A Memetic Algorithm for Timetabling

Nguyen Quoc Viet Hung, Ta Quang Binh, Duong Tuan Anh

Abstract— Examination timetabling problems are real world constraint optimization problems that are often coped with in educational institutions, such as universities. In this paper, we present a memetic algorithm for university examination timetabling. Memetic algorithm is the combination of genetic algorithm and local search technique to enhance solution quality and reduce searching time. We perform preliminary experiments of the algorithm on real data sets and results are promising.

Index Terms - genetic algorithm, memetic algorithm, local search, timetabling.

I. INTRODUCTION

The difficulty of developing appropriate examination timetables for universities is increasing. Universities are enrolling more students into a wider variety of courses in many different fields. For examples, at the Ho Chi Minh City University of Technology, approximately 10000 students have to be fitted into about 320 exams over two and a half week period. Consequently, examination timetabling is a difficult combinatorial optimization problem and in fact a NP-complete problem.

The examination timetabling problem regards the scheduling for the exams of a set of university courses, avoiding overlaps of exams of courses having common students, and spreading the exams for the students as much as possible.

This scheduling problem has been studied extensively and many approaches developed to solve it. Surveys of different methods ([8]) for exam timetabling classify the different approaches as sequential construction heuristics, constraint programming, and local search (genetic algorithms, simulated annealing and tabu search).

Heuristics used in solving graph coloring problem are examples of sequential construction heuristics. These heuristics have been widely applicable for automated exam timetabling since 1960’s ([8]).

One of the major approaches in exam timetabling over the years has been constraint programming approach. Examples of these approaches are provided by Boizmault, Delon and Peridy, 1996 ([4]), David, 1997 ([12]) and Reis, Teixeira and Oliveira, 2000 ([15]). Boizmault et al. used the constraint logic programming language CHIP to solve the exam timetabling problem. David developed a local repair technique to solve exam timetabling problem modeled as a constraint satisfaction problem. Reis et al. used the constraint logic programming language ECLiPSe to solve the exam timetabling problem.

Local search approaches play an important role in the exam timetabling literature. White and Xie ([17]) and Di Gaspero and Schaefer ([13]) used tabu search method in exam timetabling. White and Xie kept two tabu lists, the usual short-term tabu list, and a long-term tabu list keeps track of the most moved exams. Di Gaspero and Schaefer used a single tabu list, but when exams are added to this list, it is for a random determined number of iterations.

Over the last few years, simulated annealing has been investigated for exam timetabling with some level of success. In 1991, Abramson ([1]) applied simulated annealing to construct high school timetables. In 1996 Thompson and Dowsland ([16]) considered an adaptive cooling technique where the temperature is automatically reduced or increased depending upon the success of the move. Burke et al. ([5]) proposed a time-predefined variant of simulated annealing.

A growing number of researchers are now turning to genetic algorithm as a powerful method of solving difficult timetabling problems. Examples of genetic algorithm for exam timetabling are provided by Corne, Ross and Fang, 1994 ([10]), Burke, Elliman and Weave, 1995 ([6]), Corne and Ross, 1996 ([11]). It is now well-established that it is hard for a ‘pure’ genetic algorithm to ‘fine tune’ the search in complex spaces. Researchers and practitioners have shown that a combination of global and local search is almost always beneficial. Memetic algorithm is the combination of genetic algorithm and local search to enhance solution quality and reduce searching time. The first memetic algorithm for solving exam timetabling is given by Burke, Newall and Weave, 1996 ([7]).

In this paper, we present a memetic algorithm for university examination timetabling. We perform preliminary experiments of the algorithm on the real data set from the HoChiMinh City (HCMC) University of Technology.

The approach presented here is close in spirit to the one described in [7]. However, the main difference between two approaches is that our hill-climbing operator is more intelligent and our algorithm yields better performance.

The paper is organized as follows. The next section introduces the examination timetabling problem. In section 3, we elaborate our memetic algorithm for the examination
timetabling problem. Section 4 describes experimental results and gives comparisons to some existing works. Conclusions and future works are given in section 5.

II. EXAMINATION TIMETABLING PROBLEM

The development of an examination timetable requires scheduling a number of examinations (‘exam’) in a given set of exam sessions (‘time slot’ or ‘session’) so as to satisfy a given set of constraints. The most common forms of constraints in the exam timetabling at the HCMC University of Technology are:

C1. Exam Clashing: No student may have two exams in the same session.
C2. Room Clashing: Not more than two exams can be assigned to the same room.
C3. Time Clashing: All subgroups of a certain exam have to be assigned in the same period.
C4. Exam Spread: Each student’s exams should spread over the exam season.
C5. Room Restriction: When there are several student subgroups taking the same exam, rooms for that exam with all student groups should be arranged near to one another.
C6: Staff Restriction: At every period, there should not be too many exams.
C7. Release/Due Date Restrictions: Some exams have to start after a release date and to finish before a due date.
C8. Room Utilization: Rooms are assigned to exams in such a way that the room utilization can be maximized.
C9. Predefined Room Constraint: Some exams must be assigned special rooms.

Notice that in HCMC University of Technology, students enrolling in courses are divided into exam subgroups. These subgroups are the units used when assigning students to rooms to sit for exams. The first three kinds of constraints, C1, C2, C3 and C9, are hard constraints that must be satisfied, but the rest can be considered as soft constraints that should be satisfied to improve the solution.

III. A MEMETIC ALGORITHM FOR SOLVING EXAM TIMETABLING PROBLEM

Genetic algorithm is a general purpose optimization tool based on Darwin’s theory of evolution. It has the capability of producing optimized solutions even when the dimensions of the problem increase and for this reason it has been successfully applied to a wide variety of problems.

Genetic algorithm operates on a population of solutions represented by some coding. Each member of the population consists of a number of genes, each of which is a unit of information. New solutions are obtained by combining genes from different population members (crossover) to produce offspring or by altering existing members of the population (mutation). A simulation of ‘natural selection’ then takes place by first evaluating the quality of each solution and then selecting the fittest ones to survive to the next generation.

Memetic algorithm introduced by Moscato and Norman in 1992 ([14]) is an improved variant of genetic algorithm. It takes the concept of evolution as in genetic algorithm. However, while genetic algorithm is based on biological evolution, memetic algorithm is based on cultural evolution or idea evolution. In the evolution of ideas, an idea may be improved not only by recombination from others, but also by adaptation from itself. A meme, a unit of information in memetic algorithm can be improved by the individual holding it before it is passed on. A meme differs from a gene in that as it is passed between individuals, each individual adapts the meme as it sees best whereas genes are passed unchanged. A basic memetic algorithm is then an evolution algorithm incorporated with some local search technique.

In this work, the local search technique is hill-climbing and the evolutionary operators are only mutation operators (light mutation or heavy mutation). In genetic algorithm, while the mutation creates new genes for the population, the crossover operator orients seeking the best solution from the genes in the population. In memetic algorithm, this orientation is achieved by local search. Local search reduces the search space and reaches to high quality solution faster. The outline of memetic algorithm is given in Figure 1.

create initial population
repeat
1. take each individual in turn:
   Choose a mutation method (light or heavy mutation)
   Apply mutation operator to chosen individual
   Apply hill-climbing to individual just created.
   Insert it into the population.
2. select a half of them to reduce the population to its original size.
until termination condition is true

Fig. 1. Memetic Algorithm

A. Solution Representation

In this work, we use direction representation. Each solution in the population is represented as a number of memes. Each meme contains information on what exams are scheduled in which rooms for a particular period. Besides fixed number of the periods, a solution consists of a further meme that used to hold exams which could not be scheduled in the predefined periods. Figure 2 shows an example of an coded solution, where Ri is room number i and ei is exam number i.

B. Initial Population Generation

The initial population is generated using a graph-coloring algorithm in which exams are represented as vertices. Each member in the initial population has to satisfy all the hard constraints. The procedure to generate a member of the initial population is given in the Figure 3.
Generate a random ordering of vertices in graph G
Perform graph-coloring with the following steps:
1. Let \( j:=1 \), \( H:=\text{G} \);
2. Let \( v_j \) be a vertex of maximal degree in \( H \);
3. Find all the vertices \( x \) that \( x \) and \( v_j \) share common vertex \( y \), but \( x \) and \( v_j \) are not adjacent (i.e. \( x \) is adjacent to \( y \), \( y \) is adjacent to \( v_j \), but \( x \) is not adjacent to \( v_j \)). Among these vertices \( x \), find a vertex \( v \) that has maximum number of common vertices with \( v_j \). If no such vertex exists, choose a vertex of maximal degree and non-adjacent to \( v_j \).
4. Merge \( v_j \) and \( v \) into \( v_j \) and repeat from step 3 until no choice is possible then start again step 2 with \( H:=H – \{v_j\}, j:=j+1 \).
5. When there is no vertex left, stop and color all vertices merged into \( v_i \) (1 ≤ \( i \) ≤ \( j \)) with color \( i \). This will be a \( j \)-coloring of the graph \( G \).

Fig. 3. Algorithm for creating initial population.

C. Evolutionary Operators

In this work, we use only mutation operators, and ignore crossover operator. In standard genetic algorithm, the crossover aims at searching the best individuals through permuting the genes in the population. In memetic algorithm, hill climbing can achieve this task better. So only mutation operators are required to create new memes.

There are two kinds of mutation: light mutation and heavy mutation. Light mutation aims at directing the search away from local optima, whereas heavy mutation aims at speeding up the finding of higher quality solutions.

Light Mutation
In light mutation, we remove randomly a number of exams in the timetable. Then all the removed exams are rescheduled to the legal periods, therefore it creates many different memes to escape local optima. Light mutation algorithm is described as follows:

1. Initialize an empty unscheduled list.
2. Choose a random number \( \text{periodNum} \) in [1..n] (n: the number of periods)

Heavy Mutation
In the heavy mutation algorithm, all periods with low penalty scores will not be disrupted. Otherwise, one or more periods with high penalty scores will be disrupted with a probability \( P \). Instead of removing exam by exam as in light mutation, we disrupt the whole period in heavy mutation. By heavy mutation, many new memes are created and finding a good solution can be done faster. Heavy mutation can be described as follows.

1. Initialize an empty unscheduled list.
2. Move all exams in unscheduled period to the unscheduled list.
3. Let \( \text{AvgPenalty} \) be average penalty of the population.
4. For each period \( i \) in the timetable in order do
   - Calculate the penalty \( P_i \) arising from all exams in period \( i \), assuming all periods have not been disrupted.
   - If this penalty \( P_i \) is lower than the average penalty in the population \( \text{AvgPenalty} \), then give the period a probability of being disrupted:
     \[
     P = \frac{P_i}{2 \times \text{AvgPenalty}}
     \]

5. Take each exam in the unscheduled list according to decreasing order of the number of conflict exams, reschedule it to the first legal period such that no hard constraints are violated.
6. Move all remaining exams in the unscheduled list to unscheduled period.

In this algorithm, we use the graph coloring heuristic which bases on the orders of the exams to reschedule them. The order of an exam is the number of conflict-exams that this exam has. Two exams are conflicting if they have common students. Moreover, because all exams in unscheduled period are moved to the unscheduled list, the exams with higher order (i.e. having many conflict exams) get higher chance to be rescheduled. Therefore, the number of exams in the unscheduled period will decrease.

Between the two mutation operators, we have to select one of them at each iteration of the algorithm. This selection can
be done by using heuristic or randomness. In this work, it is done by heuristic. At the early stage, heavy mutation is selected with the same probability as light mutation to diversify the population. At the later stage, when the population starts to converge, we reduce the probability of heavy mutation.

After mutation phase, there are many new memes and individuals in the population. All these individuals need be improved to become the best individuals with their current memes. For this purpose, hill climbing operator is applied to all of them.

D. Hill Climbing

Hill climbing is one of the local search techniques that direct the search toward the local optima. In hill climbing algorithm, the state changes to the best state of all next states it can move to. This process is iterated until reaching local optimum.

In this work, we investigate two different hill-climbing methods: penalty-based hill climbing and constraint type-based hill climbing. Experiments show that the second hill-climbing method, also called hierarchical hill climbing by Alkan and Ozcan ([2]), yields a better performance. To compare two methods, the first one is more general and easy to apply, but we can not include heuristics to direct the uphill climbing since we do not take the characteristics of constraints into account. With the second method, we can include heuristics to direct the climbing and the priority of each constraint type can be discriminated in setting the order of satisfying them.

Penalty-based Hill Climbing

In this approach, each exam in every period is rescheduled to the other one which incurs the least penalty. After this stage, all the exams in unscheduled period are also rescheduled using graph coloring heuristic. The hill climbing step can be summarized as:

1. Take each period in order
   - Take each exam e in this period in order
     - For each period, calculate the penalty that would arise from scheduling exam e to this period without hard constraint violations.
     - Reschedule e in the period with the least penalty.
   - Reschedule e in the period with the least penalty.
2. Try to schedule exams in unscheduled period according to a decreasing order of the number of conflict exams.

Constraint type-based Hill-Climbing

The idea behind this hill-climbing approach is to create a hill climbing method for each type of constraint and combine them under a single hill climbing method. The purpose of this approach is to give higher chance to an operator of the related constraint type causing more violations. There are three kinds of hill climbing that deal with three kinds of constraint violations. Unscheduled hill climbing (UHC) deals with hard constraint violations. Spread hill climbing (SHC) deals with exam-spread-constraint violations. Other hill climbing (OHC) deals with all other soft constraint violations.

The UHC tries to schedule all unscheduled exams. To schedule an exam e, it tries to find a feasible period for e without violating any hard constraints. If possible, it assigns e to that period. Otherwise, it constructs a list L that consists of all the exams conflicting with e. Then, all the exams in L are rescheduled to other periods in order to find a feasible period in which e can be rescheduled. If this is still not successful, it turns to the last heuristic. It seeks the period that contains only one exam e’ belonging to L, removes e’ out of the period and reschedules e to that period.

The SHC deals with any violations of the exam-spread constraint. All exams that violate this constraint are found and rearranged. The priority to rearrange them is based on the number of students that suffer this constraint violation.

The OHC deals with all other soft constraint violations. This hill climbing is based on penalty. That means with each period and each exam e in this period, OHC reschedules it to other period with the least penalty.

E. Fitness Function

An optimum timetable is the one satisfying all the constraints. The fitness function is computed basing on the penalty. In this work, fitness is described as:

\[ \text{fitness} = \frac{M - \text{penalty}}{M} \]

where M is the maximum penalty of the population.

The penalty F of a timetable is based on the constraint violations of the timetable. A penalty function of a timetable consists of many components. Each component reflects the influence of a certain constraint type.

\[ F = \sum_{i=1}^{k} w_i F_i \]

where
- k: the number of constraint types
- w_i: the importance weight of constraint i
- F_i: the component reflects constraint i

There are five components: penalty quantities for unscheduled courses, for exam-spread, for room restrictions, for staff restrictions, and for room utilization. They are described as follows.

Penalty function for unscheduled courses

\[ F_1 = \frac{N_e}{N_e} \cdot N_u \]

where
- N_e: the number of students registered
- N_u: the number of exams to schedule
- N_u: the number of unscheduled exams

Penalty function for exam spread
Let $t_i$, $t_j$ be the periods assigned to exam $i$ and exam $j$. The penalty score for a given timetable in term of student restrictions, is given as follows:

$$F_2 = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \text{conflict}(i, j) \cdot \alpha_{ij}$$

where

- $n$: the number of exams to schedule
- $\text{conflict}(i,j)$: the number of students taking 2 exams, $i$ and $j$.

$\alpha_{ij}$ is defined as

$$\alpha_{ij} = \begin{cases} 3 \cdot |t_i - t_j| & \text{if } 0 \leq |t_i - t_j| \leq 3 \\ 0 & \text{otherwise} \end{cases}$$

Penalty function for room restrictions

Assume all subgroup of exam $i$ are allocated to $r$ buildings $L_{i1}, L_{i2}, \ldots, L_{ir}$. Then, the penalty in term of this constraint can be described as:

$$F_4 = \sum_{i=1}^{n} \sum_{j=1}^{r} \sum_{h=1}^{r} \sum_{k=j+1}^{r} d(L_{ih}, L_{ij}) / M$$

where

- $n$: the number of exams
- $d(L_{ih}, L_{ij})$: the distance of $L_{ih}$ and $L_{ij}$
- $M$: the minimum distance of two buildings.

Penalty function for staff restrictions

With every faculty, we calculate the average number of exams that are expected per period.

$$\overline{\text{Num}_k} = \frac{N_k}{T}$$

where

- $N_k$: the number of exams belonging to faculty $k$
- $T$: the number of periods.

The penalty in term of staff restrictions can be calculated as follows. If $\text{Num}_{ki} > \overline{\text{Num}}_{ki}$, we have

$$F_3 = \sum_{k=1}^{N} \sum_{i=1}^{T} |\text{Num}_{ki} - \overline{\text{Num}}_{ki}| \cdot \frac{N_s}{N_e}$$

where

- $N$: the number of faculties.
- $T$: the number of periods.
- $\text{Num}_{ki}$: the exam number of the faculty $k$ in the period $i$
- $N_s$: the number of students registered
- $N_e$: the number of exams to schedule,

otherwise, $F3 = 0$.

Penalty function for room utilization

The penalty in term of room utilization can be described as follow:

$$F_5 = \sum_{i=1}^{T} \sum_{j=1}^{R} \text{Space}(i, j)$$

where

- $T$: the number of periods
- $R$: the number of rooms
- $\text{Space}(i,j)$: the number spaces of room $j$.

During scheduling, any exam violating hard constraints is put into the unscheduled period. So penalty score for unscheduled courses take hard constraints into account and therefore its weight should be much higher than the weights of the others.

F. Selection Operator

Mutation operator doubles the number of individuals in population. So the selection operator will choose a half of them to reduce population size to its original size for next generation. This operator is based on classical roulette selection which involves creating a roulette wheel where each individual is assigned a section which is proportional to its fitness. The wheel is then spun a number of times equal to the population size to select each individual member of the new population.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

This algorithm was implemented with MS Visual C++ 6.0 and experimented on a Pentium IV 1.7 GHz PC. This timetabling system has been tested on the real data from HoChiMinh City University of Technology. This test data set comes from 64012 enrollments, 324 exams and 1820 exam groups.

It takes about 3 seconds to create initial population. With population size of 30, it takes about 6 seconds for each evaluation when penalty-based hill climbing is used. With the same population size, it takes about 3.5 seconds for each evaluation (or about 58 minutes for 1000 evaluations) when constraint type-based hill climbing is used. Therefore, in practice, we employ constraint type-based hill climbing in our memetic algorithm.

Starting from the initial population, the memetic algorithm makes improvements until a number of tries have elapsed in which no improvements could be found. A new timetable is then generated and the process repeated, keeping the best time-table found throughout the process.

On this real data set, it requires 35 periods to schedule all exams. With our exam timetabling system using memetic algorithm, the experimental results are described as in Table 1.

<table>
<thead>
<tr>
<th>TABLE 1. EXPERIMENTAL RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of Periods</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>33</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>31</td>
</tr>
</tbody>
</table>
With 33 periods, and 1000 evaluations, the solution improvement process can be illustrated as in Figure 4. The figure shows that at first, the penalty point decreases quickly, then it converges to the best solution and the average penalty always changes, that means it explores approximately the whole search space.

![Figure 4: The average penalty and the lowest penalty after 1000 evaluations.](image)

Compared with the previous hybrid approach which combines constraint programming and simulated annealing ([3]), the memetic approach takes more run time, but yields a quite better solution for the same data sets but with a larger set of constraints. In Table 2, the running times of the two methods can be compared. It is difficult to compare a genetic-algorithm-based approach to some simulated annealing algorithm in solving timetabling problems, however, according to the work done by Corlioni et al. ([9]), genetic-algorithm-based approach always takes more run time but is more robust and produces better timetables than simulated annealing.

<table>
<thead>
<tr>
<th>TABLE 2. MEMETIC VERSUS SIMULATED ANNEALING</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA with 5000 SA moves (All constraints except C6)</td>
</tr>
<tr>
<td>MA with 1000 evaluations (All constraints)</td>
</tr>
</tbody>
</table>

In comparison to the performance of the memetic algorithm for exam scheduling given by Burke et al. in ([7]), our system seems to run faster. This comes from the fact that our algorithm employs a more intelligent local search operator, called constraint type-based hill-climbing, that allows us to incorporate some form of domain knowledge in order to direct the hill climbing step more effectively.

V. CONCLUSION

This algorithm offers an efficient approach to evolutionary timetabling. Experiments carried out on real data set demonstrated that genetic search combined with hill climbing greatly increases the speed at which better solutions are found above the evolutionary operators alone. The local search step in the memetic algorithm serves two purposes:

- when the population is diverse it acts like a local search procedure and
- when the population converges its goal is to diversify the search.

Our main conclusion from this work is that we can solve a very difficult multi-objective scheduling problem with a memetic algorithm but we have to invent suitable evolutionary operators and determine how to combine them with local search operator. Besides, we also notice that constraint type-based hill climbing helps to speed up the memetic algorithm a lot since it can take the characteristics of constraints into account during local search step.

Typical run times of our memetic algorithm equate to no more than one hour and a half run on Pentium IV PC. The solutions produced by the memetic algorithm are better than the simulated annealing solution and they are relatively unaffected by initialization and parameter changes. We believe that memetic algorithms are particularly robust to handle the variety of difficult timetabling problems in the real world.

In a future work we plan to apply dynamic penalty functions to reduce the run time of the algorithm. Another direction of future work is to apply memetic algorithm to some other types of constraint satisfaction problems such as high school course timetabling, nurse rostering or university lecture timetabling.

REFERENCES

Timetabling”, In: Evolutionary Computing, T. Fogarty, Ed., 1994, 250-
263.

Timetabling”, In: E. Burke & P. Ross (Eds.) Practice and Theory of
Automated Timetabling, First International Conference, Edinburgh,
U.K., August/September 1995, Selected Papers, Lecture Notes in

Timetabling”, The Practice and Theory of Automated Timetabling
PATAT 1997, Selected Papers, E. Burke and M. Carter, Eds., LNCS

Examination Timetabling”, The Practice and Theory of Automated
Timetabling PATAT 2000, Selected Papers, E. Burke and W. Erben,

Salesman Problem Implementation of a Computational Ecology for
Combinatorial Optimization on Message-Passing Systems”, Parallel

Constraint Logic Programming”, Proceedings of 3rd Int. Conf. on the
Practice and Theory of Automated Timetabling PATAT 2000, Konstanz,
Germany, 181-183.

Simulated Annealing Based Timetabling Systems”, The Practice and
Theory of Automated Timetabling, 1st Int. Conference: Selected Papers,
(E. Burke & P. Ross, Eds.), Edinburg, U.K., August/September, 1995,

with Long-term Memory”, The Practice and Theory of Automated
Timetabling PATAT 2000, Selected Papers, E. Burke and W. Erben,