2008 20th IEEE International Conference on Tools with Artificial Intelligence

# A Memetic Algorithm for the University Course Timetabling Problem

Sadaf N. Jat
Department of Computer Science
University of Leicester
University Road, Leicester LE1 7RH, UK
snj2@le.ac.uk

Shengxiang Yang
Department of Computer Science
University of Leicester
University Road, Leicester LE1 7RH, UK
s.yang@mcs.le.ac.uk

## Abstract

*The design of course timetables for academic institutions is a very hectic job due to the exponential number of possible feasible timetables with respect to the problem size. This process involves lots of constraints that must be respected and a huge search space to be explored, even if the size of the problem input is not significantly large. On the other hand, the problem itself does not have a widely approved definition, since different institutions face different variations of the problem. This paper presents a memetic algorithm that integrates two local search methods into the genetic algorithm for solving the university course timetabling problem (UCTP). These two local search methods use their exploitive search ability to improve the explorative search ability of genetic algorithms. The experimental results indicate that the proposed memetic algorithm is efficient for solving the UCTP.*

## 1 Introduction

Timetabling problems are often complicated by the details of a particular task. A general algorithm approach to a problem may turn out to be incapable for other problems because certain special constraints may be required in a particular instance of that problem. In the university course timetabling problem (UCTP), events (subjects, courses) have to be set into a number of timeslots while satisfying various constraints. Timetabling varies from university to university according to the resources and constraints. There is no known deterministic polynomial time algorithm for the UCTP. That is, the UCTP is a NP-hard problem [15].

Conventional computer based timetabling methods concern themselves more with simply finding the timetable but these methods are insufficient to satisfy all the required constraints. The solution to such problems using knowledge based or operation research based approaches is hard to develop. These approaches are often slow and can be inflexible because they are based on specific assumptions about the nature of the problem.

The application of computers to timetabling problems has a long and active history, which dates back almost as soon as computers were first built. The first generation of computer timetabling programs developed in the early 1960s were largely an attempt to reduce the associated administration work. Thereafter, programs were soon presented with the aim of fitting classes and teachers to periods. In 1964, Broder [6] and Cole [12] both presented heuristic approaches to timetabling. In 1967, Welsh and Powell [26] pointed out the similarity between the timetabling problem and the one of coloring the vertices of a graph. Coloring the graph amounts to placing courses in appropriate periods. The algorithm they presented was similar to Broder's algorithm [6]. Researchers have proposed various timetabling approaches by using constraint-based methods, population-based approaches (e.g., genetic algorithms (GAs), ant colony optimization, and memetic algorithms (MAs)), meta-heuristic methods (e.g., tabu search, simulated annealing, and great deluge), variable neighbourhood search (VNS), hybrid meta-heuristics, and hyper-heuristic approaches, etc. In the last decade, a lot of research papers have been published. A comprehensive review on the timetabling problem was described in [24, 13] and recent research directions in timetabling were described in [10].

In this paper, a memetic algorithm is proposed for the UCTP, which integrates two local search techniques into GAs: one is based on events and the other is based on timeslots. MAs [17] are a class of meta-heuristic methods, which combine the population-based global search, e.g., GAs, with local search made by individuals [21]. We apply two local search methods on selected individuals in order to improve their quality. With the help of both the local search methods, GAs become more powerful for the UCTP.

The rest of this paper is organized as follows. The next section briefly describes the UCTP. Section 3 presents the MA proposed in this paper for the UCTP. Experimental results of comparing the proposed MA and other algorithms

1082-3409/08 $25.00 © 2008 IEEE
DOI 10.1109/ICTAI.2008.126

from the literature are reported and discussed in Section 4. Section 5 concludes this paper with discussions on future work.

## 2 The University Course Timetabling Problem

According to Carter and Laporte [13], the UCTP is a multi-dimensional assignment problem, in which students and teachers (or faculty members) are assigned to courses, course sections or classes and events (individual meetings between students and teachers) are assigned to classrooms and time slots.

In a UCTP, we assign an event (courses-lectures) into a time slot and also assign a number of resources (students,rooms) in such a way that there is no conflict between the rooms, timeslots and events. As mentioned by Rossi-Doria et al. [18], the UCTP problem consists of a set of $n$ events (classes, subjects) $E = \{e_1, e_2, ..., e_n\}$ to be scheduled in a set of 45 timeslots $T = \{t_1, t_2, ..., t_{45}\}$ nine for each day in a five day week, a set of $m$ available rooms $R = \{r_1, r_2, ..., r_m\}$ in which events can take place, a set of $k$ number of students $S = \{s_1, s_2, ..., s_k\}$ who attend the events and a set of $l$ available features $F = \{f_1, f_2, ..., f_l\}$ that are satisfied by rooms and required by each event.

In addition, interrelationships between these sets are given by five matrices. The first matrix shows which event is attended by which students. The second matrix indicates whether two events can be scheduled in the same timeslot or not. The third matrix gives the features that each room possesses. The fourth matrix gives the features required by each event. The last matrix lists the possible rooms to which each event can be assigned.

Usually, a matrix is used for assigning each event to a room $r_i$ and a timeslot $t_i$. Each pair of $(r_i, t_i)$ is assigned a particular number corresponding to an event. If a room $r_i$ in a timeslot $t_i$ is free or no event is placed then $-1$ is assigned to that pair. With this assignment we assure that there will be no more than one event assigned to this pair so that one of the hard constraint will always been satisfied.

For the room assignment we used a matching algorithm described by Rossi-Doria [20]. For every timeslot, there is a list of events taking place in it and a preprocessed list of possible rooms to which the placement of events can be occurred. The matching algorithm uses a deterministic network flow algorithm and gives the maximum cardinality matching between rooms and events.

In general, the solution to a UCTP can be represented in the form of an ordered list of pairs $(r_i, t_i)$, of which the index of each pair is the identification number of an event $e_i \in E$ $(i = 1, 2, \cdots, n)$. For example, the timeslots and rooms are allocated to each event in an ordered list of pairs like:

$$(2, 4),\ (3, 30),\ (1, 12),\ \cdots,\ (2, 7),$$

where timeslot 4 and room 2 are allocated to event 1, timeslot 30 and room 3 are allocated to event 2, and so on.

The real world university course timetable problem consists of different constraints: some are hard constraints and some are soft constraints. In this paper, we will test our proposed algorithm on the problem instances discussed in [20]. We deal with the following hard constraints:

- No student attends more than one events at the same time;
- The room is big enough for all the attending students and satisfies all the features required by the event;
- Only one event is in a room at any timeslot.

There are also soft constraints which are penalised equally by their occurrences:

- A student has a class in the last timeslot of a day;
- A student has more than two classes in a row;
- A student has a single class on a day.

The goal of the UCTP is to minimise the soft constraint violations of a feasible solution (a feasible solution means that no hard constraint violation exists in the solution).

## 3 The Proposed Memetic Algorithm

It is generally believed that MAs are successful because they combine the explorative search ability of recombinative evolutionary algorithms and the exploitive search ability of local search methods [7]. The term memetic algorithm was invented by Moscato [17] from Richard Dawkin's [14] term "meme". Memetic algorithms [17, 18] are a kind of evolutionary algorithms that integrate knowledge of the problem in the form of heuristics, approximate algorithms, local search techniques, specialised recombination operators, and truncated exact methods, etc [23].

Many researchers have applied MAs to address timetabling problems by combining GAs and local search techniques, e.g., Peachter et al. [19], Rossi-Doria et al. [20, 21], Alkan and Ozcan [4], Abdullah et al. [3], and Chiarandini et al. [11]. In this paper, a memetic algorithm is proposed for the UCTP, which combines two local search techniques into GAs. These two local search techniques are based on three neighbourhood structures, denoted as N1, N2, and N3. They are described as follows:

- N1: the neighbourhood defined by an operator that moves one event from a timeslot to a different one

**Algorithm 1** The proposed memetic algorithm

```
1: input : A problem instance I
2: for $i = 1$ to population size do
3:     $s_i \leftarrow$ random initial solution
4:     $s_i \leftarrow$ solution $s_i$ after Local Search 1 (LS1)
5:     $s_i \leftarrow$ solution $s_i$ after Local Search 2 (LS2)
6: end for
7: sort population by fitness
8: while termination condition not reached do
9:     select two parents from population by tournament selection
10:    $s \leftarrow$ child solution after crossover with a probability $P_c$
11:    $s \leftarrow$ child solution after mutation with a probability $P_m$
12:    $s \leftarrow$ child solution after applying Local Search 1 (LS1)
13:    $s \leftarrow$ child solution after applying Local Search 2 (LS2)
14:    child solution $s$ replaces the worst member of the population
15:    sort population by fitness
16:    $s_{best} \leftarrow$ best solution in the population
17: end while
18: output : The best solution $s_{best}$ achieved for I
```

- N2: the neighbourhood defined by an operator that swaps the timeslots of two events

- N3: the neighbourhood defined by an operator that permutes three events in three distinct timeslots in one of the two possible ways other than the existing permutation of the three events.

Algorithm 1 shows the outline of the MA proposed in this paper for the UCTP. In the MA, we first initialize the population by randomly creating each individual (via assigning a random timeslot for each event according to a uniform distribution) and applying the matching algorithm to allocate rooms for events. Then, two local search methods, Local Search 1 (LS1) and Local Search 2 (LS2), are applied in order to each member of the population. LS1 uses the neighbourhood structures N1, N2, and N3 to move events to timeslots and then uses the matching algorithm to allocate rooms to events and timeslots. With LS2, we take the timeslot with the worst penalty value from a set of randomly selected timeslots and try to improve it by trying to move each event in that timeslot to another one in the neighbourhood N1 and then using the matching algorithm for room allocations for those involved events.

After the initialization of the population, we use the steady state genetic algorithm model as mentioned in [11], where only one child solution is generated with selection, crossover and mutation at each generation. The child then will be improved by LS1 and LS2. In the end, the worst population member is replaced with the new child individual. The iteration continues until one termination condition is reached, e.g., a preset time limit $t_{max}$ is reached.

**Algorithm 2** The procedure of Local Search 1 (LS1).

```
1: input : Individual I selected from the population
2: while Termination condition not reached do
3:     for $i = 1$ to the total number of events do
4:         if event $i$ is infeasible then
5:             if there is untried move left then
6:                 calculate the next move (first in N1, then N2, and finally N3)
7:                 apply the matching algorithm to the timeslots affected by the move and delta-evaluate the result.
8:                 if the move reduces hard constraint violation then
9:                     make the move and go to line 3
10:                end if
11:            end if
12:        end if
13:    end for
14:    if any hard constraint violations remain then
15:        end LS1 (go to line 30)
16:    else
17:        for $i = 1$ to total number of events do
18:            if event $i$ has soft constraint violation then
19:                if there is untried move left then
20:                    calculate the next move (first in N1, then N2, and finally N3)
21:                    apply the matching algorithm to the timeslots affected by the move and delta-evaluate the result
22:                    if the move reduces soft constraints violation then
23:                        make the move and go to line 17
24:                    end if
25:                end if
26:            end if
27:        end for
28:    end if
29: end while
30: output : A possibly improved individual I
```

## 3.1 Local Search 1

Algorithm 2 summarises Local Search 1 (LS1). LS1 works on all events. Here, we suppose that each event is involved in soft and hard constraint violations. LS1 works in two steps. In the first step (line 3-13 in Algorithm 2), it checks the hard constraint violations of each event while ignoring its soft constraint violations. If there are hard constraint violations for an event, LS1 tries to resolve them by applying moves in the neighbourhood structures N1, N2, and N3 in order[1] until a termination condition is reached, e.g., an improvement is reached or the maximum number

[1]For the event being considered, potential moves are calculated in a strict order. First, we try to move the event to the next timeslot, then the next, then the next, etc. If this search in N1 fails, we then search in N2 by trying to swap the event with the next one in the list, then the next one, and so on. If the search in N2 also fails, we try a move in N3 by using one different permutation formed by the event with the next two events, then with the next two, and so on.

**Algorithm 3** The procedure of Local Search 2 (LS2)

```
1: input : Individual I after LS1 is applied
2: $S$ := randomly select a preset percentage of timeslots from the total timeslots of $T$
3: for each timeslot $t_i \in S$ do
4:     for each event $j$ in timeslot $t_i$ do
5:         calculate the penalty value of event $j$
6:     end for
7:     sum the total penalty value of events in timeslot $t_i$
8: end for
9: select the timeslot $w_t$ with the biggest penalty value from $S$
10: for each event $i$ in $w_t$ do
11:     calculate a move of event $i$ in the neighbourhood structure N1
12:     apply the matching algorithm to the timeslots affected by the move
13:     compute the penalty of event $i$ and delta-evaluate the result
14: end for
15: if all the moves together reduce hard or soft constraint violations then
16:     apply the moves
17: else
18:     delete the moves
19: end if
20: output : A possibly improved individual I
```

of steps $s_{max}$ is reached, which is set to different values for different problem instances. After each move, we apply the matching algorithm to the timeslots affected by the move and try to resolve the room allocation disturbance and delta-evaluate the result of the move (i.e., calculate the hard and soft constraint violations before and after the move). If there is no untried move left in the neighbourhood for an event, LS1 continues to the next event. After applying all neighbourhood moves on each event, if there is still any hard constraint violation, then LS1 will stop; otherwise, LS1 will perform the second step (line 17-27 in Algorithm 2).

In the second step, after reaching the state of a feasible solution, LS1 then deals with soft constraints and again performs a similar process as in the first step on each event to reduce its soft constraint violations. For each event, LS1 tries to make moves in the neighbourhood N1, N2 and N3 in order without violating the hard constraints. For each move, the matching algorithm is applied to allocate rooms to affected events and the result is delta-evaluated. When LS1 finishes, we get a possibly improved and feasible individual. After that, we apply LS2 on this individual.

### 3.2 Local Search 2

Algorithm 3 shows the pseudo-code of Local Search 2 (LS2). The basic idea of LS2 is to choose a high penalty timeslot that may have a large number of events involving hard and soft constraints. LS2 first randomly selects a preset percentage of timeslots[2] (e.g., 20% as used in this paper) from the total timeslots of $T$. Then, it computes the penalty of each selected timeslot and chooses the timeslot $w_t$ that has the biggest penalty value for local search. In this way, LS2 aims to help improve the existing result of LS1.

After taking the worst timeslot, LS2 tries a move in the neighbourhood N1 for each event of $w_t$ and checks the penalty value of each event before and after applying the move. If all moves in $w_t$ together reduce the hard and/or soft constraint violations, then we apply all the moves; otherwise, we do not make the moves. In this way, LS2 can not only check the worst timeslot but also reduce the penalty value for some events by moving them to other timeslots. In general, LS2 can enhance the individuals of the population and increase the quality of the feasible timetable by reducing the number of constraint violations.

### 3.3 Genetic Operators

The proposed MA uses the steady-state GA model. One offspring is generated from the current population at each generation using the following genetic operators and relevant parameters, which were also used in [20].

Selection: Tournament selection of size 2 is used, where two parents are randomly selected from the population and the fitter one is used as a parent. At each generation, the tournament selection is applied twice to select two parents for reproduction.

Crossover: A uniform crossover operator is used with a probability $P_c = 0.8$. It first assigns to each event in the offspring a timeslot from one of the two parents randomly and then allocates rooms to events in each non-empty timeslot.

Mutation: A mutation operator is used with a probability $P_m = 0.5$. It randomly selects a neighbourhood structure N1, N2, or N3, and makes a move in the selected neighbourhood to mutate an individual.

## 4 Experimental Study

The program is coded in GNU C++ with version 4.1 and run on a 3.20 GHz PC. We use a set of benchmark problem instances to test our algorithm, which were proposed by Ben Paechter for the timetabling competition, see [22]. Although these problem instances lack many of the real world problem constraints and issues [16], they allow comparison of our approach with current state of the art techniques on these instances.

[2]Rather than choosing a worst timeslot out of all the timeslots, we randomly select a set of timeslots and then choose the worst time slot. This is because for each selected timeslot we need to calculate its penalty value, which costs time. Via selecting a set of timeslots instead of all timeslots, we try to balance between the computational time and the quality of the algorithm.

**Table 1. Three groups of problem instances**

| Class | Small | Medium | Large |
|---|---|---|---|
| Number of events | 100 | 400 | 400 |
| Number of rooms | 5 | 10 | 10 |
| Number of features | 5 | 5 | 10 |
| Per room approximate features | 3 | 3 | 5 |
| Percentage (%) of features used | 70 | 80 | 90 |
| Number of students | 80 | 200 | 400 |
| Maximum events per student | 20 | 20 | 20 |
| Maximum students per event | 20 | 50 | 100 |

**Table 2. Comparison of algorithms on small and medium problem instances**

| Datasets | MA | | RIIA | | HEA | GBHH | VNS | THHS | LS | EA | AA | FA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best | Median | Best | Median | Best | Best | Best | Best | Median | Best | Median | Best |
| S1 | **0** | 0 | **0** | 0 | **0** | 6 | **0** | 1 | 8 | **0** | 1 | 10 |
| S2 | **0** | 0 | **0** | 0 | **0** | 7 | **0** | 2 | 11 | 3 | 3 | 9 |
| S3 | **0** | 0 | **0** | 0 | **0** | 3 | **0** | **0** | 8 | **0** | 1 | 7 |
| S4 | **0** | 0 | **0** | 0 | **0** | 3 | **0** | 1 | 7 | **0** | 1 | 17 |
| S5 | **0** | 0 | **0** | 0 | **0** | 4 | **0** | **0** | 5 | **0** | **0** | 7 |
| M1 | 227 | 229.5 | 242 | 245 | 221 | 372 | 317 | **146** | 199 | 280 | 195 | 243 |
| M2 | 180 | 185 | 161 | 162.6 | **147** | 419 | 313 | 173 | 202.5 | 188 | 184 | 325 |
| M3 | **235** | 238.5 | 265 | 267.8 | 246 | 359 | 357 | 267 | 77.5%In | 249 | 248 | 249 |
| M4 | **142** | 155 | 181 | 183.6 | 165 | 348 | 247 | 169 | 177.5 | 247 | 164.5 | 285 |
| M5 | 200 | 203 | 151 | 152.6 | **130** | 171 | 292 | 303 | 100%In | 232 | 219.5 | 132 |

Table 1 represents the data of timetabling problem instances of three different groups: 5 small instances, 5 medium instances, and 1 large instance. In LS1 of our MA, the maximum number of steps per local search $s_{max}$ is set to different values for different problem instances (200 for small instances, 1000 for medium instances, and 2000 for the large instance). There were 50 runs of the algorithm for each problem instance. For each run, the maximum run time $t_{max}$ was set to 90 seconds for small instances, 900 seconds for medium instances, and 9000 seconds for the large instance.

We compare our MA with other algorithms on the 11 timetabling problem instances. Table 2 gives the comparison of the experimental results of our algorithm with the available results of other algorithms in the literature on the small and medium timetabling problem instances. In the table, S1 represents small instance 1, S2 represents small instance 2, and so on, and M1 represents medium problem instance 1, M2 represents medium problem instance 2, and so on. In Table 2, the term "%ln" represents the percentage of runs that failed to obtain a feasible solution. The "Best" indicates the best result among a number of runs. We present the best of all the algorithms in the bold font. The algorithms compared in the table are described as follows:

- MA: our memetic algorithm approach

- RIIA: The randomised iterative improvement method by Abdullah et al. [1]. This paper presented a composite neighbourhood structure with a randomised iterative improvement algorithm.

- VNS: The variable neighbourhood search by Abdullah et al. [2]. In this paper they used a variable neighbourhood search approach based on the random-descent local search with an exponential Monte Carlo acceptance criteria.

- THHS: The tabu-based hyper-heuristic search by Burke et al. [8]. They introduced a tabu-search hyper heuristics where a set of low level heuristics compete with each other and this approach is tested on the course timetabling and nurse rostering problems.

- EA: The evolutionary algorithm by Rossi-Doria et al. [20]. They used an evolutionary algorithm with local search to solve the UCTP and also compared several metaheuristics methods on the UCTP.

- HEA: The hybrid evolutionary algorithm by Abdullah et al. [3]. They tested a light mutation operator

**Table 3. The t-test results of comparing MA against EA.**

| function | S1 | S2 | S3 | S4 | S5 |
|---|---|---|---|---|---|
| t-test | 8.4524 | 9.9812 | 7.9047 | 7.754 | 4.334 |
| function | M1 | M2 | M3 | M4 | M5 |
| t-test | 9.5912 | 14.9345 | 9.6104 | 19.938 | 8.8526 |

followed by a randomised iterative improvement algorithm on the UCTP.

- LS: The local search method by Socha et al. [25]. They used a random restart local search for UCTP and compare with an ant algorithm.
- AA: The ant algorithm is used by Socha et al. [25]. They developed a first ant colony optimization algorithm with the help of construction graph and a pheromone model appropriate for the university course timetabling.
- FA: The fuzzy algorithm by Asmuni et al. [5]. In this paper they focused on the issue of ordering events by simultaneously considering three different heuristics using fuzzy methods.
- GBHH: The graph-based hyper heuristic by Burke et al. [9]. They employed tabu search with graph-based hyper-heuristics on the UCTP and examination timetabling problems.

From Table 2, it can be seen that our proposed MA is better than the fuzzy algorithm [5] and graph based approach [9] on 9 out of the 10 small and medium problem instances (except on M5). Our algorithm also obtained better results than VNS [2] and EA [20] on all of the medium problem instances and tied on some or all of the small problem instances. It also gives better results than local search [25] on 9 of the 10 problem instances and is better than the ant algorithm [25] on 7 of the data set (with one tie on S5). When comparing with the result of hybrid evolutionary approach [3] and RIIA [1], it is quite interesting that our approach is better on 3 of the same medium problem instances (except on M2 and M5) and ties on all small problems. Finally, the results of our approach are better than the tabu-based hyper heuristic search [8] on most of the problem instances.

The proposed MA did not achieve a feasible result on the large instance within the running time of 9000 seconds within 50 runs. On this large instance, other algorithms from the literature also failed to give a feasible result except [9], [3], [25], and [5]. This result indicates that the neighbourhood structures may need further improvement to give feasible results for the large instance.

The results of the statistical comparison of our MA against the EA by Rossi-Doria et al. [20] using the t-test are shown in Table 3. The t-test result is based on 50 runs of their EA and our MA on small and medium instances. The t-test is carried out with 98 degree of freedom at a 0.05 level of significance. It can be seen that the performance of our proposed MA is significantly better than the EA. MA works more efficiently as compared to the EA and gives a better performance on all small and medium problems.

To summarise, the performance of our memetic algorithm was tested on the benchmark problems [22] and when compared with other published work, it can be seen that our proposed GA with local search is capable of producing some of the best results.

## 5 Conclusions and Future Work

This paper presents a MA for solving the university course timetabling problem. The MA combines the GA with two local search techniques. With only the first local search, GA does not perform well in the experiments as mentioned in [20]. But we have enhanced the functionality of the GA by introducing a second local search method. Based on the experimental results, it is clear that, with the help of the powerful local search methods, the proposed MA can obtain high quality solutions that satisfy different kinds of timetabling constraints. The proposed MA is capable of finding a near optimal solutions for the test problems. These results also show that by integrating appropriate neighbourhood moves, GAs can get the best solutions for the UCTP.

In the future, more work needs to be done by improving the genetic operator or developing new neighbourhood techniques based on different problem constraints because we believe that MAs can be improved by applying advanced genetic operators, heuristics, and evaluation routines. The inter-relationship of these techniques and proper placement of these techniques in an algorithm may furthermore lead to better results.

## References

[1] S. Abdullah, E. K. Burke, and B. McCollum. Using a randomised iterative improvement algorithm with composite neighbourhood structures. *Proc of the 6th Int Conf on Metaheuristic*, pp. 153-169, 2007.

[2] S. Abdullah, E. K. Burke, and B. McCollum. An investigation of variable neighbourhood search for university course timetabling. *Proc of the 2nd Multidisciplinary Conf on Scheduling: Theory and Applications*, pp. 413–427, 2005.

[3] S. Abdullah, E. K. Burke, and B. McCollum. A Hybrid Evolutionary Approach to the University Course Timetabling Problem. *Proc of the 2007 IEEE Congres on Evol Comput.*, pp. 1764–1768, 2007.

[4] A. Alkan and E. Ozcan. Memetic algorithms for timetabling evolutionary computation. *Proc of the 2003 IEEE Congress on Evol Comput.*, vol. 3, pp. 1796–1802, 2003.

[5] H. Asmuni, E. K. Burke, and J. M. Garibaldi. Fuzzy Multiple Heuristic Ordering for Course Timetabling. *Proc of the 5th UK Workshop on Comput Intell.*, pp. 302-309, 2005.

[6] S. Broder. Final examination scheduling. *Comm of the ACM*, 7(8): 494–498, 1964.

[7] E. K. Burke and D. J. Landa Silva. The design of memetic algorithms for scheduling and timetabling problems. In N. Krasnogor, W. Hart, and J. Smith (eds.), *Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing*, vol. 166, pp. 289–312, 2004.

[8] E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, 9(6): 451–470, 2003.

[9] E. K. Burke, B. MacCloumn, A. Meisels, S. Petrovic, and R. Qu. A Graph-based hyper heuristic for timetabling problems. *European J of Oper Research*, 176: 177–192, 2006.

[10] E. K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European J of Oper Research*, 140(2): 266-280, 2002.

[11] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling. *J of Scheduling*, 9(5): 403–432, 2006.

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

[13] M. W. Carter and G. Laporte. Recent developments in practical course timetabling. *Proc of the 2nd Int Conf on Practice and Theory of Automated Timetabling*, LNCS 1408, pp. 3–19, 1998.

[14] R. Dawkins. The Selfish Gene. *Oxford University Press*, 1976.

[15] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4): 691–703, 1976.

[16] B. McCollum. University Timetabling: Bridging the Gap between Research and Practice. *Proc of the 6th Int Conf on the Practice and Theory of Automated Timetabling*, pp. 15–35, 2006.

[17] P. Moscato. On evolution, search, optimization, genetic algorithms and martial art: towards memetic algorithms. *caltech concurrent computation program*, Technical Report, 1989.

[18] P. Moscato. Memetic algorithms: A short introduction. In: *New Ideas in Optimisation, Mcgraw-Hill'S Advanced Topics In Computer Science Series*, pp. 219–234.

[19] B. Paechter, A. Cumming, M. G. Norman, and H. Luchian. Extensions to a memetic timetabling system. *Proceedings of the 1st International Conference on Practice and Theory of Automated Timetabling*, LNCS 1153, pp. 251–265, 1996.

[20] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stützle. A comparison of the performance of different metaheuristics on the timetabling problem. *Lecture Notes in Computer Science 2740*, pp. 329–351,2002.

[21] O. Rossi-Doria and B. Paechter. A memetic algorithm for university course timetabling. *Proc. of Combinatorial Optimisation*, 2004.

[22] http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html

[23] http://www.cs.nott.ac.uk/ gxk/papers/nbhPhDthesis.pdf

[24] A. Schearf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):pp. 87–127, 1999.

[25] K. Socha, J. Knowles, and M. Samples. A max-min ant system for the university course timetabling problem. *Proc. of the 3rd Int. Workshop on Ant Algorithms, ANTS 2002*, LNCS 2463, pp. 1–13, 2002.

[26] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and it's application to timetabling problems. *Computer Journal*, 10(1): 85-86, 1967.