *Introduction to Sequencing and Scheduling*, 1974 (New York: John Wiley & Sons).

Barnes, J. and Chambers, J., Solving the Job Shop Scheduling Problem using Tabu Search. *IIE Transactions*, 1995, **27**(2), 257–263.

Beasley, J. E., OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 1990, **41**(11), 1069–1072.

Bierwirth, C., A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms. *OR Spektrum*, 1995, **17**, 87–92.

Binato, S., Hery, W. J., Loewenstern, D. M. and Resende, M. G. C., A GRASP for Job Shop Scheduling. In *Essays and Surveys in Metaheuristics*, C. Ribeiro and P. Hansen (Eds), Operations Research/Computer Science Interfaces, 2002 (Kluwer Academic Publishers).

Catoni, O., Solving Scheduling Problems by Simulated Annealing. *SIAM Journal on Control and Optimization*, 1998, **36**(5), 1539–1575.

Cheng, R., Gen, M. and Tsujimura, Y., A tutorial survey of job-shop scheduling problems using genetic algorithms: I. Representation. *Computers and Industrial Engineering*, 1996, **30**(4), 983–997.

Cheng, R., Gen, M. and Tsujimura, Y., A tutorial survey of job-shop scheduling problems using genetic algorithms: II. Hybrid genetic search strategies. *Computers and Industrial Engineering*, 1999, **36**(2), 343–364.

Chung, C.-J. and Reynolds, R. G., CAEP: An Evolution-based Tool for Real-

Valued Function Optimization using Cultural Algorithms. *Journal on Artificial Intelligence Tools*, 1998, **7**(3), 239–292.

Coello Coello, C. A., Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 2002, **191**(11–12), 1245–1287.

Coello Coello, C. A. and Landa Becerra, R., Adding Knowledge and Efficient Data Structures to Evolutionary Programming: A Cultural Algorithm for Constrained Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, W. Langdon et al. (Eds), pp. 201–209, 2002 (San Francisco, California: Morgan Kaufmann Publishers).

Coffman, Jr., E., *Computer and Job Shop Scheduling Theory*, 1976 (John Wiley and Sons).

Colorni, A., Dorigo, M., Maniezzo, V. and Trubian, M., Ant system for Job-shop scheduling. *JORBEL–Belgian Journal of Operations Research, Statistics and Computer Science*, 1994, **34**, 39–53.

Cui, X., Li, M. and Fang, T., Study of Population Diversity of Multiobjective Evolutionary Algorithm Based on Immune and Entropy Principles. In: *Proceedings of the Congress on Evolutionary Computation 2001 (CEC'2001)* pp. 1316–1321, Vol. 2, 2001 (Piscataway, New Jersey: IEEE Service Center).

Durham, W. H., *Co-evolution: Genes, Culture, and Human Diversity*, 1994 (Stanford, California: Stanford University Press).

Fischer, H. and Thompson, G. L., Probabilistic learning combinations of local job-shop scheduling rules. In: *Industrial Scheduling*, J. F. Muth and G. L. Thompson (Eds), pp. 225–251, 1963 (Prentice-Hall).

Fogel, D. B., *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*, 1995 (New York: The Institute of Electrical and Electronic Engineers).

Fogel, L. J., *Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming*, 1999 (New York: John Wiley & Sons, Inc.).

Franklin, B. and Bergerman, M., Cultural algorithms: Concepts and experiments. In: *Proceedings of the 2000 Congress on Evolutionary Computation*, pp. 1245–1251, 2000. (Piscataway, New Jersey: IEEE Service Center).

Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*, 1979 (W H Freeman & Co).

Glover, F. and Laguna, M., *Tabu Search*, 1998 (Norwell Massachusetts: Kluwer Academic Publishers).

Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, 1989 (Reading, Massachusetts: Addison-Wesley Publishing Company).

Gonçalves, J. F. and Beirao, N. C., Um algoritmo genético baseado em chaves

aleatórias para sequenciamiento de operaçoes. *Revista Revista Associação Portuguesa de Desenvolvimento e Investigação Operacional*, 1999, **19**, 123–137. (in Portuguese).

Gonçalves, J. F., de Magalhães Mendes, J. J. and Resende, M. G. C., A hybrid genetic algorithm for the job shop scheduling problem, Technical Report TD-5EAL6J, AT&T Labs, 2002.

Hart, E. and Ross, P., The Evolution and Analysis of a Potential Antibody Library for Use in Job-Shop Scheduling. In: *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover (Eds), pp. 185–202, 1999, (London: McGraw-Hill).

Hart, E., Ross, P. and Nelson, J., Producing robust schedules via an artificial immune system. In: *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*, pp. 464–469, 1998, (Anchorage, Alaska: IEEE Press).

Holland, J. H., *Adaptation in Natural and Artificial Systems*, 1975 (Ann Arbor, Michigan: University of Michigan Press).

Jin, X. and Reynolds, R. G., Using Knowledge-Based Evolutionary Computation to Solve Nonlinear Constraint Optimization Problems: a Cultural Algorithm Approach. In: *1999 Congress on Evolutionary Computation*, pp. 1672–1678, 1999. (Washington, D.C.: IEEE Service Center).

Jones, A. and Rabelo, L. C., Survey of Job Shop Scheduling Techniques, NISTIR, National Institute of Standards and Technology, 1998.

Laarhoven, P. J. M. v., Aarts, E. H. L. and Lenstra, J., Job shop scheduling by simulated annealing. *Operations Research*, 1992, **40**, 113–125.

Lawrence, S. R., Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984. (Unpublished).

Michalewicz, Z., A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. In: *Proceedings of the 4th Annual Conference on Evolutionary Programming*, J. R. McDonnell, R. G. Reynolds and D. B. Fogel (Eds), pp. 135–155, 1995 (Cambridge, Massachusetts: The MIT Press).

Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, 1996 (New York: Springer-Verlag).

Michalewicz, Z. and Schoenauer, M., Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 1996, **4**(1), 1–32.

Mitchell, T., Version Spaces: An Approach to Concept Learning, PhD thesis, Computer Science Department, Stanford University, Stanford, California, 1978.

Nakano, R. and Yamada, T., Conventional Genetic Algorithm for Job Shop Problems. In: *Proceedings of the Fourth International Conference on Genetic*

Algorithms (ICGA-91), R. K. Belew and L. B. Booker (Eds), pp. 474–479, University of California, San Diego, 1991 (San Mateo, California: Morgan Kaufmann Publishers).

Pinedo, M., *Scheduling: Theory, Algorithms and Systems*, 1995 (Englewood Cliffs, New Jersey: Prentice Hall).

Renfrew, A. C., Dynamic Modeling in Archaeology: What, When, and Where?. In: *Dynamical Modeling and the Study of Change in Archaelogy*, S. E. van der Leeuw (Ed), 1994. (Edinburgh, Scotland: Edinburgh University Press).

Reynolds, R. G., An Introduction to Cultural Algorithms. In: *Proceedings of the Third Annual Conference on Evolutionary Programming*, A. V. Sebald and L. J. Fogel (Eds), pp. 131–139 (River Edge, New Jersey: World Scientific).

Reynolds, R. G., Cultural algorithms: Theory and applications. In: *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover (Eds), pp. 367–377, 1999 (London, UK: McGraw-Hill). .

Ronald, S., Genetic algorithms and permutation-encoded problems: Diversity preservation and a study of multimodality, PhD thesis, The University of South Australia, 1995.

Ronald, S., Robust encodings in genetic algorithms. In: *Evolutionaty Algorithms in Engineering Applications*, D. Dasgupta and Z. Michalewicz (Eds), pp. 30–44, 1997 (Springer-Verlag).

Rothlauf, F., *Representations for Genetic and Evolutionary Algorithms*, 2002 (New York: Physica-Verlag).

Taillard, E., Parallel tabu search technique for the jobshop scheduling problem, Technical Report ORWP 89111, Ecole Polytechnique Federale, Lausanne, Switzerland, 1989.

Yamada, T. and Nakano, R., Job-shop scheduling. In: *Genetic Algorithms in Engineering Systems*, A. M. S. Zalzala and P. J. Fleming (Eds), pp. 134–160, 1997, (The Institution of Electrical Engineers).

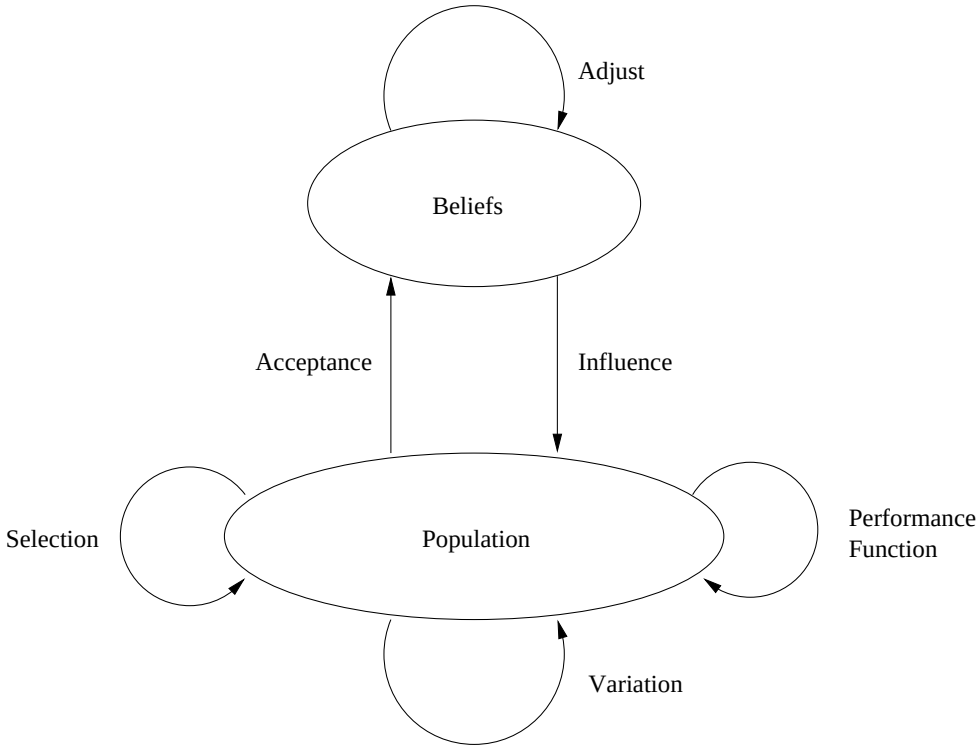


Figure 1. Spaces of a cultural algorithm

permutation with repetition 1 2 1 2 2 1 3 3 3

decoded schedule

m_1	1	2					3	
m_2			1		2		3	
m_3				2		1		3

Figure 2. Example of the decoding process of a permutation with repetitions.

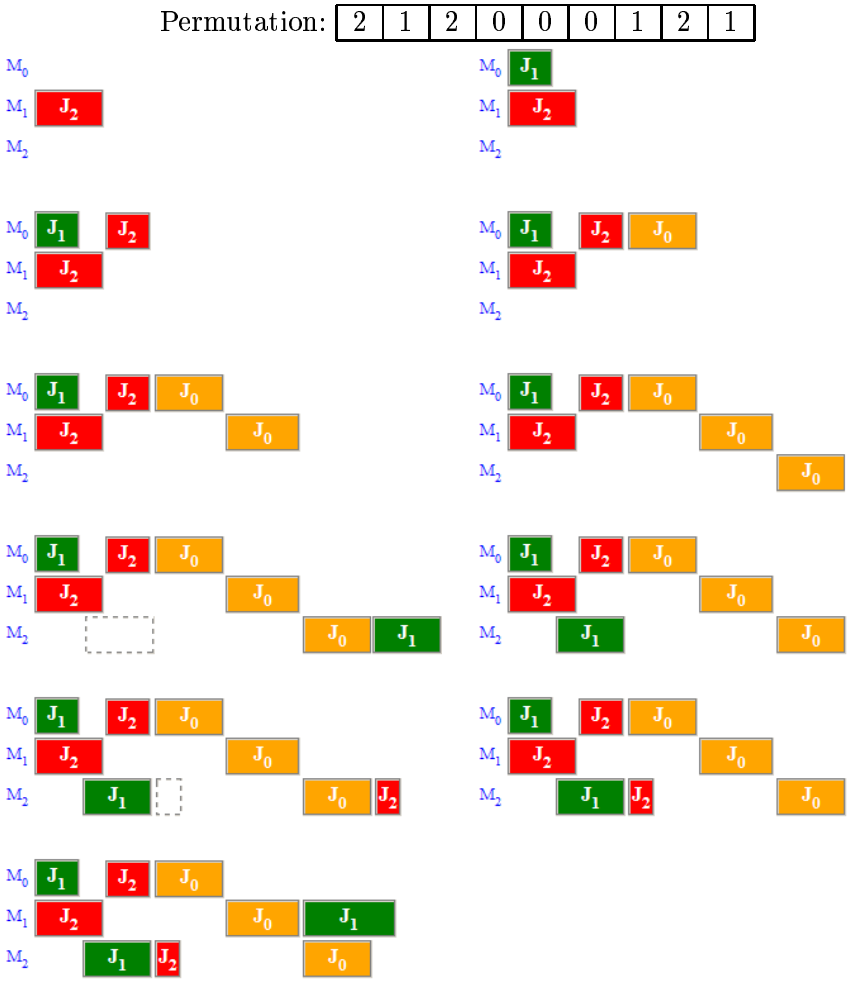


Figure 3. Left shifts applied to obtain an *active* schedule.

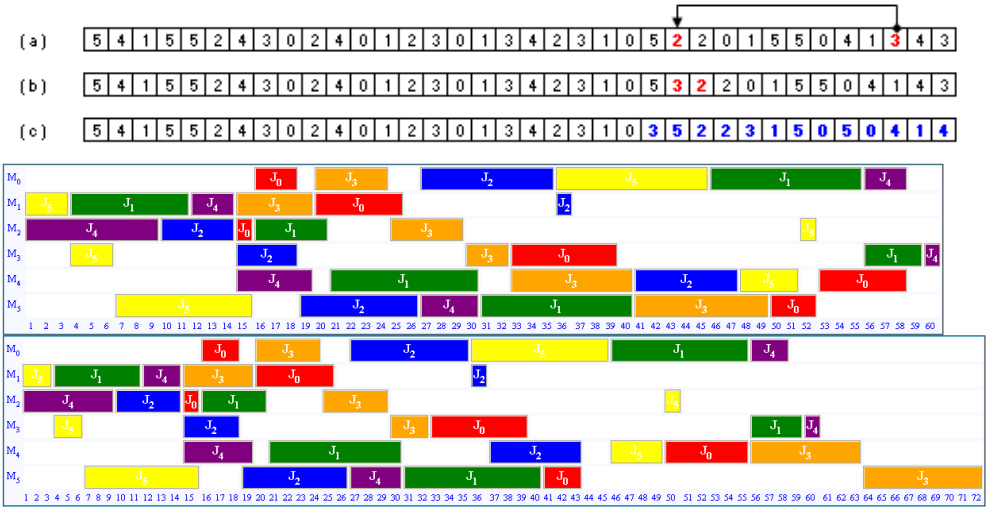
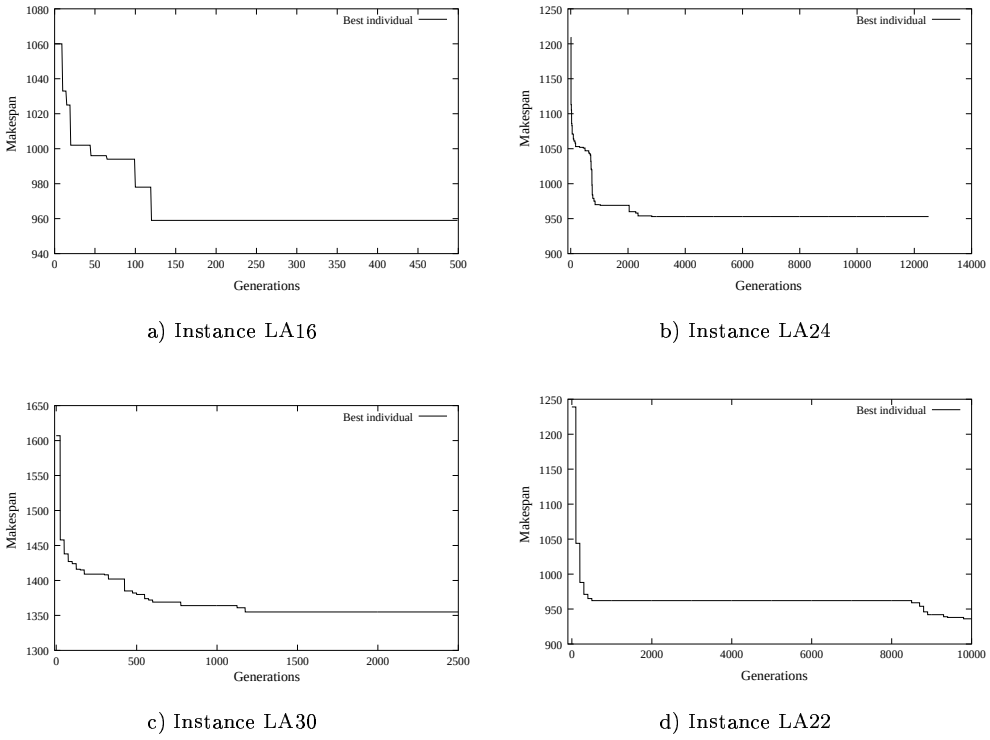


Figure 4. Example of the mutation operator modified through the use of situational knowledge.



a) Instance LA16

b) Instance LA24

c) Instance LA30

d) Instance LA22

Figure 5. Convergence graphs of CULT for 4 examples

Table 1. Example of a 3×3 JSSP.

	machine (time)		
0	0(3)	1(3)	2(3)
1	0(2)	2(3)	1(4)
2	1(3)	0(2)	2(1)

Table 2. Example of a 6 × 6 JSSP (FT06, taken from the OR-Library).

	machine (time)					
1	2(1)	0(3)	1(6)	3(7)	5(3)	4(6)
2	1(8)	2(5)	4(10)	5(10)	0(10)	3(4)
3	2(5)	3(4)	5(8)	0(9)	1(1)	4(7)
4	1(5)	0(5)	2(5)	3(3)	4(8)	5(9)
5	2(9)	1(3)	4(5)	5(4)	0(3)	3(1)
6	1(3)	3(3)	5(9)	0(10)	4(4)	2(1)

Table 3. Deviations from the best known solutions and improvement of CULT.

	Deviation	**Improvement**
CULT	0.36%	—
SB	1.38%	1.02%
GRASP	1.77%	1.41%
Parallel GRASP	0.43%	0.07%
GA	0.90%	0.54%
HGA	0.39%	0.03%

Table 4. Overall performance of CULT with respect to the 5 other algorithms against which it was compared.

	vs CULT		
	Win	Tie	Lose
SB	0	20	23
GRASP	0	24	19
Parallel GRASP	4	32	7
GA	1	22	20
HGA	4	31	8

Table 5. Comparison of results per instance. **Boldface** is used to show both the best known solution and the cases in which an algorithm reached such value.

Instance	Size	BKS	CULT	SB	GRASP	Parallel GRASP	GA	HGA
FT06	6×6	**55**	**55**	**55**	**55**	**55**	**55**	**55**
FT10	10×10	**930**	**930**	**930**	938	**930**	936	**930**
FT20	20×5	**1165**	**1165**	1178	1169	**1165**	1177	**1165**
LA01	10×5	**666**	**666**	**666**	**666**	**666**	**666**	**666**
LA02	10×5	**655**	**655**	669	**655**	**655**	666	**655**
LA03	10×5	**597**	**597**	605	604	**597**	**597**	**597**
LA04	10×5	**590**	**590**	593	**590**	**590**	**590**	**590**
LA05	10×5	**593**	**593**	**593**	**593**	**593**	**593**	**593**
LA06	15×5	**926**	**926**	**926**	**926**	**926**	**926**	**926**
LA07	15×5	**890**	**890**	**890**	**890**	**890**	**890**	**890**
LA08	15×5	**863**	**863**	**863**	**863**	**863**	**863**	**863**
LA09	15×5	**951**	**951**	**951**	**951**	**951**	**951**	**951**
LA10	15×5	**958**	**958**	959	**958**	**958**	**958**	**958**
LA11	20×5	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**	**1222**
LA12	20×5	**1039**	**1039**	**1039**	**1039**	**1039**	**1039**	**1039**
LA13	20×5	**1150**	**1150**	**1150**	**1150**	**1150**	**1150**	**1150**
LA14	20×5	**1292**	**1292**	**1292**	**1292**	**1292**	**1292**	**1292**
LA15	20×5	**1207**	**1207**	**1207**	**1207**	**1207**	**1207**	**1207**
LA16	10×10	**945**	**945**	978	946	**945**	977	**945**
LA17	10×10	**784**	**784**	787	**784**	**784**	787	**784**
LA18	10×10	**848**	**848**	859	**848**	**848**	**848**	**848**
LA19	10×10	**842**	**842**	860	**842**	**842**	857	**842**
LA20	10×10	**902**	907	914	907	**902**	910	907
LA21	15×10	**1046**	1059	1084	1091	1057	1047	**1046**
LA22	15×10	**927**	930	944	960	**927**	936	935
LA23	15×10	**1032**	**1032**	**1032**	**1032**	**1032**	**1032**	**1032**
LA24	15×10	**935**	950	976	978	954	955	953
LA25	15×10	**977**	984	1017	1028	984	1004	986
LA26	20×10	**1218**	**1218**	1224	1271	**1218**	**1218**	**1218**
LA27	20×10	**1235**	1253	1291	1320	1269	1260	1256
LA28	20×10	**1216**	1224	1250	1293	1225	1241	1232
LA29	20×10	**1157**	1186	1239	1293	1203	1190	1196
LA30	20×10	**1355**	**1355**	**1355**	1368	**1355**	1356	**1355**
LA31	30×10	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**	**1784**
LA32	30×10	**1850**	**1850**	**1850**	**1850**	**1850**	**1850**	**1850**
LA33	30×10	**1719**	**1719**	**1719**	**1719**	**1719**	**1719**	**1719**
LA34	30×10	**1721**	**1721**	**1721**	1753	**1721**	1730	**1721**
LA35	30×10	**1888**	**1888**	**1888**	**1888**	**1888**	**1888**	**1888**
LA36	15×15	**1268**	1281	1305	1334	1287	1305	1279
LA37	15×15	**1397**	1407	1423	1457	1410	1441	1408
LA38	15×15	**1196**	1215	1255	1267	1218	1248	1219
LA39	15×15	**1233**	1251	1273	1290	1248	1264	1246
LA40	15×15	**1222**	1244	1269	1259	1244	1252	1241

Table 6. Comparison of the number of evaluations performed by CULT and the number of iterations performed by GRASP and Parallel GRASP.

Instance	CULT	GRASP	Parallel GRASP
FT06	100	100,000	10
FT10	1,500,000	90,100,000	2,500,000
FT20	2,000,000	90,100,000	4,500,000
LA01	500	100,000	100
LA02	100,000	100,000	4,000
LA03	100,000	50,100,000	10,000
LA04	15,000	100,000	1,000
LA05	100	100,000	100
LA06	100	100,000	100
LA07	500	100,000	100
LA08	1,000	100,000	300
LA09	1,000	100,000	100
LA10	100	100,000	100
LA11	100	100,000	100
LA12	500	100,000	100
LA13	500	100,000	100
LA14	1,000	100,000	100
LA15	1,000	100,000	200
LA16	10,000	50,100,000	1,300,000
LA17	10,000	20,100,000	20,000
LA18	50,000	20,100,000	50,000
LA19	50,000	10,100,000	20,000
LA20	50,000	50,100,000	17,000,000
LA21	200,000	50,100,000	100,000,000
LA22	200,000	50,100,000	26,000,000
LA23	200,000	10,100,000	10,000
LA24	250,000	10,100,000	125,000,000
LA25	250,000	10,100,000	32,000,000
LA26	1,500,000	10,100,000	3,500,000
LA27	500,000	10,100,000	10,500,000
LA28	500,000	10,100,000	20,000,000
LA29	2,000,000	10,100,000	50,000,000
LA30	50,000	10,100,000	3,000,000
LA31	5,000	10,100,000	10,000
LA32	10,000	10,100,000	100
LA33	10,000	10,100,000	1,000
LA34	20,000	10,100,000	50,000
LA35	10,000	10,100,000	10,000
LA36	500,000	11,200,000	51,000,000
LA37	500,000	11,200,000	20,000,000
LA38	750,000	11,200,000	20,000,000
LA39	750,000	11,200,000	6,000,000
LA40	750,000	11,200,000	2,000,000

Table 7. Results obtained by CULT when fixing the number of fitness function evaluations to 200,000. **Boldface** is used to show both the best known solution and the cases in which CULT reached such value.

Instance	Size	BKS	Best	Median	Worst
LA01	10×5	**666**	**666**	666.5	668
LA02	10×5	**655**	**655**	660.5	667
LA03	10×5	**597**	**597**	610.2	623
LA04	10×5	**590**	**590**	593	599
LA05	10×5	**593**	**593**	593.5	595
LA06	15×5	**926**	**926**	926	926
LA07	15×5	**890**	**890**	890	890
LA08	15×5	**863**	**863**	863.4	865
LA09	15×5	**951**	**951**	951.3	953
LA10	15×5	**958**	**958**	958.1	959
LA11	20×5	**1222**	**1222**	1222.3	1224
LA12	20×5	**1039**	**1039**	1039.3	1041
LA13	20×5	**1150**	**1150**	1150.4	1152
LA14	20×5	**1292**	**1292**	1292.4	1294
LA15	20×5	**1207**	**1207**	1207.8	1209
LA16	10×10	**945**	**945**	962.8	990
LA17	10×10	**784**	**784**	793.4	811
LA18	10×10	**848**	**848**	857.5	863
LA19	10×10	**842**	**842**	859	872
LA20	10×10	**902**	907	912.6	924
LA21	15×10	**1046**	1059	1093	1114
LA22	15×10	**927**	947	964.3	985
LA23	15×10	**1032**	**1032**	1035.5	1045
LA24	15×10	**935**	950	976.6	997
LA25	15×10	**977**	998	1010.7	1034
LA26	20×10	**1218**	1219	1234.4	1260
LA27	20×10	**1235**	1279	1300	1324
LA28	20×10	**1216**	1236	1260.7	1291
LA29	20×10	**1157**	1219	1238.8	1271
LA30	20×10	**1355**	**1355**	1357.5	1369
LA31	30×10	**1784**	**1784**	1784	1784
LA32	30×10	**1850**	**1850**	1850.1	1851
LA33	30×10	**1719**	**1719**	1719	1719
LA34	30×10	**1721**	**1721**	1721	1721
LA35	30×10	**1888**	**1888**	1888	1888
LA36	15×15	**1268**	1296	1316.7	1351
LA37	15×15	**1397**	1416	1464.9	1514
LA38	15×15	**1196**	1231	1265.3	1276
LA39	15×15	**1233**	1269	1294.6	1327
LA40	15×15	**1222**	1256	1277.4	1328