Genetic Algorithms in Timetabling

Milena Karova[1], Violeta Bojikova[2], Radoj Stoyanov[3]

Abstract - **In this paper we discuss the implementation of a genetic based algorithm that is used to produce timetables for a small school and a entire University. The problem is a special version of the optimization problems. There are some classical methods, but Genetic algorithms were applied because they are robust and efficient in real life. A problem-specific chromosome representation and knowledge-augmented genetic operators have been developed: these operators 'intelligently' avoid building illegal timetables.**

The construction of timetables is a very difficult problem with a lot of constraints that have to be respected. However there exists a set of entities and constraints, which are common to every possible instantiation of the timetabling problem.

Keywords - **genetic algorithms, timetabling, constraint, time slots, crossover, lesson, class, teacher**

I. INTRODUCTION

The timetabling problem, which has an important role in education, is a special version of the optimization problems found in real life situations. The construction is a very difficult problem with a lot of constraints that have to be respected.

The timetabling problem has always been solved by human resource in small school and for University Course. Almost a week of work of an experienced person is needed to produce a timetable for an average institution and the result is often not satisfactory. When the preconditions change, the whole work becomes unusable, and has to be restarted from scratch. The problem is optimization problem. Existing solutions are either difficult to use or lead to inadequate solutions.

It is difficult to define how good a potential timetable is, but much easier is to declare when a timetable is unusable as it is always exact.

Genetic algorithms have been applied to a very wide range of practical problems, often with valuable results. This paper surveys just a example, to illustrate the diversity of approaches and to point to some of the considerations that have proved important in making applications successful. Because GAs provide a fairly comprehensible way to address a wide range of di cult engineering and optimization problems producing good if not optimal results, it seems that the technology is finding its way into real-world use much more easily than, say, expert systems did.

There are quite a lot of versions of the timetabling problem, differing from one school to the next: Class-Teacher Timetabling, University Course Timetabling, Lecture timetabling et etc

II. THE TIMETABLING PROBLEM

The Timetabling problem comes up every year in educational institutions. Students, teachers, lessons and classrooms have to be arranged optimally. Lecture timetabling is the problem of assigning times and places to a many separate lectures, tutorials, etc , to satisfy several constraints concerning capacities and locations of available rooms, free-time needs and other such considerations for lecturers, and relationships between particular courses. The most prominent overall constraint (central to all timetabling problems) is that there should be no clashes;

The course-timetabling problem essentially involves the assignment of weekly lectures to time periods and lecture rooms.

The class-Teacher Timetabling problem basically considers set of classes and set of teachers.

Timetabling the classes of University considers the resources: rooms, students and lecturers.

Faced with increasing student numbers, with new courses introduced, with shortage of lecture rooms and laboratories and with growing numbers of lessons open for students of different departments, and hence with a large number of conflicting constraints, timetables which can be largely accepted by teachers and students are very difficult to schedule at university.

Each day of week is divided into 13 60-minute periods (time slots), which results in a total of 65 periods numbered from 0 to 64, as can be seen in Fig. 1.

Each lesson must be assigned to a time period in such a way that a number of requirements are met. These requirements can be divided into 2 categories:

A. HARD CONSTRAINTS

A timetable is feasible if all hard constraints are satisfied

[1] Milena Karova is with the the department of Computer Science, Studentska 1, Technical University Varna Email: mkarova@ms.ieee.bg

[2] Violeta Bojikova is with the department of Computer Science, Studentska 1, Technical University Varna Email: bojikov@nat.bg

[3] Radoj Stoyanov is student of the department of Computer Science, Studentska 1, Technical University Varna

Time	D1	D2	D3	D4	D5
7-8	0	13	26	39	52
8-9	1	14	27	40	53
9-10	2	15	28	41	54
10-11	3	16	29	42	55
11-12	4	17	30	43	56
12-13	5	18	31	44	57
13-14	6	19	32	45	58
14-15	7	20	33	46	50
15-16	8	21	34	47	60
16-17	9	22	35	48	61
17-18	10	23	36	49	62
18-19	11	24	37	50	63
19-20	12	25	38	51	64

Figure 1

- Each lesson is scheduled to exactly one period.
- There are no clashes at all: Neither a class nor a teacher nor a room is assigned to more than one lesson in the same period.
- Teacher unavailability is considered.
- All allocated rooms are large enough to hold the students.
- Specific room requirements are taken into account.
- Lessons marked as prescheduled are scheduled to the specified time.
- Lessons are blocked, if so required.

B. SECOND ORDER (OR SOFT) CONSTRAINTS

Good timetables satisfy as many of the following soft constraints as possible, such as:

- Students, as well as some teachers, do not like to have many 'holes' (empty periods between two lessons) in their timetables.
- Lessons should be spread uniformly over the whole week, in general.
- Some teachers, by contrast, wish to have all their lessons scheduled to consecutive periods.
- Some teachers wish to have a special equipped classroom.
- In order to avoid much movement, lecture rooms should be close to the host department.
- Some lessons should not take place late in the evening.
- Each student may select a certain number of optional course modules; in order to give a real choice, conflicts where 2 modules chosen by a student are scheduled at the same period should be avoided.
- Rooms should be just large enough to hold the students.
- As far as possible, classes should either have lessons in the morning or afternoon

III. APPLYING GENETIC ALGORITHMS TO THE TIMETABLING PROBLEM

A timetable can be represented by a fixed set of tuples, each of which contains a class number, a teacher number and a room number. The scheduling algorithm must assign a period number to each of these tuples such that a given class, teacher or room does not appear more than once in a period. In order to use a genetic algorithm to solve this problem, it is necessary to devise a representation for a timetable, which maps onto chromosomes. Each chromosome is composed of genes, each of which represents some property of the individual timetable. Figure 2 (L-Label, C-Class, T-Teacher, R-Room) shows a sample timetable as a collection of tuples. Each period (timeslot 0 to 64) contains a number of tuples, identified by their label. The values of the fields of the tuple would contain valid class, teacher and room numbers.

L 1	L 2	L 3	L 4		L n
C 4 T 5 R 12	C 1 T 2 R 4	C 9 T 5 R 11	C 4 T 9 R 8	●●●●	C 3 T 3 R 5

Figure 2

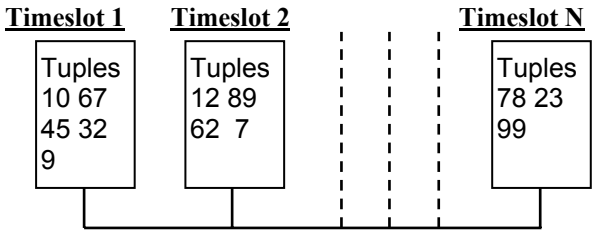

Figure 3

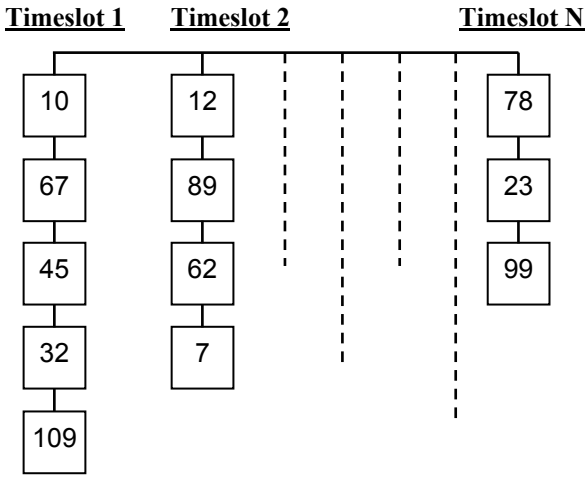

Figure 4

In Figure 3 the data has been mapped onto periods, and shows many tuples packed into each period. The cost of a timetable can then be computed by adding up the number of clashes in each period. The good and bad attributes of a timetable are related to the quality of the packing of

tuples into periods, rather than the absolute period location of any one tuple. Thus, the genetic representation must preserve good packing, remove bad ones and allow new packing to form as the population evolves.

Figure 4 shows one possible mapping of tuples onto chromosomes. In this scheme each period represents a chromosome. Using this mapping allows good packings to be preserved between generations, because corresponding periods are mated, as described shortly. We discarded a number of representations because they did not allow packings to propagate between generations.

Figure 5 shows the process of mating two timetables to produce a new timetable. The diagram only shows the crossover of one period. Each period is crossed over independently. In this example, a random crossover site is chosen, and the new period in the child is constructed with the first 3 genes of parent 1, and the last 2 of parent 2. The same process creates each period of the child. If the crossover site is at either end of the chromosome, then all of that period is taken from only one parent. The fitness of the new child can be computed incrementally as the change in cost of the period.

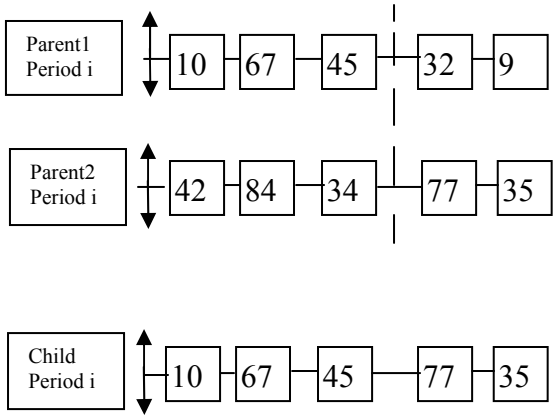

Figure 5

The child can receive further mutation of its mapping. In this example, a randomly chosen tuple from a randomly chosen period is moved to another randomly chosen period.

Genetic algorithms require a measure of the fitness of an individual in order to determine which individuals in the population survive. The higher the fitness, the more likely the individual will survive. The cost measure developed for the timetabling problem represents a high quality solution with a minimal cost value (a value of 0 is optimal). Thus, a fitness value is computed from the cost of a timetable by subtracting the cost from a fitness iling. In this way, an increased cost generates a lower fitness value. Fitness values are further scaled so that unfit individuals have a negative fitness value, and are then removed from the population.

In this example the crossover is simple (one-point). Call the initial individuals Parent1 и Parent2 and select the crossover point randomly and then copy the first part of Parent1 to Child2 and the second part of Parent2 to Child1.

The two-point crossover is a similar operator, at the beginning two crossover points are selected, then the same procedure is used as in the case of the one-point crossover.

For the order crossover (OX) two crossover points are selected randomly. The genetic substrate of Parent2 is copied to Child1, and then the items found in the middle part of Parent1 are removed from Child1. The holes, which come off, are moved to the middle part, among the two crossover points. The ends of the item are connected and start to push the items to the left from the second crossover point until the item which was originally before the second crossover point does not arrive to the position located before the first crossover point. The middle part of Parent1 can be copied to the realized free place at the middle of child1. The same process can generate Child2.

The cycle crossover (CX) does not use crossover points. It copies the first item of Parent1 to Child1, and then it looks for the first item of Parent2 and searches it in Parent1. It copies it to Child1. It looks for the item at the same position in A2, and it searches it again, and so on. It continues the process as long as it does not find an already copied item. By this time, it finished the cycle started from the first item of Parent1. The remained empty fields of Child1 are filled with the items of Parent2, from the same position.

The role of the mutation in the GA is the assurance of heuristic search; it tries to get the individuals found in the up to now, undiscovered part of the problem space.

The basic case of the consistence preserver mutation is the following: choose two points and the length in an individual, then swap the section of the randomized length from the first drawn point with the section of the same length starting from the second drawn point.

It is possible to define cost function for evaluating a given timetable. This function is an arbitrary measure of the quality of the solution. An acceptable timetable has a cost of 0. The optimization problem becomes one of minimizing the value of this cost function. The cost of any period can be expressed as the sum of three components corresponding to a class cost, a teacher cost and a room cost. It is not strictly necessary to sum the components, providing they can be combined to reflect the quality of the solution. However, by using a sum, it is easy to weight the various costs so that one may be made more important than the others. In this way the optimization process can be guided towards a solution in which some types of clash are more important than others.

The class cost is the number of times each of the classes in the period appears in that period, less one if it is greater than zero. If a class appears no times or once in a period then the cost of that class is zero. If it appears many times the class cost for that class is the number of times less one. The class cost for a period is the sum of all class costs. The same computation applies for teachers

and rooms. The cost of the total timetable is the sum of the period costs. Therefore, any optimization technique should aim to find a configuration with the lowest possible cost.
Genetic algorithms require a measure of the fitness of an individual in order to determine which individuals in the population survive. The higher the fitness, the more likely the individual will survive. The cost measure developed for the timetabling problem represents a high quality solution with a minimal cost value (a value of 0 is optimal). Thus, a fitness value is computed from the cost of a timetable by subtracting the cost from a fitness ceiling. In this way, an increased cost generates a lower fitness value. Fitness values are further scaled so that unfit individuals have a negative fitness value, and are then removed from the population.

IV. A GENETIC ALGORITHM

The genetic algorithm can be summarized as follows:

```
While ((number of generations < limit) && (no perfect individual))
  { for (each child in the new population)
    {choose two living parents at random from old population;
     create an empty child;
     for (each period of the parents)
         {mate corresponding periods;
          copy new child period to corresponding position in child;
          }
     repair lost && duplicated labels;
     apply mutation to randomly selected period && tuple;
     measure fitness of individual;
     if fitness < minimum allowed fitness (based on fitness scaling)
         {set child status to born dead}
     else
          {set child status to living}
  }
  old population = new population;
}
```

The algorithm is viewed from the perspective of the child because this allows random selection of parents. The mating is performed with a randomly chosen crossover site within the period, and is done for each period of the individual. Mutation is performed on randomly selected periods and tuples, and occurs with some specified probability.

V. CONCLUSIONS

This paper has proposed a GA based technique for solving the timetable problem and which is capable of:
• Handling many different forms of timetabling constraint while only ever dealing with feasible timetables.
• Generating high-quality solutions despite the increasing intractability, which has resulted from modularization.
• Providing a choice of several different good schedules from which the user may choose the best.
• Directing the timetabler to the most constrained parts of the timetable so that, if necessary, adjustments may be made manually.
• Allowing database queries to produce a schedule for any staff member, student, room or item of equipment.
• Generating a personalized view of the timetable for each member of staff.

VI. REFERENCES

[1] de Werra, D., "An Introduction to Timetabling", European Journal of Operational Research, Vol. 19, pag.151-162.
[2] Hertz A., de Werra D., "Informatique et horaires scolaires", Output, Vol.12, pag.53-56.
[3] Goldberg D. L., "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, 1989
[4] Goldberg D, "Web Courses", http://www.engr.uiuc.edu/OCEE, 2000.
[5] Mitchell M., "An Introduction to Genetic Algorithms", Massachusetts Institute of Technology, 1996.
[6] Rich, D. C. “A Smart Genetic Algorithm for University Timetabling”, In Lecture Notes in Computer Science, Vol.1153, p181-197, Springer.
[7] Paechter B., Rankin R., Cumming A., "Timetabling the Classes of an Entire University with an Evolutionary Algorithm", Napier University, Edinburgh, Scotland.