Annales UMCS Informatica AI 5 (2006) 133-143

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Hybrid artificial intelligence technique for solving large, highly constrained timetabling problems

Maciej Norberciak*

Insitiute of Applied Informatics, Wrocław University of Technology,
Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland

Abstract

Timetabling problems are often hard and time-consuming to solve. Profits from full automatization of this process can be invaluable. Although over the years many solutions have been proposed, most of the methods concern only one problem instance or class. This paper describes a possibly universal method for solving large, highly constrained timetabling problems from different areas. The solution is based on evolutionary algorithm's framework, with specialized genetic operators and penalty-based evaluation function, and uses hyper-heuristics to establish its operating parameters. The method has been used to solve three different timetabling problems, which are described in detail, along with some results of preliminary experiments.

1. Introduction

Timetabling problems are quite popular to be seen about and arouse interest of many researchers for more than thirty years. Their practical importance should not be underestimated – institutions involved in education, healthcare, transportation, sports, courts of law, production enterprises and many others devote considerable resources to establish effective plans of their actions. As the planning is often the most serious administrative task of institutions of such kind, over the years many approaches to its partial or complete automatization have been presented. Artificial Intelligence (AI) research community is quite active in the area of timetabling and scheduling and has developed a variety of approaches for solving such problems. They can be roughly divided into four types [1]:

– sequential methods – these methods order events using domain heuristics and then assign the events sequentially into valid time periods (also called timeslots) so that no events in the period are in conflict with each other; events are most often ordered in such a way that events that are most

* *E-mail address*: maciej.norberciak@pwr.wroc.pl

difficult to schedule are assigned into timeslots first (this course of action is called direct heuristic based on successive augmentation) [2];
- cluster methods – in these methods events are collected in clusters where any two events in a particular cluster do not conflict with each other; the main drawback of these approaches is that the clusters of events are formed and fixed at the beginning of the algorithm and that may result in a poor quality timetable [3];
- constraint-based approaches – in these methods a timetabling problem is modeled as a set of variables (i.e., events) that have a finite domain to which values (i.e., resources such as time periods) have to be assigned to satisfy a number of constraints; a number of rules is defined for assigning resources to events and when no rule is applicable to the current partial solution a backtracking is performed until a solution is found that satisfies all constraints; as the satisfaction of all constraints may not be possible, algorithms are generally allowed to break some constraints in a controlled manner in order to produce a complete timetable [4,5];
- meta-heuristic methods – variety of meta-heuristic approaches such as simulated annealing, tabu search, evolutionary algorithms and hybrid approaches have been investigated for timetabling; meta-heuristic methods begin with one or more initial solutions and employ search strategies to find optimal solution, trying to avoid local optima in the process [6-10].

Recently the application of case-based reasoning to timetabling has become increasingly popular [11-13]. Most approaches use heuristics because traditional combinatorial optimization methods often have a considerable computational cost. Although they can produce high quality solutions, they are not suitable for solving large, highly constrained problems.

Based on this enumeration one could conclude that AI-based automatic planning is at a quite mature level, all the problems solved in principle, and the research tends to steer towards making existing methods faster, more effective and giving better quality solutions for more complex and larger problems. However it must be pointed out that vast majority of the solutions concern only one, specific problem type (e.g. [14,15]) or some particular problem class [16,17] and to be employed in concrete, practical case, time and resources have to be devoted to adapt the solution to the specifics of the considered problem.

This paper presents an attempt to create a universal method, i.e. that capable of solving problems from different areas with minimum user-side interaction. Three different problems have been chosen for testing. The typical university course timetabling problem is one of the most popular and widely featured in research thus making the access to test data, both real and artificially generated, very easy. Similar, but more specific problem, with some different constraints, is timetabling on Faculty of Computer Science and Management of Wroclaw University of Technology. The last problem belongs to personnel scheduling

class – it is the problem of making monthly duties plan in the ward of one of the Polish hospitals. Although it is relatively small in size, a large number of different constraints makes it quite interesting.

2. Description of the problems

The typical timetabling problem consists in assigning a set of activities/actions/events (e.g. work shifts, duties, classes) to a set of resources (e.g. physicians, teachers, rooms) and time periods, fulfilling a set of constraints of various types. Constraints stem from both nature of timetabling problems (e.g. two events using the same resources cannot be planned at the same time) and specificity of the institution involved. In other words, timetabling (or planning) is a process of putting in a sequence or partial order a set of events to satisfy temporal and resource constraints required to achieve a certain goal, and is sometimes confused with scheduling, which is the process of assigning events to resources over time to fulfill certain performance constraints (however, many scientists consider scheduling as a special case of timetabling and vice versa) [9].

Timetable problems are subject to many constraints that are usually divided into two categories: “hard” and “soft”. Hard constraints are rigidly enforced and have to be satisfied for the timetable to be feasible, for example no resource can be demanded to be in more than one place at any time. Soft constraints are those that are desirable but not absolutely essential (e.g. an event may need to be scheduled in a particular time period or one event may need to be scheduled before/after the other). In real-world situations it is usually impossible to satisfy all soft constraints (often because they are mutually excluding).

The first problem considered is a typical university course timetabling problem (UCTP). It consists of a set of events (classes) to be scheduled in a certain number of timeslots, and a set of rooms with certain features and size which events can take place in. There is a defined set of students attending each event and the number of timeslots is 45 (5 days, 9 timeslots each). Test sets for this problem come from the International Timetabling Competition.

A feasible timetable is one in which all the events have been assigned a timeslot and a room, and the following hard constraints are satisfied:

- only one event is scheduled in each room at any timeslot,
- the room is big enough for all the attending students and satisfies all the features required by the event,
- no student attends more than one class at the same time.

There are also three soft constraints defined; they are broken if:

- a student has a class in the last slot of the day,
- a student has more than two classes in a row,
- a student has a single class on a day.

The second problem – timetabling on the Faculty of Computer Science and Management (FCSM) of Wrocław University of Technology has some constraints related to teachers' availability – each event had a teacher assigned and each teacher had a defined set of forbidden timeslots, and the set of students attending each event was undefined (only the number of students and the faculty they attend was known) and it has to be concluded from the other data. In this problem the number of timeslots is 35 (5 days, 7 timeslots each) and each event had a defined course (the class is a part of particular university course). Some test sets for this problem come from real data from FCSM and some have been artificially generated. In this problem, similarly to the previous one, feasible timetable is one in which all the events have been assigned a timeslot and a room, so the following hard constraints have to be satisfied:

- only one event is scheduled in each room at any timeslot,
- the room is big enough for all the attending students and satisfies all the features required by the event,
- no teacher carries on more than one class at the same time,
- no teacher carries on any class in timeslot which is forbidden for him; if a particular course has only one class assigned, no class with students from the same faculty is scheduled at the same timeslot with this course (this covers the obligatory courses which are usually taught for all the faculty's students).

A typical hospital department employs about a dozen or so physicians of various specialties. On each day one or more doctors has a duty. The number of doctors on duty may vary from day to day. A planning horizon (i.e. a period of time for which the problem must be solved) amounts one month (in both aforementioned course timetabling problems it is one week). If specialties of physicians in particular department are not homogenous (e.g. casualty ward employs surgeons and anesthesiologists) there are often requirements for specialty of doctors on duty. The following hard constraints are defined:

- all the timeslots (i.e. days) have a proper number of physicians of appropriate specialties assigned,
- no physician has a duty in two (or more) consecutive days,
- no physician has more than two duties in the week,
- at least one physician on each duty is able to perform duties single-handed (that means that a particular doctor has a certain degree of medical education and is experienced and responsible enough).

In order to consider and model fairness and job satisfaction issues, the following soft constraints are introduced:

- physicians have duties on preferred days of the month and, symmetrically, they have no duties assigned in timeslots they do not want to have duties;

– if more than one physician has a duty assigned in a particular time period social preferences are taken into consideration (doctors have duties with persons they like).

Duties on special days, like Eastern, Christmas, New Year's Eve, etc. are preassigned, according to schedules from previous years so no one has duty on the same holiday two years in a row.

3. Solution details

Evolutionary algorithms (EA) are considered to be a good general-purpose optimization tool due to their high flexibility accompanied by conceptual simplicity. Moreover, they have proven to be quite an effective tool for solving timetabling problems ([3,18]). Thus EA framework has been taken into consideration. Solution has been implemented solely by the author of the paper using Microsoft Visual C++.

3.1. Representation of the problem

In order to assure universality of the approach each solution (genotype) of particular problem's instance is represented directly each timeslot has a list of assigned events and each event – a list of resources. Genotype's length is constant for a particular problem – in the case of hospital duties genotype has a length of the number of physicians on duty times the number of timeslots; course timetable genotype's length amounts the number of timeslots times the number of rooms. The data needed to describe a particular problem class is abstract and unified for all the problem classes.

3.2. Genotype initialization strategies

In order to work EA has to be provided with the initial population of solutions. In most of the approaches either random or heuristics initialization is used. Random method has the least computational complexity and does not take into consideration problem's domain knowledge. Heuristic approaches have proven to be more effective though, i.e. the final solution tend to be found faster than in the case of random initialization. Nevertheless, heuristics always employs some kind of event sequencing strategy – the events are placed in the timetable in order of their decreasing "difficulty" to plan – i.e. the events that are the most difficult to schedule are allocated first. Either some kind of graph coloring or problem-specific heuristics is used. Reduction of the problem representation to a graph (as described in [19]), where events are represented by graph's vertices and if an edge between two vertices exists, the events represented by these vertices can not be scheduled together (at the same timeslot) has unfortunately many limitations (e.g. tells nothing about the reason why particular events can not be scheduled together, so it is impossible to tell the

difference between individual constraints). On the other hand, using problem-specific heuristics compromises flexibility of the solution. In the approach described in this paper, random initialization has been used as the point of reference to grade the other method – the peckish initialization method. In this approach for each timeslot k sets of events (and resources) are chosen at random; the set that breaks the least number of hard constraints is assigned to the timeslot. The number k is called greediness level – when k amounts 1 this method corresponds to random initialization; when k aspires to the number of combination of events and resources, the algorithm becomes greedy. The details of establishing k value are given in chapter 4.

3.3. Evaluation function

Penalty-based evaluation function was used. Penalty for genotype g amounts

$$f_g = \sum_{i=0}^{i<t} \sum_{j=0}^{j<c} w_j n_{ij}, \tag{1}$$

where t is the number of timeslots, c is the number of constraint types, w_j is the weight assigned to a particular constraint type, and n_{ij} is the factor determined by the penalization method. Four different methods have been considered:

a) the timeslot is penalized once for every type of constraint broken (i.e. n_{ij} amounts either 0 or 1);
b) the timeslot is penalized every time the particular type of constraint is broken;
c) as in b), but additionally for each subsequent constraint of the particular type broken, the penalty is doubled;
d) binary penalty – if the timeslot with events planned breaks no constraints, penalty for this timeslot amounts 0 (1 otherwise); this is an exception to (1), as no weights are used to determine value of the penalty.

Value of the evaluation (fitness) function for solution g is calculated by dividing the lowest penalty value in the population by penalty value for g

$$F_g = \frac{f_{\min}}{f_g}. \tag{2}$$

After generating the initial population the evolutionary algorithm begins to operate. Creation of population in subsequent generations (iterations) is achived by means of classical genetic roulette, as described in [20], but 20% of the population is always preserved from previous generation: 10% consist of best solutions (in terms of evaluation function described above) and the remaining 10% are the solutions that are most distant from the rest of the population, in order to preserve population diversity. The distance between two timetables can be measured in three ways:

– the number of events planned with the same resources in the same timeslot in both timetables,

- the number of pairs of events planned with the same resources in the same timeslot in both timetables; as described in [1] this method is favored because it allows to represent diversity as a single value average and did not have the drawback of method where absolute positions of the events in timetables are considered,
- search space coverage – how often the tuple <event, resources, timeslot> appears in the whole population.

The higher the score is, the smaller the distance between timetables.

Additionally, three methods of determining the weights have been proposed:

- unified weights (all weights amount 1),
- weak constraints have a weight value of one, strong constraints have a weight which amounts the number of weak constraints,
- automatic weight assignment – the details of this procedure are given in chapter 4.

3.4. Genetic operators

In the classic evolutionary algorithm in each iteration after the selection phase some specimens are exposed to genetic operators – mutation and crossover. The contents of operators' set and their operation depend strongly on both specifics of problem being solved and the approach chosen. In the approach described only mutation operators are used, because of high computational and conceptual complexity of recombination operator. Resources, events and timeslots can be mutated – that gives a set of three different types of mutation operators. In the "classic" EA mutation operator is "blind", i.e. changes the solution at random. This approach, however, has proven to be ineffective ([3]) so directed operators have been used. The place in genotype (tuple <event, resources, timeslot>), which breaks the most constraints (so it is most difficult to schedule) is selected to be mutated (if a few places are tied to be most difficult to schedule, one is chosen at random). The operators try to reschedule the event in such a way, that they would eliminate one particular type of conflict (broken constraint of particular type), caused by this event – k possible variants are examined, and the one, that breaks the least constraints of particular type is chosen (like in the peckish initialization algorithm). The order of eliminating the conflicts is established by means described in chapter 4. Application of any operator can spoil timetable in terms of both evaluation function and the number of constraints broken, but allows the algorithm to escape the local optima efficiently.

4. Hyper-heuristic

As can be seen in the description of method, there some parameters that have to be established for the solution to work. Such parameters are usually

established either arbitrarily (e.g. based on the domain knowledge) or experimentally. However, in the ‘‘knowledge poor’’ algorithms, designed to solve a range of problems such an approach proves impossible to be applied. It has recently been suggested ([21]), that hyper-heuristic methods can be used to cope with this setback. A hyper-heuristics denotes a heuristic that selects heuristics for a wide variety of problems, including timetabling. It differs from the widely used term “metaheuristic” in that the term meta-heuristic usually refers to a heuristics which manages one other heuristics for a particular problem. A hyper-heuristic can be thought of as a heuristic to choose or to create heuristics.

4.1. Automatic weight assignment

This procedure allows to establish the weights employed in calculating of the evaluation function based on how frequently constraints of a particular type are broken in randomly generated solution. A set of solutions is generated at random and the least frequently broken constraint is assigned a weight of one. The rest of the weights are established proportionally – the more frequent the constraint is broken, the higher the weight is.

4.2. Establishing greediness level

After assigning the weights (by means of any aforementioned method), greediness level of peckish initialization strategy is established. A dozen or so sets of solutions are generated with ascending greediness level (due to increasing computational complexity of the generation process, the highest greediness level considered has been arbitrarily set up to amount the number of timeslots in the particular solution). Then the average fitness for each set of solutions is calculated, along with average time in which the solutions were generated. The greediness which gives the best score (the shortest time and the highest average fitness) is chosen.

4.3. Second-level EA

To find out which methods of penalization, measuring the distance between solutions and weight assignment, along with the order of conflict elimination and greediness level of genetic operators constitute the most effective set of parameters (in terms of solution quality and time to reach feasible solution), the evolutionary algorithm is used. The genotypes represent the aforementioned parameters (greediness level is a natural number no greater than the number of timeslots, order of conflict elimination is an ordered sequence, the rest of attributes are nominal). In each iteration of the algorithm the solution of a particular problem is generated using the parameters given in each of the

genotypes (to avoid infinite operation the first-level algorithm ceases to operate after finding a feasible solution or after 1000 iterations). The 2nd level genotypes that did not give feasible solutions are scrapped, the rest are evaluated – the value of the evaluation function is the value of the binary penalty function for the best genotype in population. The best set of parameters is memorized, and then the genetic operators of mutation and crossover are applied to the solutions. Mutation (acting with the probability of 0.2) changes one of the parameters at random (in the case of the conflict elimination order, changes that order). Crossover (with the probability of 0.05) swaps random parts of two parameter sets (treating conflict elimination order as one parameter). The procedure stops after the fixed number of iterations or if no improvement has been made in two subsequent iterations.

5. Results of experiments

Some preliminary experiments have been conducted with the approach described in this paper. For all the experiments the second-level EA had a population size of 100, and the first-level of 500. The second-level EA ran for 1000 iterations.

The first task was to prove, that the method is able to find a feasible solution for all the test problems. Ten sets from the International Timetabling Competition (ITTC) has been used for the first problem, two real datasets for the second and one real and nine artificially generated for the third. The feasible solution has been found for all the test sets. The first problem proved most difficult, as it required a few hundred iterations of the second-level EA for feasible solution to be found. For the other two problems solutions were sometimes found without the help of the second-level algorithm but its operation improved the results considerably.

More extensive experiments were conducted on UCTP. The results are presented in Table 1. The fitness function values of the best solution achieved by the method described in this paper presented in the table have been recalculated to match the method used for evaluating solution in ITTC (all weights equal one, second penalization method). The results have been compared with the best and average results of ITTC competitors.

Table 1.

Dataset No.	1	2	3	4	5	6	7	8	9	10
ITTC best	45	25	65	115	77	6	12	29	17	61
ITTC average	137	87	150	289	248	143	145	129	123	153
Universal	158	103	156	399	336	146	125	110	154	153

As it can be seen, most of the results are worst than average of the ITC score. Nevertheless, it has to be emphasized, that the methods used in ITC were designed to perform only one task, and the parameters of their operation were chosen intentionally to perform that task in the most effective way possible. Moreover, almost all the methods used some form of local search to improve the solutions.

Conclusions and future work

The question whether the universal, "knowledge-poor" method is able to perform better or at least comparably well as the domain-specific one remains open. In terms of computational time it is probably not possible, as the general method searches the parameter space blindly. Preliminary results look appealing but more work is needed to improve the algorithm – employing local search seems especially promising. Nevertheless, universal methods will always have one distinctive advantage over the specialized ones – they do not need a laborious and time consuming process of redesigning and fine-tuning to fit specific needs.

Acknowledgements

The research was supported by Grant No. 3 T11C 031 27 from the State Committee for Scientific Research in Poland.

References

[1] Burke E.K., Petrovic S., *Recent research directions in automated timetabling*, European Journal of Operational Research, (2002) 140.
[2] Burke E.K., Newall J.P., Weare R.F., *A Simple Heuristically Guided Search for the Timetable Problem*, Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems, ICSC Academic Press, Nottingham, (1998)
[3] Myszkowski P., Norberciak M., *Evolutionary Algorithms for Timetable Problems*, Annales UMCS, Sectio Informatica, Lublin, I (2003).
[4] Do M.B., Kambhampati S., *Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP*, Artificial Intelligence, (2001) 132.
[5] Yakhno T., Tekin E., *Application of Constraint Hierarchy to Timetabling Problems*, Proceedings of EurAsia-ICT 002, Springer-Verlag, (2002).
[6] Colorni A., Dorigo M., Maniezzo V., *Genetic Algorithms and Highly Constrained Problems: the Time-Table Case*, Proceedings of the First International Workshop on Parallel Problem Solving from Nature, Lecture Notes in Computer Science, (1990) 496.
[7] Colorni A., Dorigo M., Maniezzo V., *Genetic Algorithms: a New Approach to the Time-Table Problem*, Lecture Notes in Computer Science – NATO ASI Series, Combinatorial Optimization, F82 (1990).
[8] Newall J.P., *Hybrid Methods for Automated Timetabling*, PhD Thesis, Department of Computer Science, University of Nottingham, (1999).
[9] Ross P., Corne D., *Comparing GA, SA and Stochastic Hillclimbing on Timetabling Problems. Evolutionary Computing*, AISB Workshop, Sheffield 1995, Selected Papers, ed. T. Fogarty, Springer-Verlag Lecture Notes in Computer Science, (1995) 993.

[10] Valouxis C., Housos E., *Hybrid optimization techniques for the workshift and rest assignment of nursing personnel*, Artificial Intelligence in Medicine, (2000) 20.
[11] Burke E.K., MacCarthy B., Petrovic S., Qu R., *Structured cases in case-based reasoning – re-using and adapting cases for time-tabling problems*, Knowledge-Based Systems, (2000) 13.
[12] Foulds L.R., Johnson D.G., *SlotManager: a microcomputer-based decision support system for university timetabling*, Decision Support Systems, (2000) 27.
[13] Lee S.-J., Wu C.-H., *CLXPERT: A Rule-Based Scheduling System*, Expert Systems With Applications, 9(2) (1995).
[14] Carter M.W., *A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo*, E. Burke, W. Erben (Eds.): Proceedings of PATAT 2000, Springer-Verlag, (2001).
[15] McCollum B., *The Implementation of a Central Timetabling System in a Large British Civic University*, E. Burke, M. Carter (Eds.): Proceedings of PATAT 1997, Springer-Verlag, (1998).
[16] Ross P., Hart E., Corne D., *Some Observations about GA-Based Exam Timetabling*, E. Burke, M. Carter (Eds.): Proceedings of PATAT 1997, Springer-Verlag, (1998).
[17] Socha K., Knowles J., Sampels M., *A MAX-MIN Ant System for the University Course Timetabling Problem*, M. Dorigo et al. (Eds.): Proceedings of ANTS 2002, Springer-Verlag, (2002).
[18] Norberciak M., *Feasible genotype initialization for evolutionary timetabling*, Proceedings of 9th International Conference on Soft Computing MENDEL 2003, Brno, (2003).
[19] Burke E.K., Elliman D.G., Weare R.F., *A University Timetabling System based on Graph Colouring and Constraint Manipulation*. Journal of Research on Computing in Education, 27(1) (1994).
[20] Michalewicz Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag, (1996).
[21] Burke E.K., Meisels A., Petrovic S., Qu R., *A Graph-Based Hyper Heuristic for Timetabling Problems*, Computer Science Technical Report No. NOTTCS-TR-2004-9, University of Nottingham, (2004).