INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 9 (2002) 703–717

A genetic algorithm for a university weekly courses timetabling problem

Enzhe Yu and Ki-Seok Sung[a]

Department of Industrial Engineering, Seoul National University, [a]*Department of Industrial Engineering, Kangnung National University, Korea*
E-mail: enzhe@snu.ac.kr [Yu]; sung@kangnung.ac.kr [Sung]

Received 10 July 1999; received in revised form 11 February 2002; accepted 4 April 2002

Abstract

The timetabling problem is concerned with the allocation, subject to constraints, of given resources to objects in space and time in such way as to satisfy as nearly as possible a set of desirable objectives. This problem is known to be NP-complete and as such only combinatorial optimization methods can guarantee an optimal timetable. In this paper we propose a sector-based genetic algorithm for solving a university weekly courses timetabling problem. Preliminary experimental results indicate that the algorithm is promising.

Keywords: genetic algorithm (GA), sector-based GA (SB-GA), timetabling problem

1. Introduction

The timetabling problem (TTP) is concerned with the allocation, subject to constraints, of given resources to objects in space and time in such a way as to satisfy as nearly as possible a set of desirable objectives. The TTP is encountered mainly in education institutions and to a lesser extent in business too. The problem is NP-complete, so that solving it satisfactorily often presents great difficulties.

The TTP has been investigated by a number of researchers albeit with respect to the timetabling of resources in educational institutions (Cole, 1964; Wood, 1968). A number of techniques have been developed to solve this type of problem (Carter and Laporte, 1995; Cumming, 1995). The oldest and most popular technique is based on the graph colouring algorithm. However, in practice, this method has one major disadvantage in that it requires the incorporation of non-academic constraints into the problem formulation, which makes it extremely difficult to implement. Other researchers have formulated the problem as an integer program but this too presents a number of difficulties arising from the fact that the number of variables and constraints increase. Yet others have combined constraint logic programming (CLP) with other approaches to solve the TTP. Unfortunately, the performance of CLP-based approaches is very sensitive to even minor changes in formulation. In addition CLP-based

approaches lack flexibility. Recently, there have been a number of 'non-conventional' methods that have been utilized to tackle TTP problems. These include: precise methods, heuristics and a combination of two or more of these techniques. Prominent among these are genetic algorithms (GA), which are rapidly finding application to the solution of the timetabling problem. Genetic algorithms are stochastic search techniques based on the mechanism of natural selection and genetics (Gen and Cheng, 1997; Michalewicz, 1992). A typical genetic algorithm starts with an initial set of random solutions called *populations* and each individual in the population is called a *chromosome*. A chromosome is usually, but not necessarily, a binary string and represents a solution to the problem on hand. Chromosomes evolve through successive iterations, called *generations*. During each generation, the chromosomes are evaluated, using some measures of fitness (Gen and Cheng, 1997; de Werra, 1995). To create the next generation, new chromosomes, called *offspring*, are formed, either by (a) merging two chromosomes from the current generation using a crossover operator, or (b) modifying a chromosome using a mutation operator. A new generation is formed by (a) selecting, according to the fitness values, some of the parents and offspring, and (b) rejecting the rest so that the population size is kept constant. In the process, fitter chromosomes have a higher chance of being selected. After several generations, the algorithm converges to the best chromosome, which hopefully represents the optimum or near optimal solution to the problem.

GAs are becoming popular as a technique for solving optimization problems, mainly because of three distinct advantages they have over their competition (Gen and Cheng, 1997):

1 Genetic algorithms do not involve sophisticated mathematics
2 The operators' ergodicity of evolution makes GAs very effective at performing global search
3 GAs are flexible in that they readily allow for hybridization with domain-dependent heuristics, which can result in a more powerful search routine for a specific problem.

Table 1 describes the general structure of GAs. In the table it is assumed that $P(t)$ and $C(t)$ are the parents and offspring in the current generation.

In this paper, we propose a sector-based genetic algorithm for solving a university's weekly courses

Table 1
General genetic algorithm

```
Procedure: General genetic algorithm
Begin
  t ← 0;
  Initialize P(t):
  Evaluate P(t)
while (not termination condition)
  do
  Recombine P(t) to yield C(t):
  Evaluate C(t);
  Select P(t + 1) from P(t) and/or C(t);
  t ← t + 1;
end
  end
```

timetabling problem. The rest of the paper is organized as follows. In the next section, some definitions are given. This is followed in Section 3 by a description of the proposed timetabling algorithm. In Section 4, results of some experimental test runs are presented. Finally, Section 5 concludes the paper.

2. Definitions

Before a discussion of the algorithm is presented, some definitions are in order. A typical university timetable is characterized by the following elements.

- Event set, $E = \{e_1, e_2, \ldots, e_m\}$ Events are courses offered by the university. These might be formal lectures, experiments, etc. Each event should have its attendees, the lecturer, the time-slots, and the places where the lectures will be conducted.
- Time-slots set, $T = \{t_1, t_2, \ldots, t_n\}$ A time-slot is a time interval during which an event takes place. It has a beginning and finishing time. The elements in the time-slots set are in the form of *Monday1*, *Monday2*, …, *Friday3*, …. Each time-slot has a default value of its duration V_t. For example, in the case studies, the default value was one hour, which was the minimum unit of one class-time.
- Place set, $P = \{p_1, p_2, \ldots, p_i\}$ Events happen in places. In the set, each element contains information relating to name, size and other properties of the place. For example, Place, p_i, might mean a middle-sized lecture room with maximum seating capacity of 60 students. Here 'p_i' is the *name*, lecture-room is the *property*, and '60' refers to the *size*.
- Lecturer set, $L = \{l_1, l_2, \ldots, l_j\}$ In real world cases, a certain course is usually given by the same lecturer l_j.
- Class set, $C = \{c_1, c_2, \ldots, c_k\}$ Students in these classes take courses offered according to their majors and choices. In chromosomes representation, each class contains information relating to *name* and *size*. For example, if class c_k were a group of 30 students, then c_k would be the name of the class, and 30 the number of students in that class which would be directly related to the size of the class and the required place's size.

In addition, the TTP is characterized by constraints. These can be classified into two categories, namely: 'hard' and 'soft' constraints. Hard constraints are those constraints that must be satisfied completely in a schedule. On the other hand, some or all of the soft constraints may be violated provided that the penalty costs associated with them are kept to a minimum.

Typical *hard constraints* are:

1. Each class is scheduled to take place in a specific period.
2. Neither a class nor a teacher nor a room is assigned to more than one lesson in the same time-slot.
3. No omission of classes in the timetable.
4. All allocated rooms are large enough.
5. Specific room requirements must be taken into considerations (i.e. lab).
6. Pre-scheduled lessons must be scheduled to specific time-slot (i.e. liberal classes/rooms).
7. All subjects are evenly distributed.

Typical *soft constraints* are:

1. Neither students nor teachers like timetables with a lot of empty slots (empty periods between lessons).
2. Each kind of event should be spread evenly.
3. Teacher's preferences.
4. Lecture rooms should be close to the host department.
5. Rooms should be just large enough to meet the requirements.

Although these soft constraints do not deal with item conflicts, they play an important role in real-world timetabling. If the total cost resulting from violating the soft constraints can be minimized so as to approach zero, then the generated timetable will be close to the optimal timetable.

3. The genetic algorithm for timetabling

A typical timetable is usually represented in matrix form as shown in Fig. 1. This form of representation is easy to understand and helpful in solving the TTP. However, it cannot be used at the same time to represent chromosomes. This is because it would make the crossover and mutation procedures difficult and confused as chromosomes are traditionally represented as bit strings. As is illustrated in Fig. 1, the necessary elements for a timetable are time-slots, classes, lecturers, places, and subjects. Coordination among classes' timetables and timetables scheduled for places and lecturers is required.

In order to take account of all the elements that are required in the timetable and make the evolutionary process effective, the chromosomes are designed as shown in Fig. 2.

Time-slots

Events take place during workdays. In a practical timetable, such as the one in Fig. 1, the column dimension consists of weekdays, and the row dimension consists of several time-slots within each weekday. In chromosomal design, all the time-slots are put into one dimension – column. For example,

For *Class X.*

	Mon	Tue	…	Thu	Fri
Slot 1	…	…	…	…	…
Slot 2	…	Subject Lecturer Room	…	…	…
…	…	…	…	…	…
Slot 6	…	…	…	…	…

Fig. 1. Real-world timetable example.

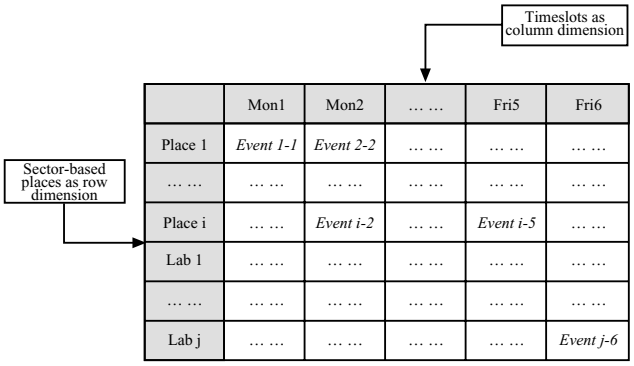

Fig. 2. Chromosome representation for TTP.

if weekdays are from Monday to Friday, and each day consists of six time-slots, then the total number of time-slots per week would be $5^*6 = 30$. These time-slots are shown in Fig. 2 as *Mon1*, *Mon2*, . . ., *Fri6*. Note that this may change according to practical situations.

Places

All the places (lecture rooms, laboratories, studios, etc.) are in row dimension. The concept of 'sector' is introduced here and shall mean a set of places, which have some common properties, such as type of place, size of place, etc. The largest set of sectors in the case of this study are those classified by type, for example, all the experimental rooms make up a lab sector, and all lecture rooms make up a lecture-room sector. The second largest set is the set classified by the size of the places. Here, size can either be large, medium, or small. These sectors are classified according to their type. For example, within the lab sector, an event that is medium-sized will probably be assigned to the medium or large-sized sector of the lab sector, but never to that of the lecture-room sector. This is because they are not of the same type.

In the row dimension, all the places are first classified into the two biggest sectors, that is, lab sector and lecture-room sector. Then, within each type-based sector, the places are classified into sub-sectors according to size. For example, in the lecture-room sector, all large-sized places are assigned to large-sized sectors, while medium-sized places are probably assigned to medium-, or large-sized-sectors etc.

Following this rule, when all the events are assigned to sectors, they are ordered alphabetically. In the vertical order, the lecture-room sector is set at the top followed by the lab sector. The final order of row dimension would be 'from lecture to lab, and within each of these two sectors, sub-sectors follow the order of from small-sized sub-sector to large-sized sub-sector. Similarly, within the size-based sector, places are ordered alphabetically'. Places may be lecture room, laboratory, etc. in TTP. Each place has the following form:

$$\text{Place} \rightarrow \text{name} \times \text{size} \times \text{host} \times \text{type}$$

The places are sorted by type, that is, lecture rooms, laboratories, etc., and within each sorted range, the elements in dimension Plc_k as shown in Fig. 2 are ordered from common size to large size, and

then to special places such as laboratories. These comprise the sub-ranges. A record is made of the range of each kind of place. This is helpful when dealing with conflicts, and deciding the penalties to be imposed on the events. Note that the form described above is not shown in the chromosome. What is shown are the corresponding numbers/codes. In a feasible and acceptable timetable, the relationships shown above should not violate hard constraints and the properties should coincide with each other as nearly as possible.

Event

Each event in the case consists of six elements:

$$\text{Event} \rightarrow \text{subject} \times \text{class} \times \text{lecturer} \times \text{size} \times \text{host} \times \text{type}$$

where *subject* stands for the name of the subject; *lecturer* is the person who gives the lecture; *class* is the name of the group which takes the course; *size* corresponds to the size of the place; *host* is the host department, of the lecturer/subject; and *type* indicates the kind of subject, i.e. lecture, experiment, etc.

The relationship between places and events is shown in Fig. 3. According to the relationship, at least two hard-constraint issues and one soft-constraint issue are involved. For example, the types of places and events should always be the same, and the places' sizes should never be smaller than those of the classes. If possible, an event should be near to the host department. During latter steps, namely: initialization, crossover and mutation, the hard constraints should always be satisfied to guarantee the feasibility of the chromosome.

When the timetable is evolved, the specific individuals' timetables can also be achieved. For example, a place Plc_i's timetable is simply the row it occupies in the timetable. Let α be the set of timetable event E, which contains all the events in row i, then Plc_i's timetable, is α.

$$TT(Plc_i) \rightarrow \alpha \times \textit{time-slots}$$

A lecturer/class/subject timetable can be achieved by searching the key words (lecturer name/class name/subject name) through all events within the generated timetable(s). So far, all the information needed to solve the real-world problem has been included in the chromosomes, i.e. events, time-slots, classes, lecturers, and places.

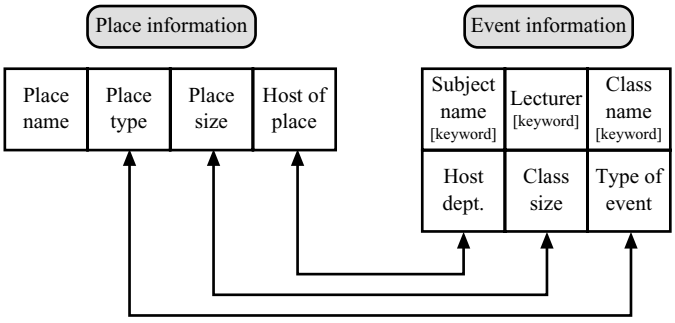

Fig. 3. Relationship between events and places.

Initialization

During initialization, a population of feasible solutions is created randomly without regard to their fitness value. This is followed by a pre-scheduling stage where events with some 'must do' requirements and priorities or hard constraints are scheduled. For example, the case that requires that a chemical experimental class E_i must be arranged in a chemical lab from 9 am to 10 am on Monday. Such events are called 'first priority events'. Some events do not have such definite requirements, but must be satisfied. These events are referred to as 'second priority events'. For example, Professor Kim's computer simulation class must be assigned to computer lab in the morning on any weekday.

The pre-scheduling stage is followed by what is known as the afterward-scheduling stage. During this stage, the rest of the events (those without the 'must do' requirements or priorities) are randomly assigned to the time–place space of the timetable according to their special requirements on the type of places. For instance, the lecture events that require large-sized classrooms are randomly assigned to the corresponding sectors prior to those requiring small-sized ones. Each event has its properties – the type, the host, and the size of the place. Thus, specific experiments must be assigned to specific types of laboratories; a lecture event must be assigned to a classroom whose size is no smaller than that of the event, etc.

In order to reduce the complexity of the initialization process, the concept of sector introduced above, is applied. For example, a lecture event is never randomly assigned to the lab sector because this makes it infeasible. Indeed, any hard-constraint violations during this randomization will make the timetable infeasible. Thus, whenever a random time-place slot is generated for an event, a check, by trial-and-error, is performed using the hard-constraint checking procedure. If a hard-constraint violation is detected, then a new time-place slot is randomly generated. This continues until no hard-constraints are violated. Generally, the cost of searching and reassigning is cheaper than that of correcting wrong timetables. For cases with slack resources, it is not too difficult to get feasible solutions by following the two stages described above. Difficulties arise when resources are tight.

In general, a normal timetabling problem ought to satisfy the following relationship:

$$C_{ij} \geqslant \mathrm{E}_{ij}$$

where C_{ij} is the capacity of all the resources of type i and size j and E_{ij} is the capacity of the necessary

Table 2
Initialization of chromosomes

Procedure Initialization (for each chromosome)
Stage 1. Pre-scheduling (case related) of hard constraint/priority events;
– with hard-constraint checking
Stage 2. Afterward Scheduling
Step 1. For event e_i, recognize the sector it belongs to;
Step 2. Randomly generate a time-place slot for e_i
Step 3. Check hard constraint violations for e_i,
If any hard violation is detected, then go to Step 2.
else, go to Step 4.
Step 4. Assign e_i to the specified slot.

resources of type i and size j for the events. If the value of $\sum(C_{ij} - E_{ij})$ is too small, especially for special resources, finding a feasible solution becomes a big problem.

In the proposed approach, if the trial-and-error fails, then it goes back to the initial state and goes through the two stages repeatedly until it finds a feasible solution. This kind of problem is called '*boring search*'. In the case studied, the value of $\sum(C_{ij} - E_{ij})$ was not too small, and few boring searches were performed.

Evaluation

The evaluation process mainly deals with soft constraints and has a mechanism for assigning penalties whenever these types of constraints are violated. The evaluation function is as follows:

$$\text{Eval.}(f) = 1/(1 + \text{cost}(f))$$

where cost (f) is a sum of weighted penalty value, i.e. $\text{cost}(f) = \sum w_i^* n_i$, $i = 1, 2, \ldots, k$. Here, n_i is the number of a certain kind of constraints within chromosomes, and w_i is the attached weight/penalty. Despite the fact that during initialization, the fitness value might be very low, it is expected that as the evaluation process continues, more and more soft constraints would be satisfied and the cost get lower. Thus, the objective is to minimize the cost and make Eval.(f) as close to one as possible.

Genetic operators

Crossover – Sector-based PMX

In this part, partially mapped crossover (PMX) is applied. And based on PMX, a sector-based PMX (SB-PMX) is proposed. In the chromosome representation and initialization procedure, all the places are first sorted by type, followed by size within each type. The sorting is done before evolution, because either crossover or mutation between different types of events, i.e. lecture and experiment, is meaningless in the whole process and will only lead to unacceptable timetables. Similarly, if events of different types are mixed by crossover or mutation, the repair procedure becomes so complex that it is better to start the process all over again. This is also true when dealing with places of the same type. For example, event E_1 belongs to event set, physics experiment. It should never, if acceptable, be assigned to a chemistry laboratory, say, L_2.

In generic PMX, the main procedures are as follows:

1. Select subsets with corresponding locations.
2. Establish relationships between genes.
3. Exchange genes according to the relationships.
4. Legalize the chromosomes.

SB-PMX is similar to generic PMX, except that the crossover space is divided into sectors according to either types of places and events, or sizes of places and events. Thus, initially, subsets of events are randomly selected by generating them from each parent. The subsets of events derived from the mother timetable M and father timetable F are denoted as S_1 and S_2, respectively. The subsets may fall into one of the following three conditions:

1. All events are lectures.

2. Containing both lecture and experiment events.
3. All events are experiments.

The crossover takes place between two subsets S_1 and S_2, and two children C_1 and C_2 are generated. The whole procedure is described in Table 3 and is illustrated in Fig. 4.

Though the search is initialized to be feasible, after crossover, the children C_1 and C_2 may produce unacceptable timetables.Therefore the hard constraint check and hard constraint repair processes need to be used to guarantee feasibility. Detailed explanations on these two procedures are given below.

Generally, SB-PMX follows the generic PMX relationship as was described by Goldberg *et al.* (1989), or by Gen and Cheng (1997), which has the same effect, except that it deals with those 'blank' genes which have no hereditary information in spite of the fact that they occupy spaces in the chromosome.

According to the basic evolutionary theory, it is believed that genes without hereditary information have no genetic effects over others. When the chromosomes are in forms of strings, a PMX is usually performed as follows: two strings (permutations and their associated alleles) are aligned, and two crossing sites are picked uniformly at random along the strings. These two points define a matching

Table 3
Sector-based PMX procedure

Procedure Sector-based PMX
Step 1 Set subset S_1 and S_2 for crossover according to the randomly generated blocks;
Step 2 Recognize the range of the locations of subsets, and set sectors for crossover;
Step 3 Apply PMX within each sector:
(1) Establish relationship between genes within each sector;
(2) Exchange genes according to the relationship;
(3) Legalize the chromosomes;
Step 4 Two children, C_1 and C_2, are generated.

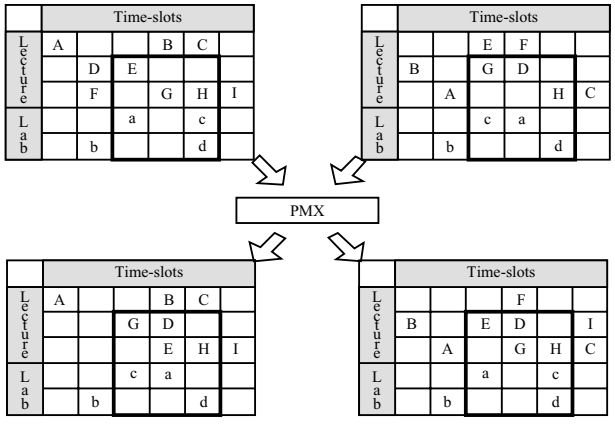

Fig. 4. Sector-based crossover.

section that is used to affect a crossover through position-by-position exchange operation. To see this consider the following two strings:

$$A = 9\,8\,4|5\,6\,7|1\,3\,2\,10$$

$$B = 8\,7\,1|2\,3\,10|9\,5\,4\,6$$

PMX proceeds by position-wise exchanges. Suppose that first string B is mapped into string A. This means that 5 and 2, 3 and 6, and 10 and 7 will exchange places. Similarly, mapping string A into string B, will cause 5 and 2, 6 and 3, and 7 and 10 to exchange places as shown in Fig. 4. And following MPX rules, two offspring, A' and B' are born.

$$A' = 9\,8\,4|2\,3\,10|1\,6\,5\,7$$

$$B' = 8\,10\,1|5\,6\,7|9\,2\,4\,3$$

where each string contains ordering information partially determined by each of its parents.

The rules mentioned above can also be applied to SB-PMX. Although the chromosomes are represented as a two-dimensional matrix, the sub-problem of mapping is almost the same as those described above, except that in the case of SB-PMX: (1) both the chromosomes and matching sections are in matrix form, (2) chromosomes may contain blank genes.

Similarly, consider two chromosomes A and B with blocked mapping sections as shown in Fig 5. When first matching takes place from A to B, it follows the rule in Table 4. For matching from B to A, the rules are the same.

Following the rule, two offspring A' and B' are produced (Fig. 6).

Details of the hard-constraint check and repair routine

According to the proposed chromosome representation and initialization procedure, hard constraints 1, 3, 5, 6, and 7 need not be checked and repaired during the evolution because they are always satisfied. The other two hard constraints, 2 and 4, are discussed below.

Hard constraint 2: neither a class nor a teacher nor a room is assigned to more than one lesson in the same period.

Parent A

A			B	C	
	D	E			
	F		G	H	I

Parent B

		E	F		
B		G	D		
	A			H	C

Fig. 5. Two matrix-form parents to be cross-overed.

Table 4
Partially-mapped crossover

Partially-mapped crossover
(Within each sector set in Step 2, mapping parent A to parent B)

Duplication of genes: $S_A^C = S_A^P$, $S_B^C = S_B^P$;
For each ith gene a_i in M_A^P with location l_i (in matrix-position-wise) in S_A^P **do if** (a_i = 'blank' gene)
  $i = i + 1$;
  else
  Step 1, Find a gene b_k with location l_k that has the same value as a_i in S_B^P.
  Step 2, Swap b_i and b_j with location l_i and l_k in S_B^C, respectively.

** M_A^P, M_B^P denote the mapping section in parent A and parent B.
S_A^P, S_B^P, denotes the specified sector for both parent A and parent B.
S_A^C, S_B^C, denotes the specified sector for both child A and child B.
l_i, l_j denote the location of genes in chromosome.

Child A'

A			B	C	
		G	D		
	F		E	H	I

Child B'

			F		
B		E	D		I
	A		G	H	C

Fig. 6. Two children of A and B after SB-PMX.

[*Analysis and strategy*]: To ensure that this constraint is not violated, you only need to examine whether or not a particular teacher/class is assigned more than once. In the proposed SB-GA, this is accomplished by simply checking vertically the keyword lecturer/class in each time-slot, as shown in Fig. 7.

When hard constraint 2 violations are detected, the chromosome should be repaired to be feasible. One method for doing this is to move the infeasible events (leave one in slot) horizontally, within the size range to which they belong by checking constantly to avoid conflicts, as shown in Fig. 8.

Hard constraint 4: all allocated rooms are large enough.
[*Analysis and strategy*]: Violation of constraint 4 occurs due to crossover between different sizes of the required events, that is, of moving up of larger-sized events to the places that cannot accommodate them. The checking procedure for these is easy in SB-GA. Simply compare the size of the events and that of places. Violation occurs if size(Plc_{ij}) $<$ size(Plc_{ij}).

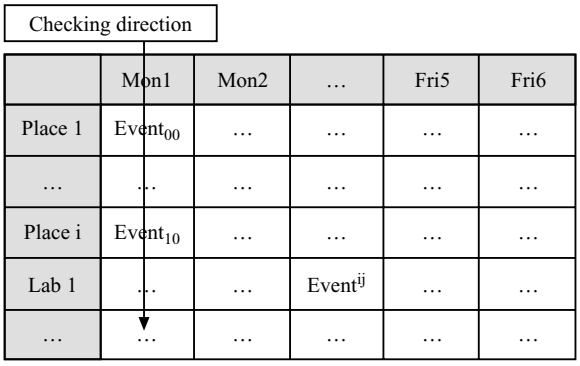

Fig. 7. Hard-constraint checking strategy for hard constraint 2.

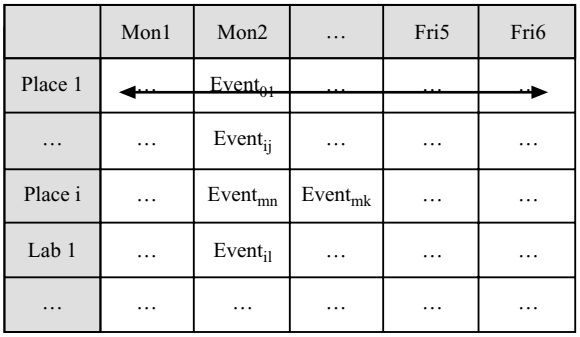

Fig. 8. Repair strategy for hard constraint 2.
*$Event_{01}$ and $Event_{il}$ are conflicted ones;
*Bi-direction arrow means moving direction.

Since this is the result of crossover, or mutation later in the process, that is, due to the exchange of locus between events, the larger-sized events have to be returned to their original area. Similarly, small-sized events that have moved down to larger areas should also be returned. Otherwise, there will be downward trend for events during the whole evolutionary iteration. Here, the exchange relationship between events during the crossover process is applied again. During the repair procedures, the trade-off between these two violations should be considered carefully.

Mutation

Mutations are performed within each sub-range of places as was described above. The process begins by randomly picking genes for mutation at a mutation rate of p_m, within each sub-range. This is the same procedure used by generic mutation, except that in the case of SB-GA, the mutated genes are from the same sector, which can effectively reduce the possibility of making illegal chromosomes.

However, during the mutation process, hard-constraint violations may also occur. Consequently, the hard-constraint check and hard-constraint repair process is again applied here.

Selection

Selection provides the driving force in a generic algorithm, and the selection pressure is critical. Few genetic algorithms can achieve satisfactory results without incorporating an effective and systematic selection procedure. There are two extremes regarding this. At one extreme, the search procedure terminates prematurely, while at the other, progress is painfully slow. In fact, initially, no measures are taken regarding the selection procedure. Therefore in this stage, regarding the three basic issues, i.e. sampling space, sampling mechanism and selection probability, the following measures were taken: as for the sampling space, regular sampling space was chosen; the sampling mechanism follows the method proposed by Goldberg (1989), and a linear scaling approach was applied to deal with the selection probability.

4. Experiments and results

Experiments were carried out on part of a university's real-world data. All the constraints described in this paper were incorporated. The goal was to check whether the algorithm could actually generate acceptable timetables for university weekly courses and to compare cases with different critical GA parameters.

In Fig. 9 results from test runs of 3000 iterations with different population sizes of 25, 50 and 100, and a crossover and mutation rates of 0.6 and 0.07 respectively are shown. As can be observed, within a proper range of population size, the larger the population size, the better the result it got. Among the experiments, a population of 100 got the best performance.

In Fig. 10 results from test runs of 3000 iterations with different crossover rates of 0.4, 0.6, and 0.8 and a population size and a mutation rated of 30 and 0.05 respectively are shown. As expected, the experiment with a lower crossover rate converged towards its goal relatively slowly at the start, while at the end, the three experiments have almost the same convergence speed.

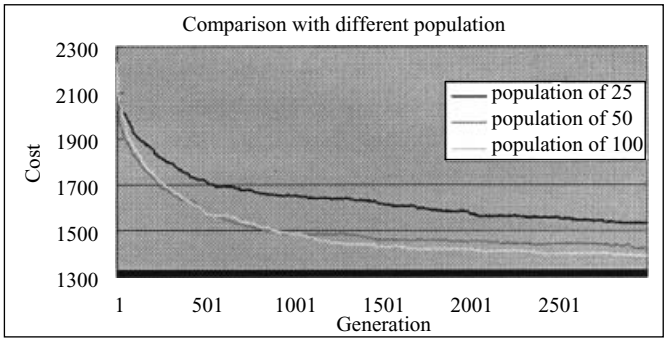

Fig. 9. Comparison of results with different population sizes.

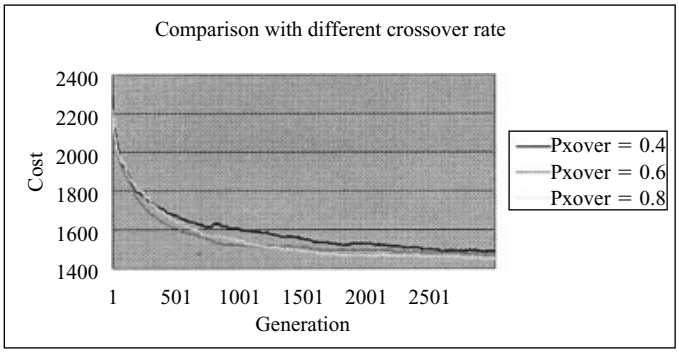

Fig. 10. Comparison of different crossover rates.

Future research is aimed at experiments on the studied university's full data and other universities' data with more constraints and diversity of requirements. We shall improve and extend sector-based operators, find more effective ways of initializing and searching during the evaluation, and investigate the effect of incorporating local optima.

5. Conclusion

In this paper, we propose a sector-based genetic algorithm for a university weekly courses timetabling problem. The concept of 'sector' is introduced and is applied to the initialization, crossover, and mutation procedures. The constraints were divided into soft constraints and hard constraints. In order to keep the solutions in a feasible space, a hard constraint 'check-and-repair' routine was adopted. Experiments were carried out on a university's real data and promising results were achieved.

Further investigation is necessary regarding the following issues: first, test of the performance of the proposed algorithm on some other real data seems necessary; second, in order to test the effectiveness of the proposed algorithm, we should also make a comparison with other algorithms. Third, in the proposed approach, only some of the hard constraints and soft constraints were taken into consideration. In the future study, some more practical hard constraints should be considered and the corresponding 'check and repair' strategies should also be developed.

Acknowledgement

The authors are grateful to the anonymous referee(s) for their insightful comments that enabled us to make a significant improvement. Also, we would like to thank Dr Theo Stewart and Dr Juwa Nyirenda for their invaluable editorial assistance.

References

Carter, M.W., Laporte, G., 1995. Recent developments in practical examination timetabling, *Lecture Notes in Computer Science*, 1153.

Cole, A.J., 1964. The preparation of examination timetables using a small store computer. *Computer Journal*, 7, 117–121.

Cumming, A., 1995. Standard Timetable Data Format. ICPTAT'95 Standard.

Gen, M., Cheng, R., 1997. *Genetic algorithms & Engineering Design*, John Wiley & Sons, Inc.

Goldberg, D.E., 1989. *Genetic Algorithms in search, optimization & Machine Learning*. Addison Wesley Publishing Company.

Michalewicz, 1992. *Genetic Algorithms + Data Structures = Evolution Programs*, second extended edition, Springer-Verlag.

de Werra, D., 1995, Some combinatorial models for course scheduling. *Lecture Notes in Computer Science*, 1153.

Wood, D.C., 1968. A system for computing university examination timetables. *Computer Journal* 11, 41–47.