A Cultural Algorithm for Solving the Job Shop Scheduling Problem

Ricardo Landa Becerra and 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

Summary. In this chapter, we propose an approach for solving the job shop scheduling problem using a cultural algorithm. 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 and to a Parallel Genetic Algorithm. The cultural algorithm proposed 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.

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 [38]. Scheduling has been a very active research area during several years, both in the operations research and in the computer science literature [4, 3, 29]. 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 [47, 10, 11], tabu search [5, 45], simulated annealing [46, 9], the ant system [15], and artificial immune systems [25, 24, 16], among others.

Note however, that this chapter presents the first attempt (to the authors' best knowledge) to use cultural algorithms to solve job shop scheduling problems. Cultural algorithms [40] 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 chapter is the use of a combination of knowledge extracted during the evolutionary search with some knowledge that is inserted *a priori* because it is normally known to be useful in the job scheduling problem. Our 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) and a Parallel Genetic Algorithm in several test problems taken from the specialized literature. Our results indicate that the proposed approach is a viable alternative for solving efficiently job shop scheduling problems.

The remainder of this chapter is organized as follows: in Section 2 we provide a brief description of the statement of the problem that we wish to solve. 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 our proposed approach to solve job shop scheduling problems using a cultural algorithm. As part of this section, we include a description of the representation of solutions adopted in our work as well as the mechanisms implemented to add domain knowledge to our evolutionary algorithm both before and during the search process. Section 5 provides a comparative study. Finally, Section 6 presents our general conclusions and some possible paths for future research.

2 Problem Statement

We can define the job shop scheduling problem (JSSP) in the following way: we have a set of jobs, $j_1, j_2, \ldots, j_n$ that we need to process in a set of machines, $m_1, m_2, \ldots, m_m$. The processing of job j_j in the machine m_r is an operation that requires of a time p_{jr}. 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, we will say that we are dealing with $n \times m$ job shop scheduling problems. A schedule is then a set of duration times for each operation $\{c_{jr}\}_{1 \leq j \leq n, 1 \leq r \leq m}$ 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 chapter, 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 [4].

Garey and Johnson [31] showed that the JSSP is an **NP-hard** problem and within its class it is one of the least tractable problems [3]. To have an idea of the difficulty of the JSSP, it is reported that a famous 10 × 10 instance formulated by the first time by Muth and Thompson in 1963 [36], was exactly solved until 1989 by Carlier and Pinson using a branch and bound algorithm [8]. 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* [28]. However, other algorithms, such as an approach called *shifting bottleneck* (SB) have been found to be very effective in practice [1]. Furthermore, a number of heuristics have also been used in the JSSP (e.g., genetic algorithms [3, 47], tabu search [5], simulated annealing [9], artificial immune systems [24], among others), as indicated before.

An instance of the JSSP can be formulated in tabular form as indicated in Table 1, where we show a 3×3 problem. 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 p_{jr} that takes to the job to be processed in that machine.

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

Table 1. Example of a 3 × 3 job shop scheduling problem.

3 Cultural Algorithms

Cultural algorithms were developed by Robert G. Reynolds as a complement to the metaphor used by evolutionary algorithms [18], which had focused mainly on genetic and natural selection concepts [40].

Cultural algorithms are based on some theories originated in sociology and archaeology which try to model cultural evolution (see for example [39, 17]). 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, we can clearly appreciate 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 attempts to capture this double inheritance phenomenon through his proposal of cultural algorithms [40]. 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 [20].

Cultural algorithms operate in two spaces. First, we have 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 from a set of operators applied to the population.

The second space is the belief space, which is where we store the knowledge acquired by individuals through generations. 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 pseudo-code of a cultural algorithm is shown in Algorithm 1.

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 Algorithm 1, we have the update of the belief space. It is at this point in which the belief space incorporates the individual experiences of a

Algorithm 1 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
While the end condition is not satisfied
```

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 [41].

In [40], Reynolds proposed the use of genetic algorithms [22] to model the micro-evolutionary process, and Version Spaces [35] 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 we apply a cultural algorithm for global optimization, the acceptable beliefs can be seen as constraints that direct the population at the micro-evolutionary level [32].

In genetic algorithms' theory, there is an expression, called *schema theorem* [26] that represents a bound on the speed at which the best schemata of the population are propagated. Reynolds [40] 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 [12, 14]).

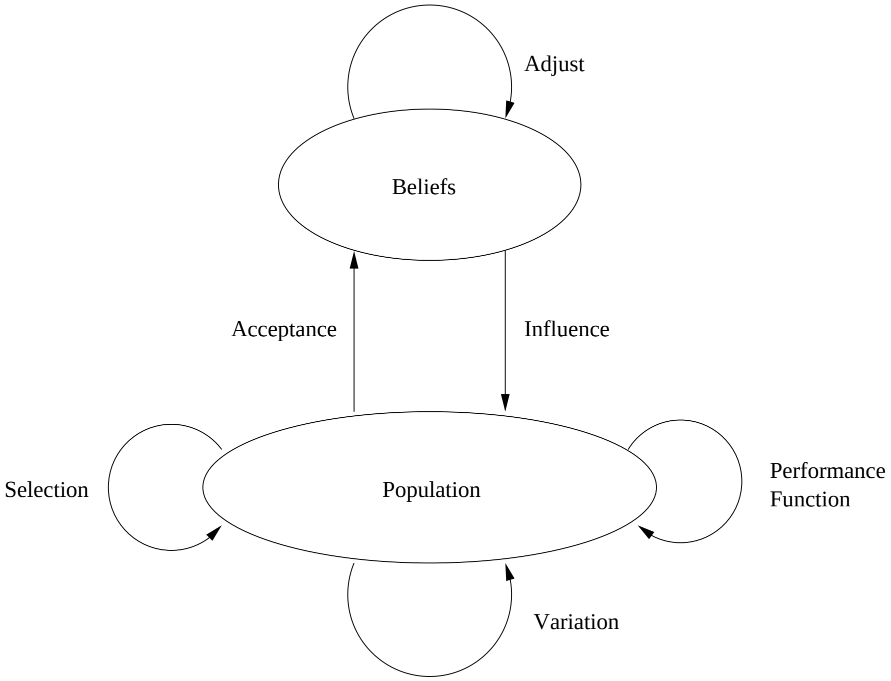

Fig. 1. Spaces of a cultural algorithm

4 Our Proposed Approach

The approach proposed in this chapter uses as its population space to the population adopted by evolutionary programming [19], together with its selection and variation operators.

In evolutionary programming, we have p individuals in the original population (such individuals are randomly generated). In the main loop of the evolutionary programming algorithm only mutation is applied (this is because this approach simulates evolution at the species level, and different species do not recombine among themselves [18]), besides selection.

The mutation operator obtains a child from each of the individuals in the population (i.e., p children are obtained). In the case of continuous optimization, the mutation operator consists of adding Gaussian noise to each variable [14]. In our case, since we are dealing with a combinatorial optimization problem, we use a set of exchanges in the sequence of the operations as our 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 p individuals with the largest number of wins are selected to constitute the population at the next generation.

All of these steps are similar to the algorithm proposed in this paper. However, in our case, we integrate the influence of the belief space in our evolutionary algorithm. This process will be explained later in the chapter.

4.1 Representation

The representation adopted to encode our solutions plays a very important role when applying an evolutionary computation technique [44, 43], and this issue plays a crucial role when specifically dealing with the job shop scheduling problem [47]. This is due to the fact that a solution to the job scheduling problem cannot be represented as a permutation as normally done in many combinatorial optimization problems [42, 33].

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 job shop scheduling problem of size $n \times m$, several types of possible encodings have been proposed in the literature. Some examples are the use of binary encoding [37], the use of disjunctive graphs [15], and the permutations with repetitions [7].

Most of the existing encodings can generate invalid schedules and thus require a repair mechanism [34]. These repair mechanisms tend to bias solutions towards a certain region of the search space and are, therefore, not always advisable [13].

The permutation with repetitions has the advantage of never generating invalid schedules and that was precisely the main reason for which we decided to adopt it for our approach. This representation consists of a permutation in which each component is repeated m times. The components of the permutation represent jobs as in our 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, we show an example of the decoding of a permutation with repetitions for the problem described in Table 1.

The main disadvantage of this representation is that different permutations can encode the same schedule. We found in our experiments 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.

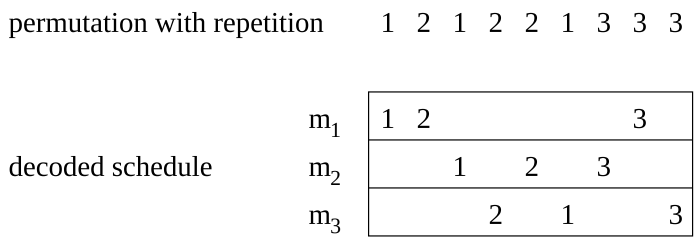

Fig. 2. Example of the decoding process of a permutation with repetitions.

4.2 Domain Knowledge Added *a priori* to the Algorithm

In order to incorporate domain knowledge into our algorithm, we use two mechanism. The first of them involves adding knowledge *a priori* (i.e., before actually running the algorithm), whereas the second involves extracting information during the execution of the algorithm following the traditional model of a cultural algorithm. We will proceed to describe first the addition of *a priori* domain knowledge.

This mechanism is integrated during the evaluation of a new individual. First, we define a *semi-active* schedule 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 large 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*. Figure 3 shows the same schedule from Figure 2 before and after applying permissible left shifts. 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 chapter, permissible left shifts are applied during the evaluation of an individual. The individual to be evaluated is applied all the permissible left shifts possible and then its genes are modified as to encode the new corresponding (active) schedule.

Since these modifications are *ad hoc* to the job shop scheduling problems, and given that they are encoded in the algorithm prior to its execution, we call them insertion of *a priori* domain knowledge.

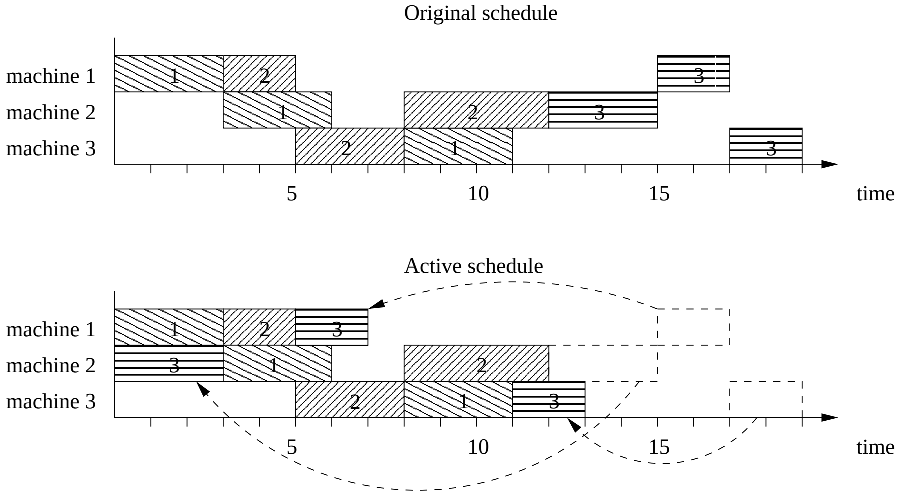

Fig. 3. Left shifts applied to obtain an *active* schedule.

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 and the use of a mechanism to enforce that new individuals generated are similar to this one, is used in our case to accelerate convergence.

This is precisely the main idea on which we based the design of the belief space of our approach. The belief space contains a part called situational knowledge, which has previously been used for continuous optimization [12, 27]. Situational knowledge consists of storing the best individual found so far, and use it as a leader that other solutions must follow.

In this case, 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.

We modify the mutation operator in order to cause that the other solutions have a greater "resemblance" (i.e., that their values are more similar) with respect to the individual stored in the situational part. The mutation operator that we propose is based on swaps of components in the permutation with repetitions.

In order to perform each of these swaps, we first locate the first match between the individual to be mutated and the individual stored in the situational part, beginning from a random position. Once we find this match, we try to increase the coincidence between the two solutions compared. This is done by extending it to the next component that is different between the two individuals compared. Since we are dealing with permutations, it is necessary to make exchanges in the components in order to keep the permutation as valid.

An example of this process is shown in Figure 4. Let's suppose that the random initial position is location 2. If the string containing the permutation finishes and no match is found, we continue searching from the beginning of the string. In the example shown in Figure 4, the match occurs in the same location 2. In location 3, the values are different, and thus we take this position as the first exchange point. The second exchange point is the location of the component that matches the next value in the individual stored in the situational part. In the example shown in Figure 4, it is location 6, which contains a value of 2.

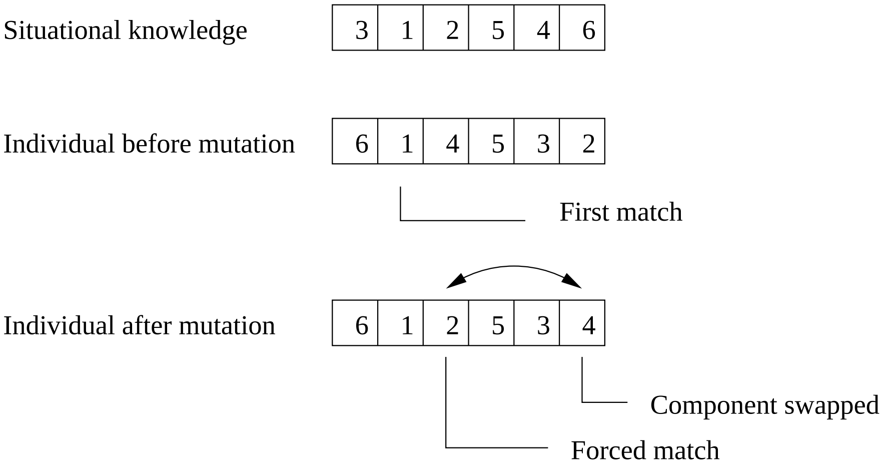

Fig. 4. Example of the mutation operator modified through the use of situational knowledge.

If, when attempting to perform an exchange (swap), we find that the individual to be mutated and the individual stored in the situational part are exactly the same, then the exchange points will be randomly selected.

The permutations with repetitions adopted in this work, have a size $n \times m$. Thus, by performing $(n \times m)/2$ exchanges, we would completely reorder the entire string (each exchange reorders 2 individuals). In order to have a mutation operator that could perform the corresponding perturbations in an efficient manner and, that at the same time, could allow an appropriate balance in terms of exploration and exploitation of the search space, we decided to set the number of swaps (or exchanges) to be performed to a certain individual (in order to generate an offspring) as a random number between 1 and $(n \times m)/2$.

The update of the belief space consists only of replacing the individual stored in the situational knowledge with the best individual found in the current population.

It is worth clarifying that the update of the belief space is not done at every generation. We perform this update at every g_{belief} generations, in order to avoid an excessive selection pressure that would cause convergence to a local optimum.

5 Comparison of Results

We compare our Cultural Algorithm (CULT) with respect to 2 different approaches: a Greedy Randomized Adaptive Search Procedure (GRASP) approach reported in [2], and a Parallel Genetic Algorithm (PGA) reported in [23]. We chose these references for three main reasons: (1) they provide enough information (e.g., numerical results) as to allow a comparison, (2) these algorithms have been been found to be very powerful in the job shop scheduling problem studied in this paper, and (3) the comparison with respect to another evolutionary algorithm that does not use knowledge as ours was an important issue for us. In this regard, to compare our results with respect to a parallel genetic algorithm was of particular importance, since we hypothesize that the use of knowledge would make our sequential cultural algorithm as competitive as a parallel genetic algorithm.

The benchmark adopted for our experiments is a subset of the job shop scheduling instances contained in the OR-library [6]. 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 our particular case, we adopted the 40 problems of Lawrence [30], labeled from LA01 to LA40.

All our tests were performed on a PC with an Intel Pentium III processor running at 866 MHz with 256 MB of RAM and using Red Hat Linux 7.3. Our approach was implemented in C language and was compiled using the GNU gcc compiler.

	deviation	CULT	Improvement
GRASP	0.45%	0.96%	-0.48%
PGA	0.92%	0.96%	-0.03%

Table 2. Comparison of results between our Cultural Algorithm (CULT) and two other algorithms: Greedy Randomized Adaptive Search Procedure (GRASP) [2] and Parallel Genetic Algorithm (GA) [23].

Table 2 shows the overall comparison of results. In the first column, we show the algorithm with respect to which we are comparing our results. In the second column, we show the average deviation of the best results obtained by each algorithm with respect to the best known solution for the 40 test problems adopted in our study. In the third column, we show the average deviation of our cultural algorithm with respect to the best known solution for the 40 test problems adopted in our study. The last column indicates the improvement achieved by our cultural algorithm with respect to each of the other algorithms compared. From Table 2, we can see that our approach was not able to improve on the results produced by any of the two algorithms compared (this is indicated by the negative values obtained in the last column of the table), but it practically tied in terms of overall performance with the PGA. Note however, that our approach performed a considerably lower number of evaluations than GRASP (except for one problem) and was implemented sequentially and not in parallel as the PGA. Despite these two important differences, our cultural algorithm produced competitive results with respect to these two other techniques.

	CULT		
	Win	Tie	Lose
GRASP	13	26	1
PGA	9	21	10

Table 3. Overall performance of our cultural algorithm (CULT) with respect to the 2 other algorithms against which it was compared. The column labeled **Win** shows the number of problems in which each algorithm beat our cultural algorithm. The column labeled **Tie** indicates ties between our cultural algorithm and the other algorithms. Finally, the column labeled **Lose** indicates the number of problems in which each algorithm lost with respect to our cultural algorithm.

In Table 3, we show the overall performance of our cultural algorithm with respect to the 2 other algorithms against which it was compared. Results indicate that GRASP beat our cultural algorithm in 13 problems and lost only in 1. This performance is, however, associated with a much higher computational cost, as we will see later on. Regarding the PGA, it beats our cultural algorithm in 9 problems and loses in 10. This indicates that the PGA has practically the same performance as our approach. Note however, that our approach is implemented sequentially and not in parallel as the PGA.

Table 4 compares the best results found by our cultural algorithm, the Greedy Randomized Adaptive Search Procedure (GRASP) approach reported in [2], and a Parallel Genetic Algorithm (PGA) reported in [23]. We use **boldface** to indicate both the best known results and when an algorithm reached such result. We do not include the number of evaluations performed by the PGA, because that information is not provided by the authors of the approach.

Instance	**Size**	**BKS**	**CULT**	**Evals (CULT)**	**GRASP**	**Iters (GRASP)**	**PGA**
LA01	10×5	**666**	**666**	4000	**666**	100000	**666**
LA02	10×5	**655**	**655**	20000	**655**	100000	666
LA03	10×5	**597**	603	8000	**597**	50.1×10^6	**597**
LA04	10×5	**590**	**590**	700000	**590**	100000	**590**
LA05	10×5	**593**	**593**	2000	**593**	100000	**593**
LA06	15×5	**926**	**926**	2000	**926**	100000	**926**
LA07	15×5	**890**	**890**	4000	**890**	100000	**890**
LA08	15×5	**863**	**863**	4000	**863**	100000	**863**
LA09	15×5	**951**	**951**	2000	**951**	100000	**951**
LA10	15×5	**958**	**958**	2000	**958**	100000	**958**
LA11	20×5	**1222**	**1222**	2000	**1222**	100000	**1222**
LA12	20×5	**1039**	**1039**	2000	**1039**	100000	**1039**
LA13	20×5	**1150**	**1150**	2000	**1150**	100000	**1150**
LA14	20×5	**1292**	**1292**	2000	**1292**	100000	**1292**
LA15	20×5	**1207**	**1207**	8000	**1207**	100000	**1207**
LA16	10×10	**945**	946	25000	**945**	50.1×10^6	977
LA17	10×10	**784**	**784**	25000	**784**	20.1×10^6	787
LA18	10×10	**848**	**848**	140000	**848**	20.1×10^6	**848**
LA19	10×10	**842**	**842**	32000	**842**	10.1×10^6	857
LA20	10×10	**902**	907	100000	**902**	50.1×10^6	910
LA21	15×10	**1046**	1089	1700000	1057	50.1×10^6	1047
LA22	15×10	**927**	945	1700000	**927**	50.1×10^6	936
LA23	15×10	**1032**	**1032**	200000	**1032**	10.1×10^6	**1032**
LA24	15×10	**935**	964	1000000	954	10.1×10^6	955
LA25	15×10	**977**	993	1000000	984	10.1×10^6	1004
LA26	20×10	**1218**	**1218**	1700000	**1218**	10.1×10^6	**1218**
LA27	20×10	**1235**	1269	200000	1269	10.1×10^6	1260
LA28	20×10	**1216**	1241	1800000	1225	10.1×10^6	1241
LA29	20×10	**1157**	1189	1800000	1203	10.1×10^6	1190
LA30	20×10	**1355**	**1355**	200000	**1355**	10.1×10^6	1356
LA31	30×10	**1784**	**1784**	15000	**1784**	10.1×10^6	**1784**
LA32	30×10	**1850**	**1850**	40000	**1850**	10.1×10^6	**1850**
LA33	30×10	**1719**	**1719**	20000	**1719**	10.1×10^6	**1719**
LA34	30×10	**1721**	**1721**	160000	**1721**	10.1×10^6	1730
LA35	30×10	**1888**	**1888**	160000	**1888**	10.1×10^6	**1888**
LA36	15×15	**1268**	1292	1000000	1287	11.2×10^6	1305
LA37	15×15	**1397**	1451	1400000	1410	11.2×10^6	1441
LA38	15×15	**1196**	1276	1800000	1218	11.2×10^6	1248
LA39	15×15	**1233**	1266	200000	1248	11.2×10^6	1264
LA40	15×15	**1222**	1265	1000000	1244	11.2×10^6	1252

Table 4. Comparison of results between our cultural algorithm (CULT), GRASP (Greedy Randomized Adaptive Search Procedure) [2], and PGA (Parallel Genetic Algorithm) [23]. Only the number of iterations of GRASP and our cultural algorithm are reported because this value was not available for the PGA. We show in **boldface** both the best known solution and the cases in which an algorithm reached such value.

In all the examples, we performed 10 independent runs of our algorithm. Note that the number of objective function evaluations performed by our cultural algorithm is variable. The criterion adopted to stop our algorithm was to detect when no changes in the result were reported after a certain (normally large) number of consecutive iterations. The number of evaluations reported is then the average value obtained from the 10 independent runs performed. That is the reason for the large variability of values.

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

$$\begin{aligned} p &= 20 \\ c &= \frac{p}{2} = 10 \\ g_{beliefs} &= 50 \end{aligned}$$

where p is the population size, c is the number of binary confrontations to be performed by each individual during the tournament selection, and $g_{beliefs}$ is the update frequency (measured in terms of generations) of the belief space.

Note that, except for LA04, in all the problems, our approach performed less evaluations than GRASP and in several instances, the difference is remarkable.[1] Some of the most remarkable examples are the following (note that in the following, we will be treating the iterations of GRASP as evaluations):

- **LA30**: In this problem both GRASP and our cultural algorithm can reach the best known solution. However, GRASP requires 10.1 million evaluations, whereas our approach only requires 200,000.
- **LA27**: In this problem, none of the 3 approaches converged to the best known solution, and both GRASP and our cultural algorithm converged to the same solution. However, our cultural algorithm required 250,000 evaluations whereas GRASP required 10.1 million evaluations.
- **LA17**: Both GRASP and our cultural algorithm found the best known solution. However, GRASP required 20.1 million evaluations and our approach required only 250,000 evaluations.
- **LA13**: Both GRASP and our cultural algorithm found the best known solution. However, GRASP required 100,000 evaluations, and our approach required only 2,000 evaluations.
- **LA16**: Our cultural algorithm converges to a solution that is only 1 unit away from the best known solution. GRASP, in contrast, converges to the best known solution. However, GRASP required 50.1 million evaluations, whereas our cultural algorithm only performed 250,000 evaluations.

---

[1] In fact, as noted in Table 4, we report the number of iterations performed by GRASP. However, at each iteration, GRASP performs several evaluations of the objective function. Thus, the real number of evaluations of GRASP is much higher than the values reported in Table 4.

Looking for a compromise to setup *a priori* a maximum number of objective function (or fitness) evaluations, we suggest to use a population size of 20 individuals, and a maximum number of generations of 10,000. 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 5. Note that in this case we also report the median and worst results found by our approach in each case. 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 our cultural algorithm. In any case, more work in this direction is desirable as to improve the search capabilities of our approach while maintaining a relatively low number of fitness function evaluations.

6 Conclusions and Future Work

We have introduced a new approach based on a cultural algorithm to solve job shop scheduling problems. The approach uses both knowledge introduced *a priori* (i.e., a heuristic to perform local rearrangements that we know beforehand that can reduce the makespan) and extracted during the evolutionary search. Our 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, our approach performs a number of evaluations that is (on average) considerably lower than those performed by GRASP [2] while producing similar results. Results are also competitive (there is practically a tie) with respect to a parallel genetic algorithm, despite the fact that our results were obtained with a sequential version of our cultural algorithm.

As part of our future work, we plan to improve the heuristics adopted to perform local moves. We also intend to introduce a backtracking mechanism to recover from movements towards local attractors and we also plan to incorporate into our algorithm certain mechanisms from tabu search [21]. Additionally, the introduction of parallelism in our approach is another possible path of future research.

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 we currently report for our algorithm.

Finally, we also plan to work on a multiobjective version of the job shop scheduling problem in which 3 objectives would be considered [3]: 1) makespan, 2) mean flowtime and 3) mean tardiness. This would allow us to

Instance	**Size**	**BKS**	**Best**	**Median**	**Worst**
LA01	10 × 5	**666**	**666**	666	667
LA02	10 × 5	**655**	**655**	666	672
LA03	10 × 5	**597**	603	617	633
LA04	10 × 5	**590**	593	600	611
LA05	10 × 5	**593**	**593**	593	593
LA06	15 × 5	**926**	**926**	926	926
LA07	15 × 5	**890**	**890**	890	897
LA08	15 × 5	**863**	**863**	863	863
LA09	15 × 5	**951**	**951**	951	951
LA10	15 × 5	**958**	**958**	958	958
LA11	20 × 5	**1222**	**1222**	1222	1222
LA12	20 × 5	**1039**	**1039**	1039	1039
LA13	20 × 5	**1150**	**1150**	1150	1150
LA14	20 × 5	**1292**	**1292**	1292	1292
LA15	20 × 5	**1207**	**1207**	1207	1225
LA16	10 × 10	**945**	946	982	995
LA17	10 × 10	**784**	**784**	792	809
LA18	10 × 10	**848**	**848**	861	897
LA19	10 × 10	**842**	**842**	862	891
LA20	10 × 10	**902**	907	917	947
LA21	15 × 10	**1046**	1096	1124	1132
LA22	15 × 10	**927**	950	977	995
LA23	15 × 10	**1032**	**1032**	1032	1085
LA24	15 × 10	**935**	974	998	1021
LA25	15 × 10	**977**	996	1021	1056
LA26	20 × 10	**1218**	1228	1257	1319
LA27	20 × 10	**1235**	1269	1312	1359
LA28	20 × 10	**1216**	1254	1297	1323
LA29	20 × 10	**1157**	1245	1253	1302
LA30	20 × 10	**1355**	**1355**	1373	1412
LA31	30 × 10	**1784**	**1784**	1784	1784
LA32	30 × 10	**1850**	**1850**	1850	1852
LA33	30 × 10	**1719**	**1719**	1719	1719
LA34	30 × 10	**1721**	**1721**	1721	1770
LA35	30 × 10	**1888**	**1888**	1888	1902
LA36	15 × 15	**1268**	1321	1347	1380
LA37	15 × 15	**1397**	1467	1496	1543
LA38	15 × 15	**1196**	1286	1308	1372
LA39	15 × 15	**1233**	1266	1305	1370
LA40	15 × 15	**1222**	1273	1292	1312

Table 5. Results obtained by our cultural algorithm when fixing the maximum number of fitness function evaluations to 200,000. We show in **boldface** both the best known solution and the cases in which our algorithm reached such value.

generate trade-offs that the user could evaluate in order to decide what solution to choose.

Acknowledgments

The first 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 second author acknowledges support from CONACyT through project No. 34201-A.

References

1. J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.
2. Renata M. Aiex, S. Binato, and Mauricio G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29(4):393–430, 2003.
3. Tapan P. Bagchi. *Multiobjective Scheduling by Genetic Algorithms.* Kluwer Academic Publishers, New York, September 1999. ISBN 0-7923-8561-6.
4. Kenneth R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons, New York, 1974.
5. J.W. Barnes and J.B. Chambers. Solving the Job Shop Scheduling Problem using Tabu Search. *IIE Transactions*, 27(2):257–263, 1995.
6. J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.
7. C. Bierwirth. A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms. *OR Spektrum*, 17:87–92, 1995.
8. J. Carlier and E. Pinson. An algorithm for solving the Job-Shop problem. *Management Science*, 35(2):164–176, 1989.
9. Olivier Catoni. Solving Scheduling Problems by Simulated Annealing. *SIAM Journal on Control and Optimization*, 36(5):1539–1575, September 1998.
10. R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms: I. Representation. *Computers and Industrial Engineering*, 30:983–997, 1996.
11. R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms: II. Hybrid genetic search strategies. *Computers and Industrial Engineering*, 36(2):343–364, 1999.
12. Chan-Jin Chung and Robert G. Reynolds. CAEP: An Evolution-based Tool for Real-Valued Function Optimization using Cultural Algorithms. *Journal on Artificial Intelligence Tools*, 7(3):239–292, 1998.
13. Carlos A. Coello Coello. 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*, 191(11–12):1245–1287, January 2002.

14. Carlos A. Coello Coello and Ricardo Landa Becerra. Adding Knowledge and Efficient Data Structures to Evolutionary Programming: A Cultural Algorithm for Constrained Optimization. In W.B. Langdon, E.Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A.C. Schultz, J. F. Miller, E. Burke, and N.Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pages 201–209, San Francisco, California, July 2002. Morgan Kaufmann Publishers.
15. Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, and Marco Trubian. Ant system for Job-shop scheduling. *JORBEL–Belgian Journal of Operations Research, Statistics and Computer Science*, 34:39–53, 1994.
16. Xunxue Cui, Miao Li, and Tingjian Fang. 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)*, volume 2, pages 1316–1321, Piscataway, New Jersey, May 2001. IEEE Service Center.
17. W. H. Durham. *Co-evolution: Genes, Culture, and Human Diversity.* Stanford University Press, Stanford, California, 1994.
18. David B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence.* The Institute of Electrical and Electronic Engineers, New York, 1995.
19. Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming.* John Wiley & Sons, Inc., New York, 1999.
20. Benjamin Franklin and Marcel Bergerman. Cultural algorithms: Concepts and experiments. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 1245–1251, Piscataway, New Jersey, 2000. IEEE Service Center.
21. Fred Glover and Manuel Laguna. *Tabu Search.* Kluwer Academic Publishers, Norwell Massachusetts, 1998.
22. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.
23. José Fernando Gonçalves and N.C.Beirao. Um algoritmo genético baseado em chaves aleatórias para sequenciamiento de operaçoes. *Revista Revista Associação Portuguesa de Desenvolvimento e Investigação Operacional*, 19:123 – 137, 1999. (in Portuguese).
24. Emma Hart and Peter Ross. The Evolution and Analysis of a Potential Antibody Library for Use in Job-Shop Scheduling. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 185–202. McGraw-Hill, London, 1999.
25. Emma Hart, Peter Ross, and J. Nelson. Producing robust schedules via an artificial immune system. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*, pages 464–469, Anchorage, Alaska, 1998. IEEE Press.
26. John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.
27. Xidong Jin and Robert G. Reynolds. Using Knowledge-Based Evolutionary Computation to Solve Nonlinear Constraint Optimization Problems: a Cultural Algorithm Approach. In *1999 Congress on Evolutionary Computation*, pages 1672–1678, Washington, D.C., July 1999. IEEE Service Center.
28. Albert Jones and Luis C. Rabelo. Survey of Job Shop Scheduling Techniques. NISTIR, National Institute of Standards and Technology, 1998.

29. E.G. Coffman Jr. *Computer and Job Shop Scheduling Theory.* John Wiley and Sons, 1976.
30. Stephen R. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984. (Unpublished).
31. David S. Johnson Michael R. Garey. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences).* W H Freeman & Co., June 1979. ISBN 0-7167-1045-5.
32. Zbigniew Michalewicz. A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155. The MIT Press, Cambridge, Massachusetts, 1995.
33. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, third edition, 1996.
34. Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.
35. Tom Mitchell. *Version Spaces: An Approach to Concept Learning.* PhD thesis, Computer Science Department, Stanford University, Stanford, California, 1978.
36. J.F. Muth and G.L. Thompson. *Industrial Scheduling.* Prentice-Hall, Englewood Cliffs, New Jersey, 1963.
37. Ryohei Nakano and Takeshi Yamada. Conventional Genetic Algorithm for Job Shop Problems. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA-91)*, pages 474–479, San Mateo, California, 1991. University of California, San Diego, Morgan Kaufmann Publishers.
38. M. Pinedo. *Scheduling: Theory, Algorithms and Systems.* Prentice Hall, Englewood Cliffs, 1995.
39. A. C. Renfrew. Dynamic Modeling in Archaeology: What, When, and Where? In S. E. van der Leeuw, editor, *Dynamical Modeling and the Study of Change in Archaelogy.* Edinburgh University Press, Edinburgh, Scotland, 1994.
40. Robert G. Reynolds. An Introduction to Cultural Algorithms. In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, River Edge, New Jersey, 1994.
41. Robert G. Reynolds. Cultural algorithms: Theory and applications. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 367–377. McGraw-Hill, London, UK, 1999.
42. S. Ronald. *Genetic algorithms and permutation-encoded problems: Diversity preservation and a study of multimodality.* PhD thesis, The University of South Australia, 1995.
43. S. Ronald. Robust encodings in genetic algorithms. In D. Dasgupta & Z. Michalewicz, editor, *Evolutionaty Algorithms in Engineering Applications*, pages 30–44. Springer-Verlag, 1997.
44. Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms.* Physica-Verlag, New York, 2002.
45. E. Taillard. Parallel tabu search technique for the jobshop scheduling problem. Technical Report ORWP 89111, Ecole Polytechnique Federale, Lausanne, Switzerland, 1989.

46. P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.
47. Takeshi Yamada and Ryohei Nakano. Job-shop scheduling. In A. M. S. Zalzala and P. J. Fleming, editors, *Genetic Algorithms in Engineering Systems*, IEE Control Engineering Series, chapter 7, pages 134–160. The Institution of Electrical Engineers, 1997.