Timetabling: A State-of-the-Art Evolutionary Approach

M. Doulaty, M. R. Feizi Derakhshi, and M. Abdi

***Abstract*—Timetabling is the task of creating a schedule while satisfying some constraints. This problem is an NP-Complete problem, so solving it needs some heuristics. There are many types of timetabling; we mainly focused on university course timetabling. There have been many efforts in literature, but most of them have used some limiting assumptions that cause their approach to be unusable in real situations. We've used an evolutionary approach based on genetic algorithm to solve this problem in real situation and in a reasonable amount of time. We've used real data from our department in the university and our approach could solve the problem in about 15 minutes, while manually timetabling may take three days.**

***Index Terms*—Timetabling, university timetabling, genetic algorithm.**

I. Introduction

University timetabling is the task of creating a timetable for courses and satisfying some constraints. There are basically two types of constraints, soft constraints and hard constraints. Soft constraints are those if we violate them in scheduling, the output is still valid, but hard constraints are those which if we violate them, the schedule is no longer valid. This problem is believed to be a NP-Complete problem [1], [2], thus an optimal solution cannot be found in a reasonable amount of time. That is why heuristics should be used. Many efforts have been made in the literature for solving timetabling problems. These efforts use genetic algorithms, Tabu search, simulated annealing, ant colony and constraint satisfaction. The main goal of timetabling is to create a feasible and valid timetable concerning the needs of both professors and students.

The search space of a timetabling problem is too huge, many solutions exist in the search space and most of them are not feasible. Feasible solutions here mean those which do not violate hard constraints. We need to choose the most appropriate one from feasible solutions. Most appropriate ones here mean those which do not violate soft constraints too much.

There is a great variety of constraints assumptions in different works, but most of them [3], [4] have under estimated the constraints. We can see some solutions which do not violate assumed soft constraints, but they are not good from a user's point of view. This case happens in most of the existing works (including [5]-[11]) [1], [3], [4], [12]. The problem arises from the matter that there is no widely accepted soft constraints between academical institutes.

Furthermore, most of the existing approaches only have a minimal set of soft constraints (this is mainly caused by the fact that they wanted to have an easy to implement and quick solution [3], [12]. For those which does not have a minimal set of soft constraints and used genetic approach, this problem is mainly related to their Fitness function [3]. But our approach is quite different from other works (as will be shown in next sections). We've assumed a large set of constraints, both hard and soft. This causes our method to be some how complicated, but the results are very promising. The wide variety of constraints in our approach makes it easy to be used among different universities.

II. Timetabling Problem Definition

Timetabling can both be used in schools and universities. Creating a timetable for school is easier than creating a timetable for university. A survey which has been done in Britain [3], concluded that manually creating a good timetable for a school takes at most one day, while manually creating a timetable for university may take up to four days. After creating a timetable, if some small changes are needed, re-creating the timetable is still a hard and time consuming task. Timetabling not only can be used for course, but also it can be used for exam scheduling as well. Up to here we used time timetabling for scheduling courses, but in general, according to Wikipedia, "timetabling is an organized list, usually set out in tabular form, providing information about a series of arranged events: in particular, the time at which it is planned these events will take place". General use of timetable may include the following ones.

1) School timetable, a table for coordinating these four elements.
2) University timetable, quite different from school timetables (discussed earlier).
3) Workplace schedule, a list of employees who are working on any given day, week or month in a workplace.
4) Airline timetable, booklets that many airlines worldwide use to inform passengers of several different things, such as schedules, fleet, security, in-flight entertainment, food menu, restriction and phone contact information.
5) Public transport timetable, a listing of the times that public transport services arrive and depart specified locations.
6) Sport events timetable, a listing of the times that different sport teams will play with each other as a part of a league or series of matches.

In this paper we mainly focused on university timetabling. However, with some minor changes, our approach can be used for any of the other timetabling tasks.

Manuscript received September 10, 2012; revised April 1, 2013.

Mortaza Doulaty is with the Department of Computer Science, University of Sheffield, Sheffield, UK (e-mail: m.doulaty@dcs.shef.ac.uk).

Mohammad-Reza Feizi-Derakhshi is with the Department of Computer Science, University of Tabriz, Tabriz, Iran (e-mail: mfeizi@tabrizu.ac.ir).

Mehrdad Abdi is with the Amirkabir University of Tehran, Tehran, Iran (e-mail: m.abdi@aut.ac.ir).

A. Constraints

As mentioned above, there are two types of constraints, hard constraints and soft constraints.

B. Hard Constraints

Keeping in mind all of related works, a set of hard constraints can be concluded as the following list.

1) There shouldn't be more than one scheduled course to be held in one class at the same time (class interference).
2) A professor cannot have two different courses at the same time (professor interference).
3) Some facilities are required by some courses, these critical facilities should not be violated. E.g. some courses needs to take place in some special classes, such as labs.
4) Groups of students which are in the same semester should not have two courses simultaneously.
5) Classes have a limited count of seats.
6) Professors cannot attend classes at some predefined times.
7) Classes should not be used in some pre-defined times.

There is not a global agreement about hard constraints in literature, while some works assume some constraint to be hard constraint, others may assume it soft. But the above list is assumed to be the list of hard constraints in our approach.

C. Soft Constraints

A comprehensive list of soft constraints can be concluded as the following list.

1) Minimizing the time-gaps in each professors timetable.
2) Minimizing the time-gaps in student groups timetable.
3) Minimizing or maximizing the count of days students attend university. (This policy differs between universities.)
4) Minimizing or maximizing the count of days professors attend university. (This policy differs between universities.)
5) Maximum and minimum hours a professor can teach different courses in a single day.
6) Maximum and minimum hours that a student can have courses in a single day.
7) Some professors prefer not to have classes in some certain hours (if possible and not obligatory).
8) Some courses prefer to have some facilities in classes (such as video/data projectors) if possible (and not obligatory).

III. Related Work

Creating a feasible timetable can be done in a number of ways which can be grouped as following [1], [3], [4], [12].

1) Graph coloring
2) Clustering methods
3) Constraint based methods (e. g. integer programming)
4) Meta heuristic methods (e. g. genetic algorithms, simulated annealing, Tabu search and ants colony)

Most of the approaches have limited the constraints to gain a quick and good result. Although their approach is fast and easy to implement, but cannot be used is real situation where there are too many constraints that in some cases are very different from one university to another. Some of the approaches are discussed below.

A. Genetic Algorithm

In [6], [7], [13], [14] each day of week is divided into six 90 minutes gaps. Each week has five working days. The day/time gaps are enumerated as:

mon_1, mon_2, …, mon_6, tue_1, tue_2, …, fri_6

Classes are defined by symbol C and $\forall c \in C$ maximum number of students is defined. Set T is the set of all professors and set R is the set of all classrooms. L denotes the set of courses and $\forall l \in L$, course type, professor and class type is defined. To make things easier, only a few hard and soft constraints are defined in most of the approaches [6], [7], [13], [14].

B. Chromosome Representation

In these approaches [6], [7], [13], [14] each chromosome is represented as:

$$f: C \times T \times L \times R \times P \rightarrow \{0, 1\}$$

$f(c, t, l, r, p) = 1$ if and only if for class c, professor t and course l, class r and date/time gap p is a feasible schedule. Each gene in this chromosome can be an element of a 5D matrix than can hold values 0 and 1 (false and true respectively).

Almost all of other genetic based approaches used the same chromosome representation while having only a small subset of constraints [6], [7], [13], [14], [15].

C. Initial Population

Generating initial population is done by selecting an random course from course list, then trying to assign it to a class in a random day/time gap (the professor is already defined in the course) [6], [7], [13], [14]. In this case only hard constraint is checked and all of the soft constraints are ignored.

In different works, different initial population size is defined. But most of them are between 10 and 40 [12].

D. Fitness Function

Evaluation of a chromosome's fitness value is done by using something similar to this function [6], [7], [13], [14]:

$$eval(f) = \frac{1}{1 + cost(f)}$$

where *cost(f)* is the values of violating soft constraints and is calculated as:

$$cost(f) = \sum_{i=1}^{t} n_i(f) \times w_i$$

where t is the count of soft constraints, $n_i(f)$ is the penalty of violating soft constraint i and w_i is its weight.

E. Mutation

For mutation, a timetable named f is selected, an integer number m and a set $\pi \subseteq P$ containing m day/time gap is selected randomly and set L (f; π) is created. Mutated timetable f' is created with assigning new day/time gaps or classrooms to π cutout.

F. Cross-Over

Two timetables f and g are selected as parents and their child is created as: each course, day/time and class is selected from its parent, $\forall c \in C$ there exists a set $\pi_c \subseteq P$ from day/time gaps, for which its timetable is described as below:

$$h(c,l,r,p) = \begin{cases} f(c,l,r,p) & iif\ p \in \pi_c \\ g(c,l,r,p) & else \end{cases}$$

G. Selection

Tournament and elitism are preferred over other selection methods. Elitism ensures that the best timetable will survive in each generation.

IV. Proposed Approach

As mentioned earlier, other approaches have some limiting assumptions that makes those approaches unusable in real situations where there are too many different (and sometimes opposite) constraints being held among different universities. So a good approach which satisfies different needs of different universities does not exist. Our aim is to create a state-of-the-art method for timetabling university courses, having a robust and flexible algorithm which satisfies different (and sometimes opposite) needs of different universities.

A. Chromose Representation

Our chromosome has a simple implementation. Each gene is consisted of four fields: Course ID which identifies the course and its professor is the main member of the gene; the other three remaining fields are class ID, day and time ID. These three fields are filled after timetabling task is done. To clarify things, we divided each day of week into different time gaps, which has a day ID and a time ID.

B. Initial Population

Our approach for creating the initial population includes these steps. First we order the list of the professors, busiest professors at top of the list. Then we start by selecting the busiest professor, choose courses which should be offered by him. For each course which should be offered by him, we select a list of classes that meets the facilities required by that course. Then we randomly select a day and time for this course. After selecting class, day and time, we check the hard constraints so that we do not violate them. If this gene is okay, we continue with the next course of the current professor and continue this task until all of the professors in the list are visited. If, at any stage of this process, we find out that we cannot continue, we simply discard and continue from the beginning. Using this method, we create an initial population of size 20 which are all feasible timetables.

C. Cross Over

For cross-over we used a PMX operator. First we select two chromosomes as parents, then select two random cross points, genes between two cross points remain the same in each chromosome, others which does not create an invalid chromo- some are swapped between chromosomes, and those which causes the chromosome to be invalid, are not swapped. Up to here we may have two complete chromosomes or may be some chromosomes which all of the genes are not present. For missing genes, we fill the values of class ID, day and time ID using the method discusses in previous section (busiest professor first). After doing this step, we have two off-springs. We add them to our population.

D. Mutation

A chromosome is randomly selected as the candidate for mutation. Then one of its gene's is selected randomly and the value of class ID, day and time ID is cleared for this gene. Then we re-schedule this gene (course).

E. Fitness Function

Our fitness function is as below:

$$f_{min}(C) = \sum_{i=1}^{n} W_i \times V_i(C)$$

where n is the number of soft constraints and W_i is some constant weights that serve as a kind of penalty for violating soft constraints and their values are tuned after different experiments. V_i () is the function for evaluating the value of violating penalty from soft constraint i.

F. Selection

We used tournament and elitism as selection method. Elitism ensures that each elite chromo- some in different generations will survive in the next generation. The size of our population does not change during the evaluation process. As said earlier, during each step of the evaluation process, all of the chromosomes are valid and feasible ones. If, for any reason, during each of the steps described above, a chromosome becomes invalid, the process is interrupted and restarted from the beginning. If this cycle continues more than three times for an individual chromosome, the process is restarted from the very beginning. This has a very great impact on our approach as will be shown in the results section.

V. Experiments and Results

Working with real data in real situation was our goal. So we used the real data from the Department of Computer Science, Faculty of Mathematical Sciences, University of Tabriz. We've entered the raw data for two semesters (courses being offered with their corresponding professors). Our implementation which is developed in C# used a forge NET framework for representing chromosomes, generations and processing the evolution. We ran our tests on a dual core 2 GHz CPU with 2 GB of RAM.

The total number of courses to be scheduled was 52. Manually timetabling these courses takes about three days. When we applied our evolutionary method for solving this problem, the results are as shown in Table I. We ran our tests ten times and have the average results written. This causes to have a minimal error rate.

The results show that an initial population of count 20 is enough to create a reasonable timetable. About generations count, 1000 seems to be just fine. The run time for creating 1000 generations from initial population of count 20 is about 17 minutes. Which is very good, compared to manually timetabling which normally takes three days.

TABLE I: RESULTS

Pop. Count	Generations	Time
20	10	10 Sec.
30	10	16 Sec.
40	10	21 Sec.
20	100	101 Sec.
30	100	164 Sec.
40	100	220 Sec.
20	1000	17 Min.
30	1000	27 Min.
40	1000	36 Min.

TABLE II: COMPARISON WITH OTHER APPROACHES

Method	Soft Const. C.	Hard Const. C.
[7]	5	3
[14]	7	4
[13]	3	4
[6]	11	4
Our	19	7

VI. CONCLUSION AND FUTURE WORK

As is shown in Table II, an evolutionary method for timetabling university courses has been proposed. Other approaches used different techniques, varying from Tabu search, genetic algorithms and ant colony to constraint satisfaction. We've based our approach on evolutionary computing. We've created a genetic algorithm for solving timetabling problem. We've mainly focused on university timetabling; however, our approach can be used in other timetabling problems as well. Unlike other approaches appeared in literature, we've used a wide variety of both hard and soft constraints. From the very beginning, we kept in our minds that this method will be used in real situation serving different needs of different universities. So unlike others, we did not have any limiting assumptions.

We've used C# and together with Forge .NET framework to develop our application. We've used real data of our department in university to test the method and see how well it is working. 17 minutes for creating 1000 generations of initial population count 20 is a very promising result (when compared to manually timetabling which takes about three days).

Our future work will be parallelizing this approach to benefit from other cores of multi-core processors. This will certainly shorten the run-time of this method. Parallel genetic algorithms will be used in our next implementations. Also we're working to have other termination criteria, maybe a compound method concerning generations count and something else.

ACKNOWLEDGEMENT

The authors would like to express their cordial thanks to Pazhoohesh Afzar Farda Co. (PAFCO) for providing real data and test bed.

REFERENCES

[1] E. Burke, K. Jackson, J. Kingston, and R. Weare, "Automated university timetabling: the state of the art," *The Computer Journal*, vol. 40, pp. 565-571, 1997.
[2] T. Cooper and J. Kingston, "The complexity of timetable construction problems," *Practice and Theory of Automated Timetabling*, vol. 1153, pp. 281-295, 2006.
[3] E. Burke, J. Newall, and R. Weare, "A memetic algorithm for university exam timetabling," *Practice and Theory of Automated Timetabling*, vol. 1153, pp. 241-250, 1996.
[4] K. Socha, M. Sampels, and M. Manfrin, "Ant algorithms for the university course timetabling problem with regard to the state-of-the-art," *Applications of Evolutionary Computing*, vol. 2611, pp. 334-345, 2003.
[5] S. Abdennadher and M. Marte, "University course timetabling using constraint handling rules," *Journal of Applied Artificial Intelligence, Special Issue on Constraint Handling Rules*, 2000.
[6] D. Abramson and J. Abela, "A parallel genetic algorithm for solving the school timetabling problem," in *Proc. 15th Australian Computer Science Conf.*, 2002.
[7] E. Burke, J. Newall, and R. Weare, "A memetic algorithm for university exam timetabling," *Practice and Theory of Automated Timetabling*, vol. 1153, pp. 241-250, 1996.
[8] E. Burke, S. Petrovic, and R. Qu, "Case-based heuristic selection for timetabling problems," *Journal of Scheduling*, vol. 9, pp. 115-132, 2006.
[9] E. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *European Journal of Operational Research*, vol. 176, pp. 177-192, 2007.
[10] E. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of Heuristics*, vol. 9, pp. 451-470, 2003.
[11] K. Socha, J. Knowles, and M. Sampels, "A max-min ant system for the university course timetabling problem," *Ant Algorithms*, 2002.
[12] E. Burke and S. Petrovic, "Recent research directions in automated timetabling," *European Journal of Operational Research*, vol. 140, pp. 266-280, 2002.
[13] D. Corne, P. Ross, and H. Fang, "Fast practical evolutionary timetabling," In *Evolutionary Computing*, vol. 865, pp. 250-263, 2004.
[14] E. Burke, R. Weare and D. Elliman, "A hybrid genetic algorithm for highly constrained timetabling problems," in *Proc. 6-th Int. Conf. Genetic Algorithms* (ICGA'5), 2005.
[15] L. Gaspero and A. Schaerf, "Tabu search techniques for examination timetabling," *Practice and Theory of Automated Timetabling III*, vol. 2079, pp. 104-117, 2001.

Mohammad-Reza Feizi-Derakhshi was born in 1975, in Tabriz, Iran. He received his B.Sc. (Hons) in Computer Engineering from University of Isfahan, Isfahan, Iran in 1997, and received his M.Sc. in Computer Engineering and Ph.D. in Computer Engineering, Artificial Intelligence from Iran University of Science and Technology, Tehran, in 2000 and 2007, respectively. He is currently an associate professor in University of Tabriz, Iran. His research interests include natural language processing, text and speech summarization, optimization algorithms and databases. He has published over 90 papers in difference conferences and journals.

Mortaza Doulaty was born in 1986, in Tabriz, Iran. He is a computer science Ph.D. student at the Department of Computer Science, University of Sheffield, UK. He received his B.Sc. (Hons) and M.Sc. (Hons) in Computer Science, Intelligent Systems from the University of Tabriz, Tabriz, Iran in 2009 and 2011, respectively. His broad research interests include machine learning, evolutionary computing and signal processing.

Mehrdad Abdi was born in 1987, in Ardabil, Iran. He received his B.Sc. degree in Information Technology Engineering from University of Tabriz, Tabriz, Iran in 2011 and currently he is a M.Sc. student in Information Security in Amirkabir University of Tehran (Tehran Polytechnic), Tehran, Iran. His research interests are in security focusing on software security,vulnerability analysis and Security testing, formal methods in security and also artificial intelligence and its applications in computing.