Memetic Algorithms for Timetabling

Alpay Alkan
aalkan@rams.com.tr

Ender Özcan[*]
Yeditepe University
Department of Computer Engineering,
34755 Kayisdagi - Istanbul/Turkey
eozcan@cse.yeditepe.edu.tr

Abstract- Course timetabling problems are real world constraint optimization problems that are often coped with in educational institutions, such as universities or high schools. In this paper, we present a variety of new operators that can be also applied in evolutionary algorithms for other timetabling problems, such as, exam timetabling. Operators include violation directed mutations, crossovers, and a successful violation directed hierarchical hill climbing method. Tests are performed on a small portion of a real data and results are promising.

1 Introduction

Timetabling problems are NP hard problems (Even 1976). University course timetabling problems are a subclass of course timetabling problems that require feasible assignment of each offered course to a slot of a discrete timetable and some other resources such as classrooms, satisfying a set of constraints. Many different approaches, including Evolutionary Algorithms (EAs), Tabu Search, Simulated Annealing, and their hybrids are developed for solving many different types of timetabling problems (Abramson 1991-Erben *et. al.* 1995, Herz 1992, Monfroglio 1988-Paechter *et. al.* 1998, Ross *et. al.* 1994, Schaerf 1996, Werra 1985). Due to the lack of interest in a common standard on specifying a timetabling problem instance, benchmarking is almost impossible. Some researchers, such as, Ross *et. al.*, produced syntactic data for their experiments (1994). Early studies in standard data format for timetabling yield a language named SSTL (Kingston 2001). These syntactic benchmarks and the use of SSTL has not been spawned further as expected, possibly because, most of the studies in the area initiated due to some practical need, making the researchers to prefer and test the real data that they have. XML based data format for representing timetabling problem instances, named TTML is under development, requiring attention to allow researchers, furthermore applications, such as web services to exchange data (Ozcan 2002). In our experiments we used a portion of real data from our earlier studies, named YU_FEA_2002F (Ozcan *et. al.* 2002).

Our previous experiments in course timetabling using Genetic Algorithms (GAs) illustrated that the individuals tend to become similar, causing premature convergence unavoidable. There are a variety of approaches to solve this problem and maintain population diversity. Crowding (De Jong 1975), using heuristics during initial population generation, hypermutation (Cobb 1990), using a diploid representation (Smith 1992) and dividing the population explicitly into subpopulations like in parallel GAs (Petey *et. al.* 1987) are some of them.

In this paper, we introduce a set of new operators for successful timetabling using memetic algorithms (MAs). In order to maintain the population diversity, several techniques mentioned above are also tested. In section 2, university course timetabling, including the common constraint types, is introduced. In section 3, components of the memetic algorithm for timetabling problems are described. In section 4, a tool, named as TEDI (Time Tabling Tool for Educational Institutions) utilizing EA Algorithms will be introduced and in section 5, experimental results follow. Finally, in section 6 conclusions are presented.

2 University Course Timetabling Problem

University course timetabling problems (UCTPs) are constraint optimization problems that can be represented by a 3-tuple (V, D, C). V is a finite set of course meetings in a department, faculty or university, $V=\{v_1, v_2, \ldots, v_N\}$, $D=\{d_1, \ldots, d_i, \ldots, d_N\}$, is a finite set of domains of variables, for example, let $G=\{t_1, t_2, \ldots, t_M\}$ represent a set of start times for a course meetings, then a possible domain of each variable can be $d_i \subseteq G$ and C is a set of constraints to be satisfied, $C=\{c_1, c_2, \ldots, c_L\}$. Domain of a variable can be a product of sets, each representing a different resource. For example, $d_i \subseteq G\text{x}S$ can be a domain of a variable, where S represents the set of classrooms. In this paper, resources other than time will be ignored. UCTP can be described as a search for finding the best *assignment* (v_i, t_j) for each variable $v_i \in V$, such that, all the constraints are satisfied. The assignment implies that the course meeting of v_i starts at t_j if there are other resources they are allocated for v_i and starting from that time. Note that the size of the search space is immense, N^M.

2.1 Types of Constraints

In general six different constraint types can be identified for UCTPs: *exclusions*, *presets*, *edge constraints*, *ordering constraints*, *event-spread constraints* and *attribute constraints*. Exclusions represent the excluded members of resources for the variables. For example, "Data Structures should not be scheduled on Tuesdays",

[*] Corresponding author

or "C Programming should not be scheduled in the afternoons". Presets represent the predetermined assignments for some variables. For example, "Digital Electronics is scheduled on Fridays at 14:00-17:00". Edge constraints represent a pair of course meetings that should be scheduled without a clash. Assuming a single timeslot assignment for each course meeting, an edge constraint might require a pair of course meetings v_i and v_k to be assigned to (v_i, t_j) and (v_k, t_l), respectively, such that, $t_j \neq t_l$. This is the most commonly used constraint. Discarding the rest of the constraints, if only the edge constraints are supported, then timetabling problem reduces to graph coloring problem (Leighton 1979). Ordering, also known as juxtaposition constraints represent an ordering between course meetings. For example, "File Structures and Database Systems should start at the same time", or "Problem solving session of Theory of Algorithms should be scheduled an hour after the course meeting". Event-spread constraints deal with the way how the course meetings are spread out in time. For example, "All semester courses should be distributed evenly in the weekly schedule". Attribute constraints represent restrictions that apply between the attributes of a course meeting and/or the attributes of its assignment. For example, assuming an attribute for a course is the total number of students taking the course, and an attribute for a classroom is its capacity, a possible constraint would be "Total number of students taking a course should not exceed the capacity of the classroom". Attribute constraints are ignored, since only T is assumed as a domain of a variable.

3 A Memetic Algorithm for Solving UCTPs

Genetic Algorithms (GAs) were introduced by J. Holland (Holland 1975), and have been used to solve many difficult problems (Goldberg 1989). Usefulness of hill climbing and local search operators in population based algorithms is emphasized by many researchers such as Moscato and Raddcliffe *et.al.* (1992, 1994). *Memetic Algorithms* (MAs) are used for solving timetabling problems by combining GAs and local search techniques.

Ross et al. (1994) introduces a set of violation directed mutation operators based on selecting a gene to mutate and an allele to mutate to, for GA approaches. Their tests show that random selection of a gene, then selecting the allele by using a tournament selection performs the best. These operators are not tested in MAs. Furthermore, we introduce a totally different set of violation directed mutation operators as well as a similar set of operators in this paper. Burke et al. (1996) applies a light or a heavy mutation, randomly selecting one, followed by a hill climbing method. Our hill climbing operator is a new one.

Source of the data, representing the variables, or the number of rows and columns in a discrete timetable does not change the nature of the problem. Hence, although a university course timetabling data is used in the experiments, the operators explained below can be applied in evolutionary algorithms for similar timetabling problems, such as high school timetabling, or exam timetabling. Crossover and mutation operators are referenced using notation OP#*id*, where OP indicates the operator and #*id* is its unique id.

3.1 Representation

The direct representation is used, consisting of course section meetings as a variable set, that are grouped with respect to courses, terms and then departments, hierarchically (Figure 1). Each gene denotes the start time of meetings of a course section. A course might require more than one meeting.

3.2 Constraints

Edge-constraints include:

- No course section meetings should overlap in a term, denoted as TED
- No course section meetings should overlap belonging to an instructor, denoted as IED

Event-spread constraints include:

- Different meetings of a course section should be scheduled on different days, denoted as CES.
- Courses in a term should be scheduled consecutively as a block with at most one hour break between them, excluding the lunch break, denoted as TES.
- Courses of an instructor should be scheduled consecutively as a block with at most one hour split between them, excluding the lunch break, denoted as IES.

As an exclude, 5th time slot in each day should be allocated as a lunch break.

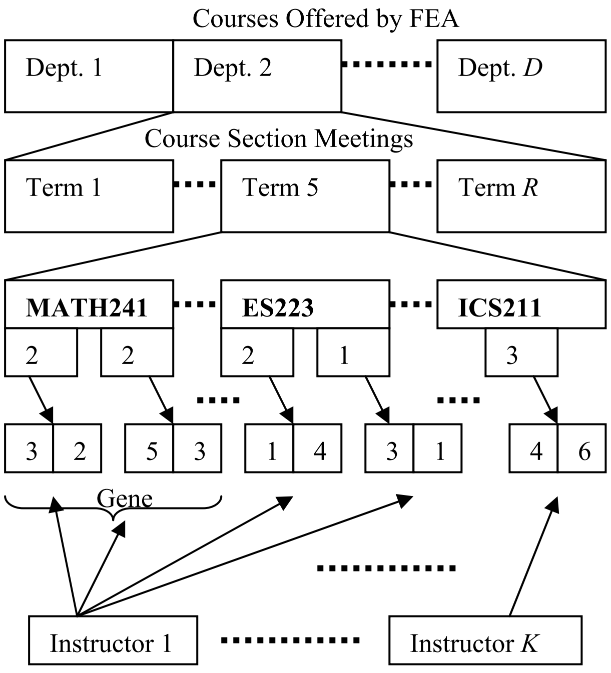

Figure 1. Individual representation

Some of the course section meeting schedules is predetermined. PRE denotes the preset constraints. Instructor requests about their schedules are also used as constraints, identifying some favored and undesirable timeslots, representing exclusions, denoted as EXC. Timetable consists of 9 consecutive 1-hour time slots per day, having 10 minutes allocated as a break between courses, starting at 9:00am for 5 days. Fifth time slot should be arranged as a lunch break in all the term schedules, if possible.

The definition of the constraint set can be extended to include the different types of constraints to be satisfied. Assume that the constraint set C has K different types of constraints, and let β_j denote the set of jth type of constraint, then

$$C = \bigcup_{1 \le j \le K} \beta_j \qquad \text{(Eq. 1)}$$

3.3 Fitness Function

An optimum timetable is the one satisfying all the constraints. Let p_j represent the penalty associated with the constraint $i \in C$ belonging to the constraint type j and $g_i(T)$ represent the number violations in the timetable T due to the constraint i and w_j represent a weight applied to p_j:

$$f(T) = \sum_{\forall j \forall i \{i \in \beta_j\}} w_j p_j g_i(T) \qquad \text{(Eq. 2)}$$

Penalty values of the constraints belonging to the same type are also the same. MA attempts to satisfy constraints with higher penalties with respect to lower ones and/or constraints that cause more violations. Unless it is mentioned w_j is set to 1.

3.4 Initialization and Allele Assignment

Domains of each course section are generated using PRE and EXC constraints. Considering this set and CES constraints, an allele is produced randomly for each gene. Hence, all the individuals produced during the evolution satisfy PRE, EXC and CES constraints.

After a random initialization, population is passed through a hill climbing method. An additional method is implemented to provide some type of guidance for the search, and penalty values are adjusted by a factor of w_j:

$$w_j = \forall y \in population\{\frac{\sum_{\forall i \in \beta_j} g_i(y)}{\sum f(y)}\} \qquad \text{(Eq. 3)}$$

This set of weights is calculated using the initial population to adjust the penalty values for once.

3.5 Mutation

Traditional mutation perturbs a gene, by using a random allele assignment as explained in the above section with a low mutation probability (MUT4). Additional mutation operators are also tested that can be considered as violation directed. MUT2 selects a term, based on ranking using the number of violations for each term, then applies MUT4 operator, as if the chromosome is the term block chosen. MUT3 operates in the same way as MUT2 with a minor difference that is the mutation is forced to take place for once. Another mutation operator selects a term and then selects a gene to mutate and randomly perturbs the allele using ranking during each selection, denoted as MUT1.

3.6 Crossover Operators

Different crossovers are used in GA based on traditional one point crossover (1PTX6) and uniform crossover (UX4). No clash constraints are applied within a block of genes that is terms and instructors. New crossover operators are developed working at the block levels. 1PTX6 and UX4 operators are extended to work as if each term is a gene (1PTX7, UX5). Another set of less disruptive crossover operators are created to work in two steps:

- Select a term, based on a strategy
- Apply crossover only inside that term

The strategy is chosen to be ranking, giving higher chance for a term with a worse contribution to the overall fitness to be imposed to a crossover.

1PTX6 and UX4 are combined with a ranking strategy to produce offspring. Selected parents' violations are added up on term basis and using this information ranking is applied to select a term for crossover, favoring a term with a worse fitness contribution. Then, one of the operators; 1PTX6 and UX4, is applied on the term selected as if it is the chromosome. New operators are named as 1PTX3, UX1, respectively. Another operator selects a term on two individuals using ranking strategy and replaces the whole term, denoted as UX2. Note that any other selection strategy can be applied to select a term for crossover other than ranking, such as tournament selection. The most successful choice for the selection strategy is beyond the scope of this paper. Figure 2 illustrates the operation of 1PTX7 and 1PTX3 crossover operators. Note that any grouping defined over course section meetings can be chosen as a block for a crossover. For example, another set of crossover operators can be generated, that might be applied as if the instructor blocks are genes using the above ideas.

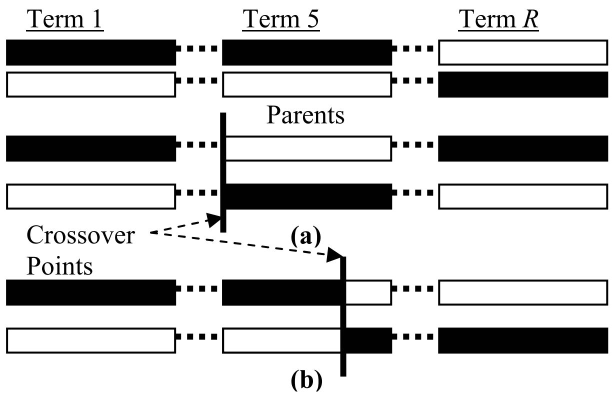

Figure 2. Offspring generated by applying **(a)** 1PTX7 on parents at a random term, **(b)** 1PTX3 on a term selected randomly by ranking .

3.7 Violation Directed Hierarchical Hill Climbing

Timetabling problems are multi-criteria optimization problems. While reducing the violations due to a constraint, overall fitness of an individual can be worsen. Remembering that hill climbing should move an individual to a local optimum, requiring computation of fitness at each step, applying a hill climbing seems to be computationally expensive in timetabling problems. Yet, if the total computation time reduces and/or the quality of solution increases, a hill climbing approach can be preferable. The idea behind our hill climbing approach is to create a hill climbing method for each type of constraint and combine them under a single hill climbing method, denoted as AHC. Starting from a high resolution, as long as the fitness improves, AHC stays at that level, otherwise, AHC lowers the resolution, restricting the aim. There are 3 hierarchical levels of resolution, consisting of 3 improvement strategies:

- Select a constraint type based hill climbing method, using a selection method, giving a higher chance to an operator of the related constraint type causing more violations.
    1. Invoke the selected operator to get rid of *all* the violations arouse due to the related type of constraints, producing a new individual.
    2. If this attempt does not make any improvement on the old one, ignore the new individual. Depending on the constraint type, a selected *block* of genes, possibly causing more violations among the other blocks, are attempted to be corrected.
    3. If this attempt also fails to produce a better individual, then using the old one, a selected single gene in a block of genes, possibly causing more violations, is attempted to be corrected.
- If the fitness of an individual improves in any case, AHC is reapplied on this individual.

Four constraint based hill climbing methods are developed for each constraint type; TED_HC, IED_HC, TES_HC and IES_HC. In a single step of TED_HC and IED_HC, overlapping course section meetings due to the violation of the constraint type in question, in the timetable produced by the individual are considered and a randomly selected meeting is rescheduled to a random empty time slot. In a single step of TES_HC and IES_HC, course meeting assignments in each single day in the timetable view of the constraint type in question, produced by the individual are pushed from morning towards evening and from evening towards morning, if there are empty slots in between the course meetings, where lunch break is an attractor slot.

3.8 Replacement

As a replacement strategy, an elitist operator is used, allowing the worst of the population to be replaced by the best of parents and offspring (EREP). Replacement is not allowed if the individuals to be replaced have duplicates in the population.

Crowding is applied to allow dissimilar but fit individuals to be replaced in the population by limiting the number of alleles that are same in two individuals. Let s*imilarity rate* denote the ratio of the number of same alleles at the same loci and the length of a chromosome.

Steady state (SS) approach requires two offspring to be produced, where as trans-generational (TG) approach requires creation of an offspring pool and replacement occurs among the old generation and the offspring. Both of these approaches are implemented in MA (SSMA, TGMA) for timetabling.

4 TEDI

Experiments are done using TEDI, which is a tool for generating optimum timetables for educational institutions using Evolutionary Algorithms. Figure 3 shows the workflow diagram of TEDI. All GA parameters and constraints are entered into the system using the corresponding GUI. Then TEDI is invoked for finding the optimum timetable with the given input. As a default Genetic Algorithm spawns to find a solution. Hill climbing can be enabled to start a Memetic Algorithm. TEDI extracts constraints, algorithm parameters and curriculum courses offered by each department from the database and start its execution using the selected algorithm.

TEDI allows users to enter the following parameters: the size of the population, initialization scheme, type and related parameters of mutation, crossover, selection and termination method, and penalty for an unsatisfied constraint.

As an output TEDI generates two types of timetable views: Term View and Instructor View. By selecting a department's term or an instructor, related timetable can be visualized.

TEDI keeps all the information entered and the all the results generated in its database. This tool is intended to be developed further to enable web access using CGI.

5 Experiments

All the tests are performed using a part of a real data, obtained from Yeditepe University (YU), Faculty of Engineering and Architecture (FEA), during the registrations in 2002/2003 Fall semester, denoted as YU_FEA_2002F_p. Initial experiments are performed for finding the best set of MA operators. First, best mutation and crossover operators are identified, and then experiments are performed for finding the best replacement strategy.

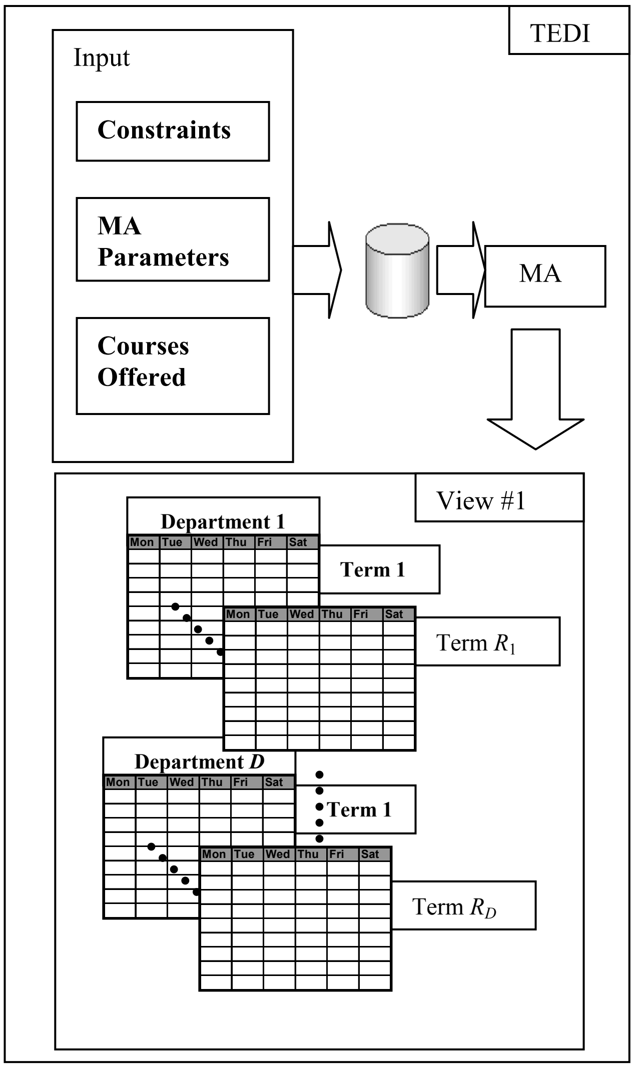

Figure 3. Workflow diagram of TEDI

5.1 Experimental Data

YU_FEA_2002F_p consists of two departments (ICS, EE) in FEA, where there are 151 offered courses, summing up to 233 course meetings to be assigned to a timetable slot in 9 ICS terms and 8 EE terms.

Define ρ, *average occupancy* for τ, as follows:

$$\rho(\tau) = \frac{\eta}{|\tau|} \qquad \text{(Eq. 4)},$$

where η is the total number of course meeting hours, τ is the set of terms or instructors. ρ is used as a measure of how many slots of the timetable will be filled for each term and instructor, choosing τ as a set of terms and instructors, respectively. YU_FEA_2002F_p has a $\rho(Terms)$=24 and $\rho(Instructors)$=9. The total number of constraints sums up to, approximately, 3,000.

5.2 Timetable

A discrete timetable is used as a domain for variables; course meetings. Each day of the week is divided into nine, fifty minute time slots as shown in Figure 4. There will be ten minute breaks in between the lectures.

	Mon	**Tue**	**Wed**	**Thu**	**Fri**
09:00-09:50	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)
10:00-10:50	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)
11:00-11:50	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)
12:00-12:50	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)
13:00-13:50	*(1,5)*	*(2,5)*	*(3,5)*	*(4,5)*	*(5,5)*
14:00-14:50	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)
15:00-15:50	(1,7)	(2,7)	(3,7)	(4,7)	(5,7)
16:00-16:50	(1,8)	(2,8)	(3,8)	(4,8)	(5,8)
17:00-17:50	(1,9)	(2,9)	(3,9)	(4,9)	(5,9)

Figure 4. Division of a week into time slots

5.3 Results

Each experiment is performed for 50 runs using random initialization. Runs are terminated whenever the expected best fitness of 0 is achieved (*best result*), or the number of generations exceeds 10,000. Notice that none of the constraints are identified as hard or soft, since the aim is to minimize the number of violations and the number of evaluations.

In the tables, summarizing the experimental results, α is the number of the best runs out of 50 in which 0 fitness value is reached, γ represents the average number of generations averaged over successful runs, ϕ represents the average number of violations in the final generations of all runs and σ is the standard deviation of related quantities. (X,M) denotes the operator id of a crossover and a mutation pair, respectively.

Table 1 summarizes the experimental results obtained applying EREP as a replacement strategy using steady state memetic algorithm (SSMA) and different crossover and mutation operator pairs.

The best crossover and mutation pair turns out to be (UX4, MUT2) and (UX5, MUT2) for the Memetic Algorithm. When the average number of generations is compared for the runs having best results, (UX5, MUT2) seems to be the best choice. The top seven pairs (TOP7), marked as bold in Table 1, having a success over the mean are tested again without using the hill climbing method. In none of the runs, a solution is found and the resulting population had approximately 38 violations on the average. Doubling the maximum number of generations for TOP7 did not help much, though (UX4, MUT2) turned out to be the best pair having 33 best results out of 50.

Further experiments are performed to see the effect of several approaches used for maintaining diversified populations. TOP7 list is also tested using crowding for various similarity rates, and crossover rates, yielding no improvement. Using weights during initialization also did not help much, but an adaptive approach could be investigated by recalculating the weights during generational translations, but it is left for further study due to the high cost of computation for the time being.

All of the above experiments are performed using a SSMA. In order to compare it with a trans-generational memetic algorithm (TGMA), TOP7 experiments are repeated.

(*X*,*M*)	α	(γ	σ)	(ϕ	σ)
(1,1)	7	5,780	1,929	4.80	3.36
(1,2)	12	7,003	1,960	2.50	2.16
(1,3)	12	5,444	1,779	5.06	4.61
(1,4)	9	7,602	1,519	2.96	2.48
(2,1)	11	7,107	-	4.68	-
(2,2)	8	6,685	-	3.32	-
(2,3)	13	6,641	-	4.04	-
(2,4)	7	4,834	-	2.92	-
(3,1)	8	5,871	-	5.40	-
(3,2)	13	6,814	-	1.98	-
(3,3)	13	5,750	-	5.98	-
(3,4)	9	6,053	-	2.60	-
(4,1)	**21**	**6,829**	-	**2.82**	-
(4,2)	**25**	**7,436**	-	**1.04**	-
(4,3)	**22**	**7,215**	-	**2.34**	-
(4,4)	**22**	**6,622**	-	**1.14**	-
(5,1)	16	4,705	-	3.00	-
(5,2)	**25**	**4,522**	-	**1.12**	-
(5,3)	14	4,095	-	3.06	-
(5,4)	**18**	**4,618**	**-**	**1.24**	-
(6,1)	11	5,366	-	3.70	-
(6,2)	17	5,479	-	2.14	-
(6,3)	14	5,400	-	3.60	-
(6,4)	15	5,586	-	2.24	-
(7,1)	14	5,020	-	4.26	-
(7,2)	**19**	**6,098**	-	**1.74**	-
(7,3)	15	4,570	-	3.59	-
(7,4)	17	4,769	-	1.86	-

Table 1. Test results obtained after running SSMA using different crossover (*X*) and mutation (*M*) operators.

Trans-generational memetic algorithm yields a better result supporting the success of (UX4, MUT2) as shown in Table 2, possibly because the number of evaluations allowed are more than SSMA. Further tests should be performed for a better comparison.

(*X*,*M*)	α	(γ	σ)	(ϕ	σ)
(4,1)	38	1,190	-	1.22	-
(4,2)	41	946	-	0.24	-
(4,3)	37	1,530	-	1.03	-
(4,4)	39	1,441	-	0.40	-
(5,2)	39	1,477	-	0.32	-
(5,4)	39	1,538	-	0.36	-
(7,2)	37	2,387	-	0.38	-

Table 2. Test results obtained after running TGMA using different crossover (*X*) and mutation (*M*) operators.

6 Conclusions

There are many researchers studying a variety of real world timetabling problems. Evolutionary Algorithms are very common approaches for tackling such problems. Varieties of violation directed and block oriented mutation and crossover operators to be utilized in Genetic Algorithms and a new violation directed hierarchical hill climbing operator to be utilized in Memetic Algorithms for timetabling are presented. Experimental results confirm that the best crossover operator is the traditional uniform crossover operator and the best mutation operator is the violation directed operator that is applied onto a block rather than the whole individual. The rest of the crossover and mutation operators are also comparable with the best. These operators are promising and will be tested further using different sets of data.

Experiments carried out using TEDI demonstrated that genetic search combined with hill climbing achieves the best performance. Using a hierarchy of resolution levels, provides means to correct conflicts once and for all, or for a group of events or for a single event, considering a randomly selected constraint type based on ranking the number of violations that arouse due to the constraint types. In our experiments, trans-generational MA yields better results than the steady state MA. The best method to keep a diversified population seems to be disallowing the same individuals to join the population during the replacement.

As a test bed, a real world data defined for a university course timetabling problem is used. It is important to emphasize that, these operators discussed in this paper can be used also in EAs for solving other type of timetabling problems and/or other type of constraint satisfaction problems, such as high school course timetabling, final exam timetabling, nurse rostering or classroom assignment.

As a future work, different combinations of hill climbing methods at different resolution levels can be investigated. Furthermore, adaptive strategies, possibly considering the violations due to different constraint types, can be developed favoring the appropriate genetic operators during evolution.

Bibliography

Abramson, D., (1991) “Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms”, Management Science, 37(1):98-113.

Abramson, D., Dang, H. and Krisnamoorthy, M., (1999) “Simulated Annealing Cooling Schedules for the School Timetabling Problem”, Asia-Pacific Journal of Operational Research, 16,pp. 1-22.

Burke, E., Elliman, D., and Weare, R., (1994) “A Genetic Algorithm Based Timetabling System”, Proc. of the 2nd East-West Int. Conf. on Computer Technologies in Education, pp. 35-40.

Burke, E., Newall, J.P., and Weare, R.F., (1996) “A Memetic Algorithm for University Exam Timetabling”, Lecture Notes in Computer Science, 1153:241-250, Springer.

Cobb, H.G., (1990) “An investigation into the use of hypermutation as an adaptive operator in Genetic Algorithms Having Continuous, Time-dependent Nonstationary Environment”, NRL Memorandum Report 6760.

Colorni, A., Dorigo, M., and Maniezzo, V., (1992) “A genetic algorithm to solve the timetable problem”. Tech. rep. 90-060 revised, Politecnico di Milano, Italy.

Cladeira, JP, Rosa, AC, (1997) “School Timetabling using Genetic Search”, PATAT 97, pp. 115-122.

De Jong, K.A., (1975) *An Analysis of the Behavior of a Class of GeneticAdaptive Systems*, PhD Thesis, University of Michigan, Ann Arbour, MI.

Erben, W., Keppler, J., (1995) “A Genetic Algorithm Solving a Weekly Course-Timetabling Problem”, Proc. of the First Int. Conf. on the Practice and Theory of Automated Timetabling (ICPTAT), pp. 21-32, Napier University, Edinburgh.

Even, S., Itai, A., and Shamir, A., (1976) “On the Complexity of Timetable and Multicommodity Flow Problems”, SIAM J. Comput., 5(4):691-703.

Goldberg, D. E., (1989) Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading (MA).

Hertz, A., (1992) “Finding a feasible course schedule using a tabu search”, Discrete Applied Mathematics, 35, 255-270.

Holland, J. H., (1975) Adaptation in Natural and Artificial Systems, Univ. Mich. Press.

Kingston, J.H., (2001) “Modeling timetabling problems with STTL”, Springer Lecture Notes in Computer Science, 2079:309.

Leighton, F.T., (1979) “A graph coloring algorithm for large scheduling problems”, Journal of Reasearch of the National Bureau of Standards, 84:489-506.

Monfroglio, A., (1988) “Timetabling Through a Deductive Database: A Case Study”, Data & Knowledge Engineering, 3:1-27.

Moscato, P. ,and Norman, M. G.. (1992) “A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems”, Parallel Computing and Transputer Applications, pp. 177-186.

Ozcan E. (2003) “Towards an XML based standard for Timetabling Problems: TTML”, The 1st Multidisciplinary International Conference on Scheduling : Theory and Applications, pp.566-569.

Ozcan E., and Alkan, A., (2002) “Solving Time Tabling Problem using Genetic Algorithms”, Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling, pp.104-107.

Paechter, B., Rankin, R., C., Cumming, A. and Fogarty, T., C, (1998) “Timetabling the Classes of an Entire University with an Evolutionary Algorithm“, Proceedings of Parallel Problem Solving from Nature (PPSN V), pp. 865-874.

Pettey, C.S., Leuze, M. R., and Grefenstette, J. J., (1987) “A Parallel Genetic Algorithm”, Proc. of the Second Intl. Conf. for Genetic Algorithms, pp. 155-161.

Radcliffe, N. J., and Surry, P.D., (1994) “Formal memetic algorithms”, Evolutionary Computing: AISB Workshop, Springer Verlag, LNCS 865, pp. 1-16.

Ross, P., Corne, D., and Fang, H-L., (1994) “Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation”, Proceedings of PPSN III, pp. 556-565.

Ross, P., Corne, D., and Fang, H-L., (1994) “Fast Practical Evolutionary Timetabling”, Proceedings of AISBWorkshop on Evolutionary Computation.

Schaerf, A., (1996) “Tabu Search Techniques for Large High-School Timetabling Problems”, Proc. of the Fourteenth National Conference on AI, pp. 363-368, August.

Smith, R.E., and Goldberg, D. E.. (1992) “Diploidy and dominance in artificial genetic search”, Complex Systems, 6(3):251-285.

Werra, D. De, (1985) “An introduction to timetabling”, European Journal of Operations Research, 19:151-162.