A Cultural Algorithm with Operator Parameters Control for Solving Timetabling Problems *

Carlos Soza Canales[1], Ricardo Landa Becerra[2]
María Cristina Riff[1], and Carlos Coello Coello[2]

[1] Universidad Federico Santa María
Departamento de Informática
Av. España No. 1680, Valparaíso, Chile
{csoza,mcriff}@inf.utfsm.cl
[2] CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Ingeniería Eléctrica, Sección de Computación
Av. IPN No. 2508 Col. San Pedro Zacatenco, México D.F. 07300, Mexico
{rlanda,ccoello}@cs.cinvestav.mx

Abstract. A cultural algorithm, together with a set of new operators for the timetabling problem, is proposed in this paper. The new operators extract information about the problem during the evolutionary process, and they are combined with some previously proposed operators, in order to improve the performance of the algorithm. The proposed algorithm is tested with a benchmark of 20 instances, and compared with respect to three other algorithms: two evolutionary algorithms and a simulated annealing algorithm which won an international competition on this problem.

1 Introduction

The timetabling problem is a combinatorial problem that can be viewed as an optimzation task. It consists of assigning schedules to several workers or students, which also require some resources. In order to make a feasible timetable, a set of hard constraints must be satisfied (most of them technical constraints); moreover, a good timetable must satisfy some soft constraints (frequently, comfort-related constraints), and if all soft constraints are met, we can consider the solution as optimal. From this point of view, the timetabling problem can be considered as an optimization problem, when trying to minimize the violations of the existing soft constraints.

This NP-hard problem presents several variants, such as the employee, exam and university timetabling problems. In 2002, the Metaheuristics Network organized a competition on the University Course Timetabling Problem (UCTP), and published a set of instances of the problem, in order to make easier the comparisons of different algorithms. However, in that competition no evolutionary

* *Partially supported by CONACYT/CONICYT Collaboration Project México-Chile No. J110.331/2005*

algorithms were proposed, even when evolutionary algorithms have shown to be very effective techniques when solving combinatorial problems.

Cultural algorithms [1] 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 normally known to be useful when solving combinatorial problems. 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.

Several heuristics have been used for different types of timetabling problems: evolutionary algorithms ([2]), memetic algorithms ([3]) tabu search ([4]), simulated annealing ([5]), the ant system ([6]), and hybrid algorithms ([7–9]), among others.

Note however, that this paper presents the first attempt (to the authors' best knowledge) to use cultural algorithms to solve timetabling problems.

The proposed approach is compared with respect to an evolutionary algorithm with specialized crossover operators [2], a recently published memetic algorithm [3], and a simulated annealing approach [5] that won the competition of the Metaheuristics Network, in all the test cases proposed for that competition. The obtained results indicate that the proposed approach is a viable alternative for solving, efficiently, timetabling 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 university course timetabling 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 variant of the problem tackled here was proposed by Ben Paechter for the International Timetabling Competition organized by the Metaheuristics Network [10]. It is referred to in the following as the University Course Timetabling Problem (UCTP). Lecture must be scheduled in 45 timeslots (5 days of 9 hours each) and a number of rooms, with varying facilities and student capacities, so that the following hard constraints are satisfied:

- *H1*: lectures having students in common cannot take place at the same time;

- *H2*: lectures must take place in a room suitable for them in terms of facilities and student capacity; and
- *H3*: no two lectures can take place at the same time in the same room.

We consider as well the following soft constraints:

- *S1*: students should not have to attend lectures in the last timeslot of the day;
- *S2*: they should not attend more than two lectures in a row; and
- *S3*: they should not have a single lecture in any given day.

A timetable in which all lectures have been assigned to a timeslot and a room so that no hard constraints are violated, is said to be feasible. The aim of the problem is to find a feasible solution with minimal soft constraint violations.

2.1 Model

Parameters The parameters of the problem:

- R: Set of Rooms
- E: Set of Events
- F: Set of Features
- S: Set of Students
- T: 45 timeslots (5 days of 9 slots per day).

Variables Let: a, indicate students attendance:

$$a_{(s_i,e_j)} \begin{cases} 1 \text{ if student } s_i \text{ attends event } e_j \\ 0 \text{ otherwise} \end{cases}$$

g, indicate wich rooms are suitable for which events (i.e. the room provides all the features that the event requires and has the right size) e_j:

$$g_{(r_l,e_j)} \begin{cases} 1 \text{ if room } r_i \text{ is suitable for the event } e_j \\ 0 \text{ otherwise} \end{cases}$$

p, indicate the placement of events:

$$p_{(e_j,t_k,r_l)} \begin{cases} 1 \text{ if event } e_j \text{ is placed in timeslot } t_k \text{ and room } r_l \\ 0 \text{ otherwise} \end{cases}$$

Hard Constraints

- Lectures having students in common cannot take place at the same time.

$$H1 : \forall_{s \in S} \forall_{t \in T} \sum_{j=0}^{|E|} \sum_{t=0}^{|R|} a_{(s,e_j)} * p_{(e_j,t,r_l)} \leq 1 \tag{1}$$

– Lectures must take place in a room suitable for them in terms of facilities and student capacity.

$$H2 : \forall_{e \in E} \sum_{k=0}^{|T|} \sum_{t=0}^{|R|} p_{(e,t_k,r_l)} * g_{(r_l,e)} = 1 \tag{2}$$

– No two lectures can take place at the same time in the same room.

$$H3 : \forall_{t \in T} \forall_{r \in R} \sum_{j=0}^{|E|} p_{(e_j,t,r)} \leq 1 \tag{3}$$

Soft Constraints

– Students should not have to attend lectures in the last timeslot of the day.

$$S1 : \sum_{d=0}^{4} \sum_{l=0}^{|R|} \sum_{j=0}^{|E|} \sum_{i=0}^{|S|} p_{(e_j,t_{9d+8},r_l)} * a_{(s,e_j)} \tag{4}$$

– They should not attend more than two lectures in row.

$$S2 : \sum_{d=0}^{4} \sum_{i=0}^{|S|} \sum_{k=9d}^{9d+7} h(), \tag{5}$$

$$h() = \begin{cases} 1 \text{ si } \sum_{n=0}^{2} \sum_{t=0}^{|R} \sum_{j=0}^{|E|} p_{(e_j,t_k,r_l)} * a_{(s_i,e_j)} = 3 \\ 0 \text{ e.o.c.} \end{cases}$$

– They should not have an only lecture in any given day.

$$S3 : \sum_{d=0}^{4} \sum_{i=0}^{|S|} q(), \tag{6}$$

$$q() = \begin{cases} 1 \text{ si } \sum_{k=9d}^{9d+9} \sum_{t=0}^{|R} \sum_{j=0}^{|E|} p_{(e_j,t_k,r_l)} * a_{(s_i,e_j)} = 1 \\ 0 \text{ e.o.c.} \end{cases}$$

Objective Function

$$\textit{MINIMIZE } \{ \textit{ S1 + S2 +S3 } \} \tag{7}$$

The goal of solving the problem formulated in such a way is to minimize the number of soft constraint violations.

3 Cultural Algorithms

Cultural algorithms were developed by Reynolds [1] as a complement to the metaphor used by evolutionary algorithms ([11]), 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 [12]). 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 [13]. 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 information encoded in the ancestors' genes.

Reynolds [1] 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 [14].

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.

Algortihm 1 shows the pseudo-code of a cultural algorithm.

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, 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

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
until the end condition is satisfied
```

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 [15]. The implementation details for these functions in the current proposal are given in the next section.

In [1], it is proposed the use of genetic algorithms [16] to model the micro-evolutionary process, and Version Spaces [17] 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 [18].

In genetic algorithms' theory, there is an expression, called *schema theorem* [19] that represents a bound on the speed at which the best schemata of the population are propagated. Reynolds[1] provided a brief discussion regarding how the belief space could affect the schema theorem. His conclusion was 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 [20, 21]).

4 Proposed Approach

The approach proposed in this paper uses, in its population space, a population based on the evolutionary algorithm originally proposed in [2]. A pseudo-code with the main steps of the proposed cultural algorithm is shown in Algorithm 2.

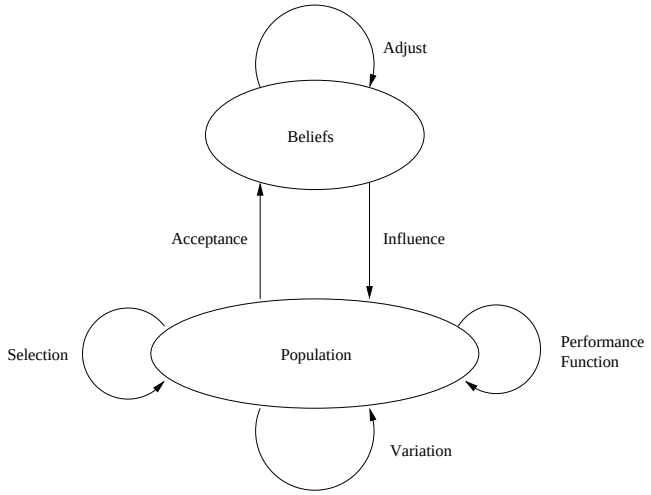


Fig. 1. Spaces of a Cultural Algorithm

In our algorithm, we have considered three types of knowledge: situational, normative and domain knowledge. Also, we are using five variation operators: two of them use the cultural knowledge (cultural mutation and repair) while the other three are designed to add the exploration component of the algorithm (interchange, sequencing and simple mutation). It is worth mentioning that only one of the exploration operators is applied to each individual.

4.1 Representation

The representation adopted to encode the solutions plays a very important role when applying an evolutionary computation technique [22, 23]. In this case, a matrix representation was adopted, where columns represent slots of time, and rows represent rooms for the events.

This encoding was chosen because it can represent any feasible timetable, and is easier to analyze the violation of some hard constraints, considering only one column at a time.

4.2 Exploration Operators

The exploration operators are those that allow to maintain diversity of the population. They are listed next.

The sequencing operator is similar to the one in [5], and its intention is to generate a large change in the individual since it interchanges two timeslots (this operator is the most destructive one used here).

The interchange operator of [2], interchanges two events, and its purpose is to modify the individuals when the problems have in their feasible solutions the same number of places available and events to assign.

The simple mutation operator changes the place of an event, and it is useful when the problems have more places availabe in their feasible solutions (without considering the last periods of the day) than events to assign. The last two operators make use of the matching algorithm [24] to increase their rate of success.

Algorithm 2 Pseudo-code of the cultural algorithm adopted

```
Generate s random schedules (initial population)
Compute the fitness of each individual in the initial population
Initialize the belief space (copying the best individual to the situational belief space
and create the normative matrix)
repeat
  for each individual in current population do
      Apply cultural mutation operator
      switch (operator)
          case Interchange:
              Apply Interchange Operator
          case Sequencing:
              Apply Sequencing Operator
          case SimpleMutation:
              Apply Simple Mutation Operator
      end switch
      Apply repair operator (with domain knowledge)
  end for
  Selection proccess
  Update the belief space (with the individuals accepted)
until the end condition is satisfied
```

Parameter Control for the Application of Exploration Operators The parameter control is a process, concurrent to the search of solutions, that allows values of the parameters to change during this process [25]. We use a mechanism of parameter control in order to select the exploration operator (interchange, sequencing or simple mutation) to apply during the mutation process, using a roulette wheel and based on the success rate of each operator.

This mechanism consists of updating the probability of each operator to be applied, following some simple rules.

If the application of the operator number i results on an improvement of the fitness of the generated individual (with respect to his parent) ($f_{cur} < f_{prev}$), the update of the probabilities is made as follows:

$$operator[i] = operator[i] + \Delta variation$$

where $operator[]$ is the array that contains the probabilities of the operators to be applied, $\Delta variation = \frac{f_{prev}-f_{cur}}{f_{prev}+f_{cur}}$, and $\forall j \in \{1, \ldots, NumOper\}$ and $i \neq j$, $operator[j] = operator[j] - \frac{\Delta variation}{NumOper-1}$, with $NumOper = 3$ in this case, because we have three operators.

This technique to update the operator rates was taken from the techniques to adapt the population size proposed by Eiben [26] and Michalewicz [27].

When an operator i is applied and the present solution gets worse ($f_{cur} > f_{prev}$); the updating of the probabilities is made as follows:

$$operator[i] = operator[i] - \Delta variation * \alpha$$

where $\alpha = \frac{PresentTime}{TotalTime}$, and $\forall j \in \{1, \ldots, NumOper\}$ and $i \neq j$, $operator[j] = operator[j] + \frac{\Delta variation * \alpha}{NumOper-1}$.

The goal of incorporating the α factor is to mantain controlled the level of decrement, with the objective of not dusturbing those operators whose decreasing ranks are much greater, like the sequencing operator.

Initially, the 3 operators in competition start with the same probability of being chosen: $\forall i \in \{1, \ldots, NumOper\}, operator[i] = 1/CantOper$. In order to assure that all operators always have a probability $\neq 0$ of being chosen, all values in $operator[]$ remain between $MinProb = 0.1$ and $MaxProb = 0.8$.

4.3 Mutation Operators with Cultural Influence

The operator begins selecting an event E and a position (r, t) to move it. This is done through different types of cultural influence.

Situational Influence With the situational influence each individual tries to follow a leader. Such a leader is the best individual found, and is stored in the situational belief space. The key idea is that the individual to be mutated becomes more similar to the leader after the mutation process. The mutation operator randomly selects an event E from the leader, and tries to inherit its position (r, t) to the individual.

The situational belief space is updated at each generation. If the best individual of the current generation is better than the leader in the situational belief space, then the leader is replaced by that individual.

Normative Influence This type of influence is more complex. At each generation, the above average individuals are selected. The idea is to influence the individual to be mutated to inherit some of their characteristics. Before describing the procedure, we need the following definitions:

Definition 41 *We define a* ranking of events *as the set of all the events ordered by the number of events with shared students among them. Thus, the event most connected with other events is the first in the ranking.*

Definition 42 *Given a population $P(g)$ of the generation g and the set S_g composed by the best s individuals of the generation g, we define $M = Matrix(event, individual)$, where each element M_{ij} is the timeslot assigned to the event i in the individual j which belongs to S_g.*

The operator proceeds as follows. The room r is fixed. The event is chosen from the ranking of events using a roulette wheel procedure which is biased to the most interconnected events. The new timeslot in the same room r is randomly selected from the matrix M, thus the most common timeslot t of the event E in M has the biggest probability of being selected. The hardest event to be assigned, from the constraints point of view, is the event that shares students with the largest number of events.

The matrix M is updated at each generation g, after the selection of the set S_g (the above average individuals).

Once an event E and the position (r, t) have been selected (by any of the cultural influences mentioned), the process of mutation continues as shown in Algorithm 3. First of all, the operator identifies the current position (r_E, t_E) of the event E in the individual to be mutated. If the new position selected (r, t) is empty and if it is feasible to place E there (from the hard constraints point of view), the current position of event E is modified to (r, t). In case another event E_m is in (r, t), the operator makes swapping moves to change E_m to another position, in order to release (r, t).

Algorithm 3 mutation(E, (r, t)) procedure, which implements mutation after the influence of cultural selection

```
mutation_finished = FALSE
identify the position (r_E, t_E) of the event E in the chromosome
while mutation_finished ≠ TRUE or maxtries < 1000 do
    if the position (r, t) of the chromosome is empty then
        try to move the event E from the position (r_E, t_E) to (r, t), satisfying the hard
        constraints
    else
        try a swapping move of the event E_m in (r, t)
    end if
    if the position of E was changed then
        mutation_finished = TRUE
    end if
end while
```

4.4 Domain Knowledge

Our algorithm makes a post-processing procedure which uses the domain knowledge to modify individuals. In the timetabling problem, it is known that the best solution does not include events in the last timeslots of each day, thus the

purpose of the repair operator is to try to move the events located in the last timeslots to the earliest ones, always satisfying the hard constraints.

5 Comparison of Results

The Cultural Algorithm (CA) is compared with respect to 3 different approaches: a Simulated Annealing (SA) that was the winner of the competition [5], a recent version of a Memetic Algorithm (MA) [3] and the Evolutionary Algorithm (EA) in which this work is based [2]. These references were chosen because they are representative of the state-of-the-art and very competitive on the timetabling problem. The comparison with another EA shows the improvement obtained with the incorporation of culture. The SA approach still presents the best results, but we compare results with it even when it is not an evolutionary algorithm.

The benchmark adopted to make the tests and comparisons are the 20 instances of UCTP from the timetabling competition [10]. Those problems are characterized for being of varied difficulty, they consider the individual satisfaction of the students (which allows to consider them individually, not in classes nor groups), and have at least one solution that fulfills both types of restrictions.

The proposed approach was implemented in the C++ programming language and was compiled using the GNU g++ compiler in the operating system Debian 3.1. Also, the matching algorithm found in the LEDA library [24] was used.

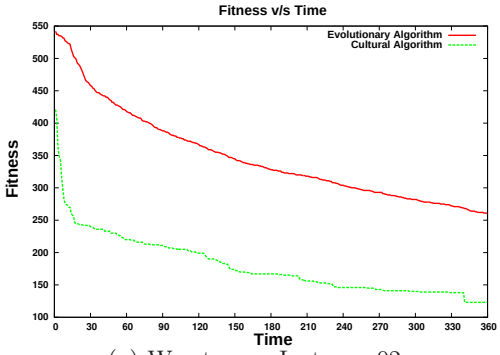
The cultural algorithm was executed 360 s, as was indicated for the bechmarking program of the competition, for our system configuration.

5.1 Cultural Algorithm and Evolutionary Algorithm

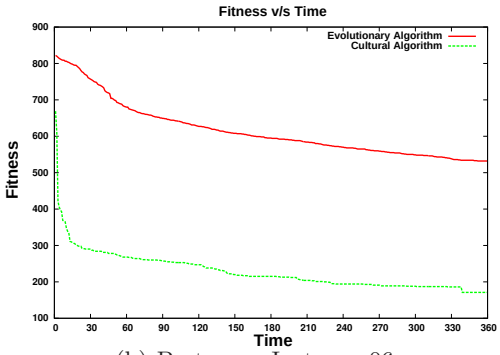
The graphs of Figure 2 show the best and the worst case of improvement of CA with respect to EA, in the 20 instances considered. The worst case (Figure 2(a)) and the best behavior (Figure 2(b)) consider a significant improvement in the first stages which is reflected directly in the final result, in which the cultural algorithm has better results. These graphs show that the incorporation of culture tends to accelerate the convergence of the algorithm and to improve the results.

5.2 Cultural Algorithm and Other Algorithms

Table 1 shows the results obtained by each algorithm in the 20 instances. Table 2 shows a comparison of distances of the cultural algorithm with respect to the other 3 algorithms. Table 3 shows a summary of the obtained results emphasizing that the CA improves all the results of EA. The results of the CA are very close in quality from those of MA. Finally, SA is still the most robust approach to solve timetabling problems.



(a) Worst case: Instance 02



(b) Best case: Instance 06

Fig. 2. Comparison in time

Instance	AE	ACCP	MA	SA
1	288	140	104	45
2	260	123	91	25
3	322	149	126	65
4	679	330	189	115
5	557	306	212	102
6	532	171	90	13
7	430	159	127	44
8	305	133	94	29
9	283	101	78	17
10	311	147	113	61
11	328	120	90	44
12	350	187	138	107
13	420	233	185	78
14	469	267	187	52
15	400	204	120	24
16	302	102	74	22
17	521	311	182	86
18	254	100	75	31
19	550	296	224	44
20	424	159	60	7

Table 1. Comparison of results

Instance	AE - ACCP	MA - ACCP	SA - ACCP
1	148	-36	-95
2	137	-32	-98
3	173	-23	-84
4	349	-141	-215
5	251	-94	-204
6	361	-81	-158
7	271	-32	-115
8	172	-39	-104
9	182	-23	-84
10	164	-34	-86
11	208	-30	-76
12	163	-49	-80
13	187	-48	-155
14	202	-80	-215
15	196	-84	-180
16	200	-28	-80
17	210	-129	-225
18	154	-25	-69
19	254	-72	-252
20	265	-99	-152

Table 2. Improvement of the results of CA compared with the three other approaches

Algorithm	Average	Standard Deviation
AE	399,25	119,46
ACCP	186,9	76,58
MA	127,95	50,72
SA	50,55	32,39

Table 3. Summary of results for all instances

5.3 Adaptation on Operators Application Rate

The incorporation of a mechanism to control the parameters of the cultural algorithm, during the selection of the operator to use, resulted on an improvement on the performance of every instance of the benchmark. The graphs of Figure 3 show two representative instances of UCTP. One of them is the instance number 20 (Figure 3(a)) where 350 events in 400 places are considered; in such a case the simple mutation operator resulted useful because an important factor was the number of free places to assign events. On the other hand, instance number 09 (Figure 3(b)) has less options to schedule an event, while it has 440 events and just 440 places; in this case, the interchange operator was more useful.

6 Conclusions and Future Work

In this paper, we propose the use of domain knowledge, both *a priori* and extracted during the search, to improve the performance of an evolutionary algorithm when solving timetabling problems. The executed experiments provided very encouraging results.

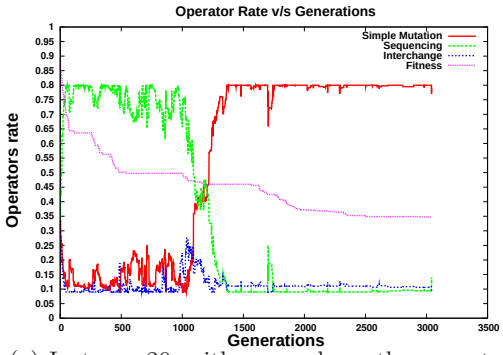
As a future work it would be very interesting to analyze the mechanisms of the simulated annealing method, in order to incorporate them in an evolutionary algorithm or a cultural algorithm. Also, the development of a classification of instances, is a very interesting topic to research, mainly to better understand the performance of different algorithms on different instances.

Acknowledgments

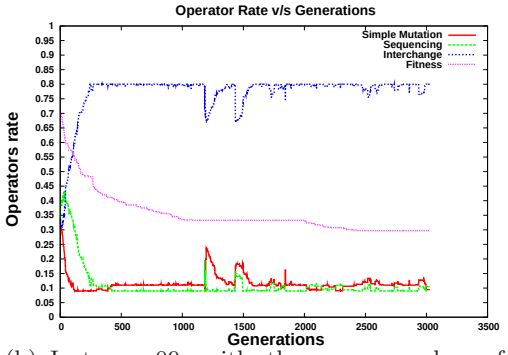
The second author acknowledges support from CONACyT through a scholarship to pursue graduate studies at CINVESTAV-IPN. The fourth author acknowledges support from CONACyT through project number 45683-Y.

References

1. Reynolds, R.G.: An Introduction to Cultural Algorithms. In Sebald, A.V., Fogel, L.J., eds.: Proceedings of the Third Annual Conference on Evolutionary Programming. World Scientific, River Edge, New Jersey (1994) 131–139



(a) Instance 20, with more places than events



(b) Instance 09, with the same number of places and events

Fig. 3. Operators' rate of application

2. Lewis, R., Paechter, B.: New crossover operators for timetabling with evolutionary algorithms. In: 5th International Conference on Recent Advances in Soft Computing (RASC 2004). Volume 5 of Lecture Notes in Computer Science., Nottingham, UK (2004) 189–195
3. Rossi-Doria, O., Paechter, B.: A memetic algorithm for university course timetabling. In: Proceedings of Combinatorial Optimisation (CO'2004), School of Computing, Napier University, Scotland (2004)
4. Cordeau, J.F., Jaumard, B., Morales, R.: Efficient timetabling solution with tabu search. International Timetabling Competition (2003) `http://www.idsia.ch/Files/ttcomp2002/jaumard.pdf`, last access: December 2006.
5. Kostuch, P.: Timetabling competition - sa-based heuristic. International Timetabling Competition (2003) `http://www.idsia.ch/Files/ttcomp2002/kostuch.pdf`, last access: December 2006.
6. Socha, K., Sampels, M., Manfrin, M.: Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In: Proceedings of EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization. Volume 2611 of LNCS., Springer-Verlag (April 2003) 334–345
7. Gaspero, L.D., Schaerf, A.: Timetabling competition ttcomp 2002: Solver description. International Timetabling Competition (2003) `http://www.idsia.ch/Files/ttcomp2002/schaerf.pdf`, last access: December 2006.
8. Chiarandini, M., Socha, K., Birattari, M., Rossi-Doria, O.: International timetabling competition. A hybrid approach. Technical Report AIDA-03-04, Intellectics Group, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany (March 2003)
9. Arntzen, H., Løkketangen, A.: A local search heuristic for a university timetabling problem. International Timetabling Competition (2003) `http://www.idsia.ch/Files/ttcomp2002/arntzen.pdf`, last access: December 2006.
10. Metaheuristics Network: International timetabling competition. `http://www.idsia.ch/Files/ttcomp2002/` (2003) Last access: December 2006.
11. Fogel, D.B.: Evolutionary Computation. Toward a New Philosophy of Machine Intelligence. The Institute of Electrical and Electronic Engineers, New York (1995)
12. Durham, W.H.: Co-evolution: Genes, Culture, and Human Diversity. Stanford University Press, Stanford, California (1994)
13. Richerson, P.J., Boyd, R.: Not By Genes Alone: How Culture Transformed Human Evolution. University Of Chicago Press (December 2004)
14. Franklin, B., Bergerman, M.: Cultural algorithms: Concepts and experiments. In: Proceedings of the 2000 Congress on Evolutionary Computation, Piscataway, New Jersey, IEEE Service Center (2000) 1245–1251
15. Reynolds, R.G.: Cultural algorithms: Theory and applications. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, London, UK (1999) 367–377
16. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Company, Reading, Massachusetts (1989)
17. Mitchell, T.: Version Spaces: An Approach to Concept Learning. PhD thesis, Computer Science Department, Stanford University, Stanford, California (1978)
18. Michalewicz, Z.: A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. In McDonnell, J.R., Reynolds, R.G., Fogel, D.B., eds.: Proceedings of the 4th Annual Conference on Evolutionary Programming. The MIT Press, Cambridge, Massachusetts (1995) 135–155
19. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan (1975)

20. Chung, C.J., Reynolds, R.G.: CAEP: An Evolution-based Tool for Real-Valued Function Optimization using Cultural Algorithms. Journal on Artificial Intelligence Tools **7**(3) (1998) 239–292
21. Coello Coello, C.A., Landa Becerra, R.: Adding Knowledge and Efficient Data Structures to Evolutionary Programming: A Cultural Algorithm for Constrained Optimization. In Langdon, W., et al., eds.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002), San Francisco, California, Morgan Kaufmann Publishers (July 2002) 201–209
22. Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Physica-Verlag, New York (2002)
23. Ronald, S.: Robust encodings in genetic algorithms. In Dasgupta, D., Michalewicz, Z., eds.: Evolutionaty Algorithms in Engineering Applications. Springer-Verlag (1997) 30–44
24. GmbH, A.S.S.: Leda 5.1. `http://www.algorithmic-solutions.com/index.htm` (2006)
25. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter Control in Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation **3**(2) (July 2000) 124–141
26. Eiben, A., Marchiori, E., Valko, V.: Evolutionary Algorithms with on-the-fly Population Size Adjustment. In: Parallel Problem Solving from Nature, PPSN VIII. Volume 3242. (2004) 41–50
27. Arabas, J., Michalewicz, Z., Mulawka, J.: GAVaPS-A Genetic Algorithm with Varying Population Size. In: Proceedings of the 1st IEEE Conference on Evolutionary Computation (ICEC'94). Volume 1., Orlando, Florida, IEEE Service Center (1994) 73–78