Using an Evolution Strategy for a University Timetabling System with a Web Based Interface to Gather Real Student Data

Thomas B George
GE GAPS Suite 400
25925 Telegraph Road
Southfield, Michigan 48034 USA

Vitaliy Opalikhin
Automated Media Incorporated
12171 Beech Daly Road
Redford, Michigan 48239 USA

Chan-Jin Chung
Math & Computer Science Dept.
Lawrence Technological University
Southfield, MI 48075

Abstract

Determining the best, or near best timetable of lecture/courses for a university department, which optimizes enrollment, is a challenging problem. Our department is further challenged by restrictions on the availability of faculty, since adjunct faculty members who have other jobs teach 80% of all the classes offered by the department. Described herein is a platform developed to obtain both actual student generated data of course preferences and availability as well as department scheduling restrictions via a web interface. The web interface is essential, because our timetabling system incorporates students' preference data to generate schedule minimizing time conflicts. The system is coupled to an optimizer using an Evolution Strategy to generate the optimal or near optimal schedule of classes. In addition, the optimizer permits manipulation of initial constraints such as population size, the initial mutation rate, the step rule for varying the mutation rate, and the crossover point.

1 INTRODUCTION

In this paper we implement an evolutionary strategy, using actual student survey input, obtained using a web based interface, to solve university curriculum timetable problems. The web interface is essential, because our timetabling system incorporates students' preference data to generate schedule minimizing time conflicts. If successful, it is our ultimate goal to establish a working system that can be used at our university and also be readily adapted for use by other university departments.

Generating the best timetable consists of scheduling courses into appropriate timeslots and assigning resources such as instructors, classrooms, projection facilities, etc. In addition, consideration should be given to maximizing enrollment while ensuring students progress towards completion of their degree requirements. At our University, this requires consideration of scheduling courses both during the day for full time students and also in the evening for working students. By optimizing within these constraints, the university can provide the most economical and efficient operation while satisfying student demands on the curriculum, a critical criterion in this competitive day.

Because this problem incorporates a search space that is too large to explore exhaustively, an evolutionary algorithm is the natural choice. In fact, the class of timetabling problems has been intensively studied and known to be NP-hard (Even, Itai and Shamir, 1976).

1.1 THE TIMETABLING PROBLEM

There are many versions of the timetabling problem and they usually incorporate many nontrivial constraints of various kinds. The Timetabling problem here for our university can be described as follows (de Werra , 1985):

- There are a finite set of

  classes, $C = \{c_1, c_2, ..., c_{|C|}\}$,

  time slots $T = \{t_1, t_2, ..., t_{|T|}\}$,

  students $S = \{s_1, s_2, ..., s_{|S|}\}$,

  professors $P = \{p_1, p_2, ..., p_{|P|}\}$

- We have a function $t : C \mapsto P$ which maps each course to the professor that teaches it.
- $x_{i,k}, 1 \le k \le d_i, 1 \le x_{i,k} \le p$ the time that the k[th] teaching period of course c_i is given, p the number of possible teaching periods of the university.
- A solution to this problem is any assignment to the variable $x_{i,\,k}$ such that the following constraints are respected:

A professor cannot give more than one lecture at a time

$$\sum_{i,k}[x_{i,k} = m][p_n = p(c_i)] \le 1, \forall m, n$$

Similarily, a student cannot attend more than one lecture at a time

$$\sum_{i,k}[x_{i,k} = m][s_n = s(c_i)] \le 1, \forall m, n$$

- A number of soft constraints were not considered with this implementation such as the fitness of a one class room over another due to characteristics such as AV equipment, stadium seating, size, etc. These characteristics were built into the object oriented model used in the optimizer but, were omitted at this stage to facilitate analysis of the optimizer's behavior.

Our current objective function is to find a timetable that maximizes the number of student survey data *s*, which matches to the tuple in the timetable generated, while satisfying various constraints (predicted enrollment). In other words, by using the survey data gathered, we are trying to minimize time conflicts when students register classes. This will be a win-win strategy both for students and the University, because for students there will be high chance to take classes they plan to take, and the University can maximize number of students enrolled.

We can further divide constraints into hard constraints and soft constraints. Soft constraints are those that would allow us to assign values to generated timetables that allow us to rank them in order of preference, even though the maximized student enrollment is equal for all timetables being considered. An example might be noting that, given two timetables T1 and T2, T1 provides equal total enrollment to that of T2 but in more classes (average enrollment per class is lower). The university may choose to give T2 preference as it provides more efficient use of University Resources, or policy may determine that T1 is preferred as it improves the Faculty to Student ratio. At this stage, we do consider some soft constraints in ranking timetables during the "Survival of the Fittest Function" with one of the tested optimizers.

Hard constraints are those constraints that ensure that a given timetable is valid. The following hard constraints are implemented in our project:

- A Single Class may not be scheduled concurrently. In other words, if a Discrete Math Class must meet twice a week, the two sessions may not occur simultaneously. The total number of classes in session at any given moment must not exceed the total number of rooms available.

A single student may not attend two or more classes that occupy the same or overlapping time slots.

1.2 The Use of Real Student Constraint Data

Since selection of the best population size, mutation rate, mutation rate adaptive rule and crossover point remains more of an art than a science, we elected to develop a platform that will accept real course selection data from the student body, before a schedule is published, in order to generate the best schedule possible. Then, using both generated and real life data, the algorithm can be run repeatedly while varying different parameters to see which values give the best result most consistently.

The Math & Computer Science Department is expected to run the final version of platform over the course of 1-2 weeks, generating several solutions, from which the best may be selected. Thus, we are searching for the parameters which give the best peak performance over a small sample as opposed to the average performance (Eiben and Jelasity, 2002).

2 UNIVERSITY MCS CURRICULUM

Our University Math & Computer Science Department Curriculum consists of over 90 graduate and undergraduate courses, any combination of which may be offered in a given a semester. Required or core courses often must be offered both during the day and during the evening to accommodate the student body which consists of both full time students and working professionals.

A survey conducted, during the Fall of 2002, determined that 38% of our student body is employed full time, 33% part time and 84% of our student body lives off campus. With such a large percentage of working students, the need to schedule courses efficiently in the evenings becomes more pronounced.

Furthermore, we have a potentially severe constraint in that our department only has 14 lecture rooms available for use. While classes are sometimes scheduled in other department's facilities, this practice is undesirable due to the need for computer resources provided within the lecture rooms and short commuting distance between rooms.

3 WEB-BASED INTERFACE FOR COLLECTING INPUT DATA

The main idea of collecting data was to provide the students with the list of classes available for next semester and the times when the classes can be offered. Based on given options the students can choose the classes they are interested in and the most suitable time for them. After collecting the students' responses the program will create the list of classes that will possibly be offered next semester and survey data that later can be used to generate the final schedule. It will allow maximum number of students to attend offered classes.

Implementing a university timetabling system to schedule classes is a common timetabling problem (Erben and Keppler, 1995), (Paechter, Rankin, Cumming and

Fogarty, 1998), (Burke, Elliman and Weare, 1994), (Wong, Côté and Gely, 2002), (Foulds and Johnson, 2002), (Müller and Barták , 2001).

For our system the web based University Schedule Wizard (USW) was created. Web based applications in general are more convenient for users, for example, current or prospective students can complete survey from home without going on campus. USW was developed using JSP and Java Servlet for user interface, MySQL database and JRun Linux based web server. In order to provide rich graphic-user interface Javascript and DHTML technology were used. Also, the users can select courses by department, because list of all courses offered by the University can be huge.

The whole process of collecting student's preferences would take just a few minutes. The system was used to collect real data and had showed satisfactory results.

The process of collecting data through USW can be divided into two major parts: LTU Student Part and Administrator Part.

3.1 ADMINISTRATOR PART

The administrator part provides authorization based on login and password. Administrator part (Fig. 1) is used to maintain data that is needed to create the initial schedule file (initsch.txt). The initial schedule data includes list of classes, class times, semester information and number of available rooms. Using this part an administrator can generate and print output files that are used by a schedule generator.

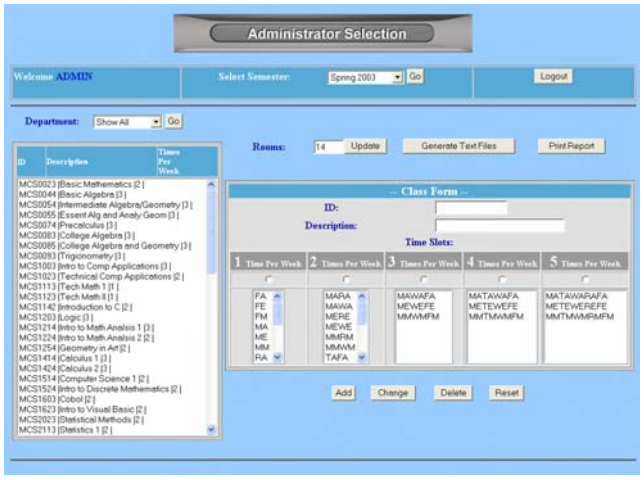


Fig 1. View of the Interface for the Administrator's Part.

The Administrator can add, modify or delete courses, indicates which times are available for scheduling, and generates output summarizing the curriculum and student data information.

University Schedule Wizard (USW) uses timeslots pre-built by administrator for representing class time. First letter in timeslot represents day of the week and second one represents part of the day. For example: MM - Monday Morning, RE – Thursday Evening, FA – Friday Afternoon. USW has five different types of timeslots that are depended upon the number of times per week the class is scheduled. For example: one class is scheduled for Monday Afternoon (MA) and another class - Tuesday Evening and Thursday Evening (TERE). USW implements timeslot maintenance for classes that are scheduled up to five times per week.

After administrator builds timeslots, administrator goes to maintenance part where he can add, modify or delete courses. For each course administrator provides course number, description and timeslots. The timeslots selected from the available pre-built timeslots are based on provided instructor's preferences. A course can be offered within one of five groups of timeslots (1 – 5 times per week). The timeslots from different groups cannot be mixed for same course.

At the present time system is being updated by exporting the list of courses from the different sources.

The administrator is also responsible for entering the number of rooms.

After administrator builds information for all the courses students can start submitting their preferences.

3.2 STUDENT PART

Student part provides authorization based on student id. Student part (Fig. 2) maintains a student's preferences that will be used to generate the student survey file (stusurvey.txt). Those preferences include list of classes the students want to take, class times and semester information.

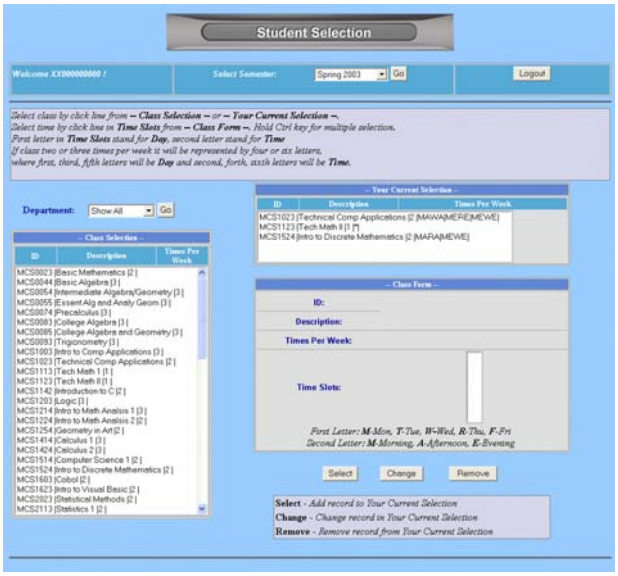


Fig 2. View of the Interface for the Student Part.

Students can add, modify or delete list of courses in which they wish to enroll. They can also indicate which times they are available to attend classes.

The students are not required keying in any information manually. All they need to do is to select a semester, class and timeslots from the timeslots provided by administrator for this class.

This simplicity allows user to avoid errors, to save a lot of time and also helps building reliable data. Student can select one or more classes or timeslots for each class. Students can also change his/her selections at any time. They modifications can be made until the administrator generates the final schedule.

USW also supports '*' that means any available timeslot for current class.

3.3 OUTPUT FORMAT

Data collected by USW is formatted to produce output of two text files: 'initsch' and 'stusurvey' (Fig. 3). The files then are used by the schedule generator to create the schedule of classes.

```
---initsch.txt---
ME TE WE MEWE TERE WEFE
14
MCS5013 ME TE WE
MCS5043 ME TE WE
MCS5103 ME TE WE
MCS5203 ME TE WE
MCS5303 MEWE TERE WEFE
MCS7993 TE

---stusurvey.txt---
AS000000647 MCS5013 *
CC000000708 MCS5013 TE
CC000000708 MCS5303 *
IT000001301 MCS5103 WE
IT000001301 MCS5203 *
IT000001301 MCS7993 ME
```

1st line displays all available timeslots

2nd line displays available rooms

All other lines display possible classes to schedule and available timeslots.

All lines display student id, class and time preferences.
'*' indicates the student is available to take this class any time (no constraint).

Fig. 3. View of text tile output.

This data is available for administrator review and is used by the built-in optimizer to generate schedules. The data is also available for export, for use by optimizer 2.

4 OPTIMIZER IMPLEMENTATION

Before we can present the results generated by the ES Optimizer, it is necessary to describe, in detail, the implementations. The ES Optimizer implements an algorithm based on a variation of ES(N+N) where the N is the selected population size and the mutation rate and adaptive mutation rate can be selected in advance by the user. The current implementation reads in the initialization data from the parameter selection on the main GUI (Fig. 4) and also from the two data input text files (Fig. 3). From the two text files and user selections, the optimizer initializes the following:

- The adaptive rate (based on the 1/5th rule).
- The initial Mutation Rate.
- The era (number of generations that pass before the 1/5 rule is evaluated.
- The Population Size (initial and maximum allowed to survive each generation).
- The crossover rate (% of a child's "genetic material" that is derived from the more fit of two parents.
- Generate N (10 is the default) Schedules to make up the population. Schedules contain every possible course.
- Each course in a schedule is randomly assigned a "legal" timeslot.

The fitness characteristics for each schedule are determined based on total enrollment, student preferences fulfilled (students can only attend one class during a given period), and average enrollment per course.

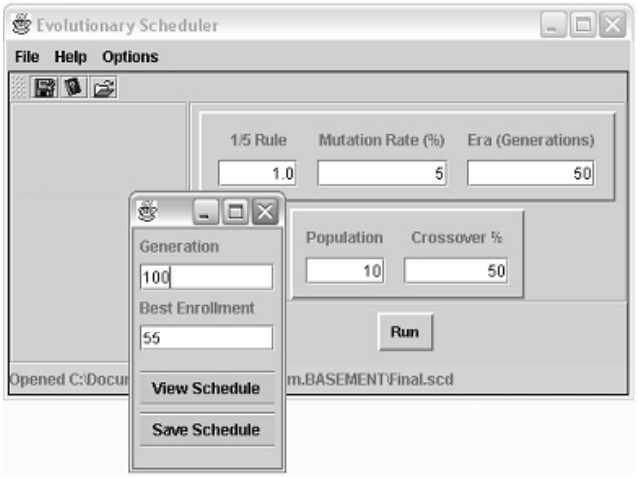


Fig. 4. View of the GUI Interface for the ES Optimizer.

This GUI can be used to set the adaptive function, base mutation rate, era size, crossover rate and population size.

4.1 THE EVOLUTIONARY LOOP

The Individual

Each individual in the population consists of a complete schedule. Courses scheduled illegally (violating any of the basic constraints) are marked cancelled. Thus an individuals simplest representation consists of a list of courses with the scheduled weekly meeting times appended.

Crossover

1. Allow each individual to mate with another. Mates are currently randomly selected with the following conditions:
   - Each individual is guaranteed to mate at least once and generate one child.
   - Schedules cannot mate with themselves.

2. Generate offspring. For each mated pair, the parent with the best fitness rating contributes X% (50% default) of scheduled courses, selected randomly. The second parent contributes the remaining courses. Thus every child contains all possible courses.

   Note that a single crossover point is NOT selected. Genes are selected randomly from the first parent until the selected contribution percentage is fulfilled. This should result in more degrees of freedom in traveling the search space. Order of classes within each individual (schedule) is arbitrary and should have no bearing on the problem or solution.

Mutation:

1. All of the children are subject to mutation (parents are not). Each course (gene) in a child has a (Mutation Rate) % chance of mutation. (Provision of a switch to allow mutation of parents may enhance the algorithm's ability to locate local maxima.)
2. If mutated, a new legal set of class times is randomly selected from the pool and assigned.

Enrollment (for each child):

1. Enrollment for each course within a child schedule is determined based on the student survey information. Students are not allowed to enroll twice in the same course or to enroll in a course that does not fit their availabile time constraints. They schedule all courses in order of decreasing preference until either all selections have been evaluated or their maximum enrollment has been reached.
2. The courses are sorted in order of decreasing enrollment.
3. Courses are assigned rooms (professor limit is equally valid) for each period scheduled in order of decreasing enrollment. If no rooms are available, that course is marked cancelled.
4. Students re-enroll in the courses that are not cancelled to determine the final enrollment

   Note: it is possible that enrollment would be better with another selection of course cancellations, but the student preferences would have been ignored in that case).
5. A count of preference violations and determination of average enrollment per class is now recorded in the schedule object for future comparison purposes.

Survival of the Fittest

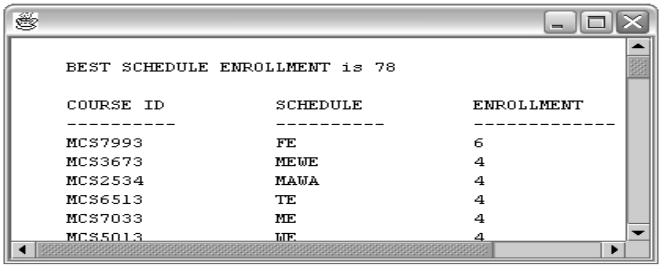
1. Duplicate schedules are eliminated from the population.
2. Schedules are now sorted in order of decreasing fitness (enrollment).
3. The top N (10) schedules survive (If current population is less than N, new members are generated).
4. If the best schedule is better than those from previous generations, an "Improved" Counter is Incremented and the best schedule is saved.
5. A generation counter is incremented.
6. If the generation is a multiple of the Era or Window Size:
   - The mutation rate /= OneFifthRule (1.2 is the default) if more than 1 of 5 generations have shown improvement.
   - The mutation rate *= OneFifthRule (1.2 is the default) if fewer than 1 of 5 generations have shown improvement. (Mutation rate is not permitted to exceed 100%)

Loop Decision:

If the population has not improved for an arbitrary number of generations (Era + 50 currently), then the loop exits. Otherwise it continues. The optimizer should run enough generations to ensure that evolution towards a better solution is unlikely.

EXIT COMPUTATION:

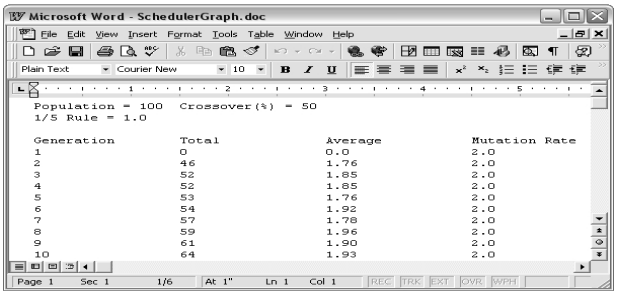
At this point, the best schedule (Fig. 4) can be viewed on screen or saved as a text file for later analysis. Data to specific to each generation (mutation rate, best schedule enrollment, etc.) are also recorded to a text file (Fig. 5) for analysis.



```
BEST SCHEDULE ENROLLMENT is 78

COURSE ID          SCHEDULE          ENROLLMENT
----------         ----------        --------------
MCS7993            FE                6
MCS3673            MEWE              4
MCS2534            MAWA              4
MCS6513            TE                4
MCS7033            ME                4
MCS5013            WF                4
```

Fig.5. Sample Portion Of The Output Of A Schedule



```
Microsoft Word - SchedulerGraph.doc
File Edit View Insert Format Tools Table Window Help
Plain Text  Courier New  10
Population = 100  Crossover(%) = 50
1/5 Rule = 1.0

Generation        Total          Average          Mutation Rate
1                 0              0.0              2.0
2                 46             1.76             2.0
3                 52             1.85             2.0
4                 52             1.85             2.0
5                 53             1.76             2.0
6                 54             1.92             2.0
7                 57             1.78             2.0
8                 59             1.96             2.0
9                 61             1.90             2.0
10                64             1.93             2.0
Page 1  Sec 1  1/6  At 1"  Ln 1  Col 1  REC TRK EXT OVR WPH
```

Fig. 6. Sample Output Of Data During The ES Process For Graphing And Statistical Analysis.

5 THE TEST DATA SETS AND TEST SUITES

We asked the Department Chair for the data to populate the "initsch.txt" file. He selected available time slots for each class, considering the instructors' preferences and availability. This is very important since adjunct faculty members who have other jobs teach 80% of all the classes offered by the department. Thus there are severe restrictions on what times a course may be offered.

In order to capture real world data, we requested all students (undergraduate and graduate) majoring in a math or computer science discipline to participate by logging into the web-based interface to select their courses as if they were registering. 69 courses listed in our published Spring 2003 Schedule were available for selection. The students provided their personal time constraints but were not told of any restraints for the classes.

We achieved participation of 51 students, which is approximately 20% of the total number of students available. Each student selected a mean of 2.6 courses (σ = 1.4) with a maximum selection of 6 courses. We used this data and ran both optimizers five times each with population sizes of 1, 3 and 10, using the following fixed parameters:

Mutation Rate = 5%

Crossover (Optimizer 2) = 50

Era (Optimizer 2) = 50

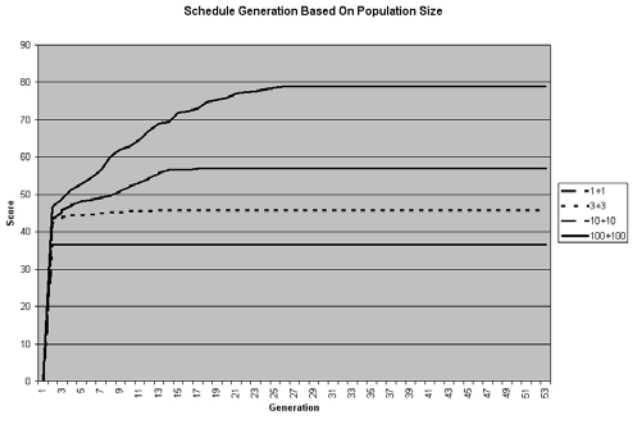


Fig. 7. Comparison Of Convergence Using 4 Different Population Sizes

6 CONCLUSIONS

As you can see from Figure 7, the optimizer converged quite rapidly. We also found that the data generated was very consistent. While the above figure represents the best of 5 runs for each of population sizes 1, 3, 10 and 100 respectively, additional runs at the same population size failed to yield better peak results. In other words, a separate trial running 20 more runs on a population size of 10 failed to find a solution with enrollment >= to 60. This is in contrast to multiple runs with population sizes of 100 in which no result provided an enrollment of less than 70.

This leads to the supposition that the trend is to find better solutions with increasing population size. However, additional runs with population sizes in excess of 100 failed to yield better results and additional runs with population sizes of 50 rendered near identical results. So, it appears that with the constraints or our data set, the optimizer is able to consistently find the near best solution with a population size of 50 or larger.

Knowing that courses had available a minimum of 5 allowed time slots and a maximum of 10 allowed time slots , the total search space encompasses more than 5^{69} possible combinations. Even allowing for restrictions that prevent more than 14 classes being scheduled in an overlapping time period (14 rooms) this still conservatively yields 5^{14} or approximately 6 billion combinations as a bottom limit. Clearly a population size of 50 << than the total search space and it is unlikely that we are exhaustively searching the set of all solutions.

In conclusion, this application of an evolution strategy for a university timetabling problem appears to be quite successful and we are ready to implement it to generate actual schedules for future terms. It is our hope that this system can also be adapted for use at other universities and for use in similar problems such as scheduling conferences, reducing the number of conflicts preventing attendees from attending conflicting presentations.

References

Erben, W., Keppler, J.: A Genetic Algorithm Solving a Weekly Course-Timetabling Problem (1995). Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95) http://citeseer.nj.nec.com/erben95genetic.html

Paechter, B., Rankin, R.C., Cumming, A., Fogarty T.C.: Timetabling the Classes of an Entire University with an Evolutionary Algorithm. Parallel Problem Solving From Nature --- PPSN V
http://citeseer.nj.nec.com/paechter98timeta bling.html

Burke, E., Elliman, D., Weare, R.: A Genetic Algorithm Based University Timetabling System (1994). Proceedings of the 2nd East-West International Conference on Computer Technologies in Education (Crimea, Ukraine, 19th-23rd Sept 1994). http://www.asap.cs.nott.ac.uk/publications/pdf/crimea94.pdf

Wong, T., Côté, P., Gely P.: Final Exam Timetabling: A Practical Approach(2002).
http://citeseer.nj.nec.com/544465.html

Foulds, L.R., Johnson, D.G.: A Decision Support System Improves Course Timetabling at the University of Waikato. 37th Annual ORSNZ Conference (2002). http://www.esc.auckland.ac.nz/Organisations/ORSNZ/conf37/Papers/Foulds.pdf

Müller, T., Barták R.: Interactive Timetabling(2001). http://arxiv.org/ftp/cs/papers/0109/0109022.pdf

Ross, P. Corne, D. Fang, H.: Successful Lecture Timetabling with Evolutionary Algorithms. Applied Genetic and other Evolutionary Algorithms: Proceedings of the {ECAI}'94 Workshop

Burke, E. Elliman, D., Weare, R.: Specialised Recombinative Operators for Timetabling Problems. Lecture Notes in Computer Science (1995)

Goltz, H., Matzke, D.: University Timetabling Using Constraint Logic Programming. Lecture Notes in Computer Science – Volume 1551, 1999

Woods, D., Trenaman, A.: Simultaneous Satisfaction of Hard and Soft Timetable Constraints for a University Department Using Evolutionary Timetabling. Artificial Intelligence and Cognitive Science (AIC1999)

Zervoudakis, K., Stamatopoulos, P.: A Generic Object-Oriented Constraint Based model for University Course Timetabling. Lecture Notes in Computer Science Volume 2079 (2001)

de Werra, D.: An Introduction to Timetabling. European Journal of Operational Research, 19:151-162, 1985

Even, S., Itai, A., Shamir, A.: On the Complexity of Timetable and Multicommodity Flow Problems. SIAM Journal of Computing. Vol. 5. No. 4. (1976) 691-703

Eiben, A.E., Jelasity, M.: A Critical Note on Experimental Research Methodology in EC. Institute of Electrical and Electronics Engineers 2002. 0-7803-7282-4/02

Walker, J.: RandomX Library for Java. The HotBits Genuine Random Number Source. (1996) http://www.fourmilab.ch/hotbits/

Chan, C., Gooi, H., Lim, M.: A Co-evolutionary Algorithm approach to a University Timetable System. Institute of Electrical and Electronics Engineers 2002. 0-7803-7282-4/02

Srinivasan, D., Seow, T., Zu, J.: Automated Time Table Generation Using Multiple Context Reasoning for University Modules. Institute of Electrical and Electronics Engineers 2002. 0-7803-7282-4/02

Michalewicz, Z., Dasgupta, D., Le Riche, R., Schoenauer, M.: Evolutionary Algorithms for Constrained Engineering Problems.

George, T, Chung, C.: Applying Evolution Strategies to a University Timetabling System. In 2002 Genetic and Evolutionary Computation Conference: Late-Breaking Papers (2002) 179-184

Negnevistsky, M.: Artificial Intelligence. Pearson Education Limited (2002) 217-243