

# Evolutionary Scheduling Techniques Applied to University Timetable Using the Java - Xml Technology

Ionel IORGA

Faculty of Mathematics and Computer Science,  
Department of Computer Science,  
University of Craiova, Romania  
ioneliorga@yahoo.com

**Abstract.** In this paper I describe how, using the java technology and an evolutionary algorithm I obtain a solution on a classical problem of artificial intelligence: the timetable scheduling. The evolutionary algorithm was applied in scheduling a university timetable.

**Keywords:** scheduling, timetable, selection, crossover.

**Math. Subjects Classification 2000:** 68T20, 68Q05, 68P05, 68N19, 68M20

## 1 INTRODUCTION

When we want to solve a problem, we are usually looking for some solution which will be the best among others, using some a priori criteria and by choosing the candidate solutions from a search space.

In order to evaluate each possible solution, in the framework given by an evolutionary algorithm, I applied an evaluation mapping (*a fitness function*), obtaining, in this manner, a partial order which I used in the computability process which led to discovery of an optimal solution.

We refer, in this context, to each point from the search space, as an individual from a genetic population and to the cardinal of this population as the number of the points from the search space with the observation that the dimension of a point from the search space is equal to the number of genes of the chromosome of an individual and each such an individual is composed from a single chromosome.

In this manner, through the use of the *fitness function* the degree of the adaptation of each individual to the environment is determined and, consequently, how many specific-conditions imposed by the environment the individual respects.

## 2 THE TIMETABLE PROBLEM CONSTRAINTS

The timetable problem involves scheduling some relations that can be obtained between finite specific resources such as teachers, students, rooms, hours, sub-

jects and equipments with the respect of some *hard* and *soft* constraints. The main differences between the *hard* and *soft* constraints are:

- "hard" - those constraints we cannot violate
- "soft" - constraints over that we can pass sometimes but it is preferable not to do so and every time we violate them this fact is reflected in the value of the *fitness function*.

Some examples of **hard** constraints are:

$HC_0$  - A teacher can only teach in a single place at a time.

$HC_1$  - A teacher can only give one lecture at a time.

$HC_2$  - A room can only host one lecture at a time.

$HC_3$  - A student can only attend one lecture at a time.

$HC_4$  - Room capacities must be respected.

$HC_5$  - No more than a teacher is scheduled to teach in a room each time.

$HC_6$  - Each subject is scheduled in a proper room (for example, a laboratory needs a proper equipment).

$HC_7$  - Every teacher must have scheduled all his hours.

$HC_8$  - Every student must have scheduled all his hours.

We denote the fact that some conditions can be **deduced** from other constraints. For example  $HC_0, HC_2 \rightarrow HC_1$ .

Some examples of **soft** constraints are:

$SC_0$  - A teacher should not teach more than 6 hours a day.

$SC_1$  - A student should not have more than 8 hours a day.

$SC_2$  - There shouldn't be gaps in the activity of the teachers.

$SC_3$  - There shouldn't be gaps in the activity of the students.

$SC_4$  - The courses should be scheduled in the morning and the seminars and laboratories in the after-noon.

$SC_5$  - Some lectures are scheduled *a priori*.

$SC_6$  - As much as possible the preferences of the teachers and the ones of the students should be respected.

$SC_7$  - Launch break should be respected.

All those constraints mentioned must be active in the *fitness function* with the observation that for the hard constraints the penalties are higher than the ones for the soft constraints.

### 3 FORMAL DESCRIPTION

The complexity of the problem has been studied by many authors and a quite large number of techniques were employed to resolve the problem. The most common techniques used linear programming, constraint logic programming,

simulated annealing and graph colouring heuristics. In [3] a combined technique involving logic programming and an evolutionary algorithm was described.

The timetable problem is **NP-hard** and building a timetable using any algorithm involves, in ultimate instance the use of some kind of **finite automata** in order to implement the algorithm.

In the sequel, we give some definitions of such automata and a definition for the NP hard problem as the ones given in [4].

**Definition 1.** A **pushdown automaton** is a system  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  where:

- $Q$  is a finite set of states
- $\Sigma$  is the input alphabet
- $\Gamma$  is the internal alphabet
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \longrightarrow 2^{Q \times \Gamma^*}$  is the transition function
- $q_0 \in Q$  and is called initial state
- $Z_0 \in \Gamma$  and is called initial internal symbol
- $F \subseteq Q$  and its elements are called final states.

where by  $\lambda$  we denote the null word.

**Definition 2.** A **deterministic Turing machine** is a system  $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$  where:

- $Q$  is a finite set of states
- $\Sigma$  is the input alphabet and  $\Sigma \subseteq \Gamma \setminus \{B\}$  where  $B$  we denoted the blanc symbol.
- $\Gamma$  is the internal alphabet
- $\delta : D \mapsto Q \times (\Gamma \setminus \{B\}) \times \{L, R\}$  is the transition function where  $D \subseteq Q \times \Gamma$   
In general, there are pairs in  $Q \times \Gamma$  for which  $\delta$  is not defined;  $\delta$  is a partial mapping.
- $q_0 \in Q$  is called initial state
- $F \subseteq Q$  and its elements are called finite states.

In theoretical computer science, a deterministic Turing machine (**DTM**) has a transition rule that specifies for a given current state of the head and machine  $(s, q)$  a single instruction  $(s', q', d)$ , where  $s'$  is the symbol to be written by the head,  $q'$  is the subsequent state of the machine and  $d$  is the direction (left or right) in which to step.

A **non-deterministic Turing machine (NTM)** differs in that, rather than a single instruction triplet, the transition rule may specify a set of zero or more instructions. At each step of the computation we can imagine that the machine "branches" into many copies, each of which executes one of the possible instructions.

Whereas a DTM has a single "computation path" that it follows, a NTM has a "computation tree". If any branch of the tree halts with an "accept" condition, we say that the NTM accepts the input.

Intuitively it seems that the NTM is more powerful than the DTM, since it can execute many possible computations in parallel, requiring only that one of them succeeds. Any computation carried out by a NTM can be simulated by a DTM, although it may require **significantly longer time**.

**Definition 3.** In computational complexity theory, **NP - non-deterministic polynomial time** is the set of decision problems solvable in polynomial time on a non-deterministic Turing machine.

**Definition 4.** A decision problem is **NP-complete** if it is in NP and every other problem in NP is reducible to it.

**Definition 5.** A problem satisfying the condition that every other problem in NP is reducible to it but not necessarily the condition to be in NP is said to be **NP hard**.

Examples of NP problems: the choice of the perfect move in certain board games or the timetable problem.

## 4 EVOLUTIONARY ALGORITHMS

The considered structure of an **evolutionary algorithm** is the one given in [1]:

```

Evolutionary algorithm
begin
  t ← 0
  initialize P(t)
  evaluate P(t)
  while (not termination-condition) do
    begin
      t ← t+1
      select P(t) from P(t-1)
      alter P(t)
      evaluate P(t)
    end
  end
end

```

The genetic approach is inspired from the evolution concept from the real life. The idea, behind this approach, is to create a population of individuals, in which, any individual represents a timetable. Starting with an initial population the natural evolution towards better individuals is simulated.

At the end of the evolution program the best individual from the final population is selected and, in this manner, we obtain an optimized timetable. On the the individuals the following genetic operators are applied: selection, crossover and mutation. The considered expression of the **fitness function** is the one given by Perzina in [2]:

$$\sum_{i=0}^{s-1} \left( \sum_{j=0}^{t-1} x_{ij} * p_{ij} + \sum_{k=0}^{s-1} c_{ik} * st(i, k) \right)$$

where by

- s we denote the number of subjects, with t the number of time-room slots
- x the matrix in which each line correspond to one teacher and each column to a specified time room slot
- $x_{ij}$  the ij element of the x matrix
- $p_{ij}$  the ij element of the preference matrix (when a teacher prefers to have hours)
- $c_{ij}$  the ij element of a clash matrix
- $st(i, j)$  the *sametime* function which returns true if and only if the activities i and j aren't scheduled in the same time.

The fitness function is a function which should be minimized in order to obtain the best individual.

The number of time room slots is computed by the formula:

- number of time room slots= number of rooms \*number of days \*number of hours per day

The formula of calculus of an **index TimeRoomSlots** knowing idhour, idday, idroom is:

$$TRSlot = idhour + nohours * idday + (nodays * nohours) * idroom$$

For example, the slot in space and time which corresponds Monday, 8 o'clock, room 136 is  $420 = 0 + 12 * 0 + 60 * 7$ .

In order to compute the idhour, idday, idroom knowing the index TimeRoom-Slots we apply the following formulae:

$$idroom = \lfloor \frac{TimeRoomSlots}{nodays * nohours} \rfloor$$

$$TRSlot = TRSlot \bmod (nodays * nohours),$$

$$idday = \lfloor \frac{TRSlot}{nohours} \rfloor,$$

$$idhour = TRSlot \bmod nohours.$$

A **while cycle** cycle, in which the genetic operators are applied, in order to ensure the genetic diversity of the population, is executed until the current generation number is equal with the maximum number of generations.

## 5 JAVA IMPLEMENTATION

Some reasons why the java language was preferred for implementation were: the ease of coding, portability of the code and stability, the ease of deployment and efficiency of integrating and working with heterogenous data sources.

The implementation is structured as follows:

- the chromosomes are implemented as objects of a class which models an individual from the genetic population.

- the class which implements the genetic algorithm contains the *genetic operators* which are used in the process of building the new generations of individuals.
- the program make use of xml technology in order to gather the data needed for the evolutionary computation which led to the solution.

The following xml files are used as *knowledge bases*: RTC.xml, SGSB.xml and TCSB.xml.

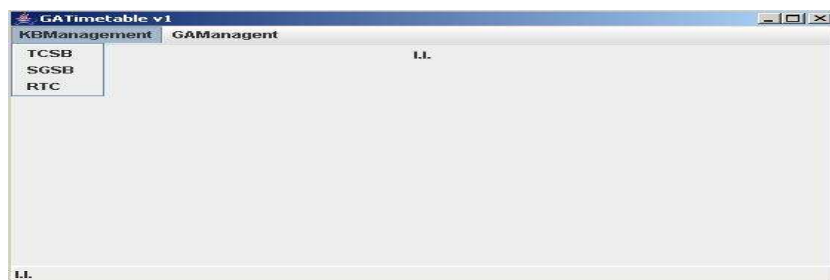
The RTC.xml contains informations about the rooms (the name of the room, the type of the room  $\in \{course, seminary, laboratory\}$  and capacity), the SGSB.xml models the links between subgroups and the subjects. The basic structure of the SGSB file is the name of the subgroup, the name of the subject, the type of the activity  $\in \{course, seminary, laboratory\}$  and the number of hours allocated for the activity.

In a very similar way with the SGSB file, the TCSB file models the links between teachers and the subjects. His basic structure is very similar with the one of the SGSB file. Among the advantages given by the use of xml technology is the standardization, platform independence and good integration with existing programming languages.

## 6 EXPERIMENTAL RESULTS

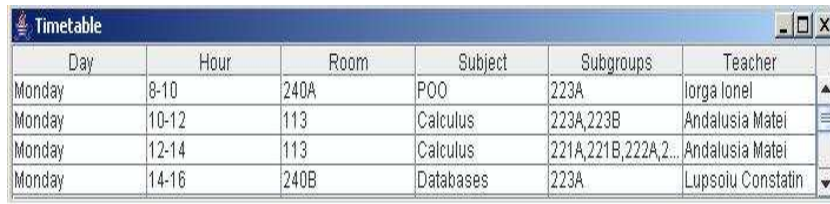
The data set from the Department of Computer Science of the Faculty of Mathematics and Computer Science of the University of Craiova was used for building a timetable.

The main interface of the program is:



The interface provides access to manipulate the xml knowledge bases and to set the working parameters of the genetic algorithm.

After the genetic algorithm provides a solution some views are available on the generated timetable. For example the interface for viewing the timetable for a specified subgroup of students is:



Day	Hour	Room	Subject	Subgroups	Teacher
Monday	8-10	240A	POO	223A	Iorga Ionel
Monday	10-12	113	Calculus	223A,223B	Andalusia Matei
Monday	12-14	113	Calculus	221A,221B,222A,2...	Andalusia Matei
Monday	14-16	240B	Databases	223A	Lupsolui Constantin

## 7 CONCLUSIONS AND FURTHER WORK

For the data set used the described approach has proved her efficiency but, however, further work is needed in order to test the algorithm on other data samples and to increase the convergence speed of the algorithm.

## References

- [1] **Zbigniew Michalewicz**, Genetic Algorithms+Data Structures =Evolution Programs, Springer-Verlag, Second Edition.
- [2] **Radomir Perzina**, A Self-Adapting Genetic Algorithm for Solving the University Timetabling Problem, SCI conference 2004, Orlando, Florida.
- [3] **Iorga Ionel**, Genetic algorithms, logic programming and Java - Prolog connection applied to the university timetable scheduling, AIDC Conference 2005, Craiova, Romania
- [4] **Alexandru Dinca, Mirela Andrei** , Formal Languages and applications, Universitaria Publishing House, 2002, (in Romanian)