Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 149 – 157

ISSN 1896-7094

Universal Method for Solving Timetabling Problems Based on Evolutionary Approach

Maciej Norberciak[1]
[1]Institute of Applied Informatics, Wroclaw University of Technology, Wroclaw, Poland
maciej.norberciak@pwr.wroc.pl

Abstract. Timetabling problems are often hard and time-consuming to solve. Most of the methods of solving them concern only one problem instance or class. This paper describes a universal method for solving large, highly constrained timetabling problems from different domains. The solution is based on evolutionary algorithm's framework and employs tabu search to speed up the solution finding process. Hyper-heuristics is used to establish operating parameters of the algorithm. The method has been used to solve three different timetabling problems with promising results of some preliminary experiments.

Keywords: evolutionary algorithms, timetabling

1 Introduction

Many institutions devote considerable resources to establish effective plans of their actions. Over the years many approaches to partial or complete automatization of such tasks have been presented. Most approaches use heuristics as traditional combinatorial optimization methods often come with a considerable computational cost. Although they can produce high quality solutions, they are not suitable for solving large, highly constrained problems. Among many different approaches (sequential [4] and cluster [12] methods, constraint-based approaches [19], applications of case-based reasoning [1]) a 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 [8, 12, 13, 15, 18].
Although AI-based automatic planning is at a quite mature level, it must be pointed out that vast majority of the methods concern only one, specific problem type (e.g. [5, 10]) 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 problems from different classes have been chosen for testing, so universality and flexibility of the method can be assured.

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 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) [15].

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 in order the timetable to be feasible. Soft constraints are those that are desirable but not absolutely essential.

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 International Timetabling Competition (ITTC) [9].

In UCTP 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:

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

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

1) a student has a class in the last slot of the day,
2) a student has more than two classes in a row,
3) a student has a single class on a day.

The second problem – timetabling on Faculty of Computer Science and Management of Wroclaw University of Technology – is similar to the first, but has additional constraints related to teachers and the set of students attending each event is undefined (only number of students and faculty they attend was known) and has to be concluded from 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 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 (apart from constraints 1) and 2) from UCTP):

- 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 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).

Third problem depicts a typical hospital department which employs about a dozen or so physicians of various specialties. On each day one or more doctors has a duty. 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. 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 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 don’t want to have duties;

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

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 ([12, 14]), thus EA framework has been chosen as a basis to build universal timetable problems solver. Term "EA" is used in this paper in its most generic meaning – it will indicate any population-based metaheuristic optimization algorithm that uses mechanisms inspired by biological evolution, such as reproduction and mutation.

3.1 Representation of the solutions

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

3.2 Evaluation function

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

$$f_g = \sum_{i=0}^{i<t} \sum_{j=0}^{j<c} w_j n_{ij} \qquad (1)$$

where t is the number of timeslots, c is the number of constraint types (in case of weak constraint with more than two preference levels, all preference level are considered to be separate constraints), w_j is the weight assigned to particular constraint type, and n_{ij} is the factor determined by penalization method. Four different methods have been considered:

- the timeslot is penalized once for every type of constraint broken (i.e. n_{ij} amounts either 0 or 1);
- the timeslot is penalized every time the particular type of constraint is broken;
- as in the first method, but additionally for each subsequent constraint of the particular type broken, the penalty is doubled;
- binary penalty – if the timeslot with events planed breaks no constraints, penalty for this timeslot amounts 0 (1 otherwise); this is an exception from (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.

$$F_g = \frac{f_{\min}}{f_g} \qquad (2)$$

After generating the initial population the evolutionary algorithm begins to operate. Creation of population in subsequent generations (iterations) is archived by means of classical genetic roulette, as described in [11], but

20% of the population is always preserved from previous generation. 10% consists of best solutions (in terms of evaluation function described above). 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:

- number of events planned with the same resources in the same timeslot in both timetables;
- number of pairs of events planned with the same resources in the same timeslot in both timetables; as described in [2] 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 – this procedure allows establishing the weights basing on how frequently constraints of 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).

3.3 Genotype initialization strategies

In most of the approaches either random or heuristics initialization is used to provide EA with initial population of solutions. 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. final solution tend to be found faster than in case of random initialization. Nevertheless, heuristics always employ some kind of event sequencing strategy – the events are placed in the timetable in order of their decrementing "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. In the approach described in this paper, random initialization has been used as point of reference to grade the other method – peckish initialization method [6]. 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. After assigning the weights (by means of any aforementioned method), greediness level 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 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 (shortest time and biggest average fitness) is chosen.

3.4 Genetic operators

In classic evolutionary algorithm in each iteration after selection some solutions are exposed to genetic operators – mutation and crossover. The contents of operator's set and their operation depend strongly on both specifics of problem being solved and approach chosen. In this particular case recombination operator would probably have

high computational and conceptual complexity. Even simple, one-point crossover operator would be quite complicated – after swapping parts of different timetables the integrity of resulting solution would have to be assured. That means checking if no event appears in the timetable twice and removing the copies accordingly. Thus, only mutation operators are used. Resources, events and timeslots can be mutated – that gives a set of three different types of mutation operators. In "classic" EA mutation operator is "blind", i.e. changes the solution at random. This approach however has proven to be ineffective ([12]). 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 event in such 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 peckish initialization algorithm – in typical approaches either only one random change is considered [7] or some form of local search is employed [13]; the method proposed is a simple, yet effective alternative to that).

3.5 Tabu search phase

Preliminary tests have shown an interesting phenomenon which appears about half-way through the solution finding process. If the average value of penalty function for a particular solution is being observed one can notice a steady, steep drop until population reaches a certain plateau, where the value oscillates slightly (about 1% up and down). That phenomenon proves that although directed genetic operators have been used, the algorithm still searches solution space somewhat blindly. Therefore it tends to be stuck in local optima and is only able to escape through random mutation. To speed up the solution finding process and avoid aforementioned oscillations, tabu search (TS) has been employed. If for fifty generations the average penalty for population deviates less than 20%, the genetic roulette is stopped and tabu search begins to operate. Tabu list length amounts $10*k$ (all the parameters for tabu search operation has been arbitrary established and fixed to avoid introducing new variables into the method). The algorithm operates as follows:

(1) Find place in solution (tuple <event, resources, timeslot>) which breaks the most constraints (if there are a few such places, choose one at random)
(2) Generate k solutions with event rescheduled with different resources and/or timeslot.
(3) Choose the solution which has lowest penalty score and is not on the tabu list, add it to tabu list and go to (1). The chosen solution is now the current one.

The tabu search algorithm operates for 50 iterations – after that evolutionary algorithm takes over again. The results of experiments with tabu search application are described in section 4.

3.6 Hyper-heuristic

As it can be seen in method's description, there are some parameters that have to be established in order the solution to work. Ordinarily such parameters are being established either arbitrarily (e.g. based on the domain knowledge) or experimentally. However, in the ''knowledge poor'' algorithms, designed to solve range of problems such approach proves impossible to be applied. It has recently been suggested ([3]), that hyper-heuristic methods can be used to cope with this setback. A hyper-heuristic 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 heuristic which manages one other heuristic for a particular problem. A hyper-heuristic can be thought of as a heuristic to choose or to create heuristics. Current applications of hyper-heuristic to timetabling tend to steer towards using metaheuristic to search for permutations of graph heuristics which are then used for constructing timetables [3]. In the method described in

this paper, metaheuristic is used to find the best parameters for another metaheuristic. Automatic weight assignment and establishing greediness level procedures are both part of the hyper-heuristics.

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 (in terms of solution's quality and time to reach feasible solution), evolutionary algorithm is used. The genotypes represent the aforementioned parameters (greediness level is a natural number no greater than 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 particular problem is generated using the parameters given in every genotype (to avoid infinite operation the first-level algorithm ceases to operate after finding a feasible solution or after 1000 iterations). The 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 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 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) – in case of 2^{nd}-level algorithm there is no threat that crossover operator could compromise data integrity (as in the 1^{st}-level algorithm), as it just mixes the parameters from two solutions. The procedure stops after fixed number of iterations or if no improvement has been made in two subsequent iterations.

4 Results of experiments

Some preliminary experiments have been conducted with the approach described in this paper. The first task was to prove, that the method is able to find a feasible solution for all the test problems. Ten sets from International Timetabling Competition has been used for 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. More extensive experiments were conducted on UCTP, as the problem is widely recognized and the results can be compared with those found in other publications. The results presented in the tables have the fitness function's values recalculated to match the method used for evaluating solution in ITTC (all weights equal one, second penalization method). As there's no analytical means to compare complexity of particular problem instances, only empirical comparison makes sense. All the datasets depict problems with similar sizes (about 400 events, 200 students and a dozen rooms and features).

4.1 First level algorithm – EA vs. tabu search

First set of experiments has been conducted to answer the underlying question: is using EA as a first-level algorithm is sensible. Many researches report excellent results using only some variation of tabu search, so the importance of this question cannot be underestimated. Every dataset has been solved twenty times for each method variation (EA only, TS only and EA with tabu search used to escape from local optima). Both algorithms operated until feasible solution has been found (or for 1000 generations if no such solution has been found). Greediness level for EA amounted 15, tabu list length was 150 and the population size 500. Second penalization method with automatic weight assignment has been used.

The feasibility of the solutions found has not been taken into account (most of the time the algorithms were not able to find one in only 1000 iterations) – it is possible, that two solutions with the same score exist, and only one of them is feasible.

Table 1. Results of experiments – first row is the best result obtained only with evolutionary algorithm, the second is the average of the best solutions in 50 runs, the next two rows are the best and average results using tabu search, last to – evolutionary algorithm combined with tabu search phase

Dataset	1	2	3	4	5	6	7	8	9	10
EA best	176	135	201	551	391	207	159	186	212	167
EA avg.	191	163	249	610	412	257	199	206	281	199
TS best	168	174	228	511	388	201	148	211	217	159
TS avg.	181	195	245	536	391	226	158	241	247	178
Both best	166	145	192	510	357	175	144	197	184	157
Both avg.	187	156	210	570	386	242	163	254	255	163

Tabu search shows better average result, although not all "best" results has been better than these of the EA. Probable cause of this phenomenon is that tabu search operates in more organized and predictable way, than EA. That comes with the price – EA is more likely to find a better solution by pure chance. The combination of TS and EA shows improvement in both best and average results – incorporating tabu search phase speeds up the search process. It is possible that using only tabu search would produce better overall solution but with considerably larger computational cost (spent on looking through tabu list).

4.2 Experiments with second-level algorithm

For all the experiments second-level EA had a population size of 100, and the first-level of 500. Second-level EA ran for 1000 iterations, and then the best 10 specimen of second-level algorithm ran first-level algorithm for 5000 generations.

Table presents results archived by ITTC participants, which will be used as comparison to the method presented in this paper. It has to be noted that the best results has been gathered from all the participants and the winner hasn't achieved best known solutions for all the problem instances. The experiment with algorithm described in this paper has been conducted 20 times and the table 2 shows averages of the best result.

Table 2. Results of experiments with second level algorithm.

Dataset	1	2	3	4	5	6	7	8	9	10
ITTC best	45	25	65	115	77	6	12	29	17	61
				28	24					
ITTC average	137	87	150	9	8	143	145	129	123	153
				39	33					
Without TS	158	103	156	9	6	146	125	110	154	153
With TS	141	101	145	340	271	138	107	98	146	139
				32	26					
Best 10 with TS	130	93	139	0	4	136	104	92	138	128

In all the experiments, feasible solution(s) has been found for all the problem instances before second level algorithm ceased to operate. As it can be seen, most of the results are better than average of the ITC score, but none is near the best score achieved by competitors. 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 the most effective way possible.

5 Conclusions and future work

The question whether universal, "knowledge-poor" method is able to perform better or at least comparably well as the domain-specific one remains open. In terms of computation time it's 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 some form of local search seems especially promising. Nevertheless, universal methods will always have one distinctive advantage over the specialized ones – they won't need laborious and time consuming process of redesigning and fine-tuning to fit specific needs.

Results of second-level algorithm's operation have to be looked into – it is possible, that some methods of penalization, weight assignment and distance measurement may prove useless for all the problem's variations and as such they may be removed from the search space.

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

References

1. Burke E. K., MacCarthy B., Petrovic S., Qu R.: *Structured cases in case-based reasoning – re-using and adapting cases for timetabling problems,* Knowledge-Based Systems **13** (2000).
2. Burke E. K., Petrovic S.: *Recent research directions in automated timetabling,* European Journal of Operational Research **140** (2002).
3. Burke E. K., Petrovic S., Meisels A., Qu R.: *A Graph-Based Hyper Heuristic for Timetabling Problems,* Computer Science Technical Report No. NOTTCS-TR-2004-9, University of Nottingham (2004).
4. 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).
5. 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).
6. Corne D., Ross P.: *Peckish Initialisation Strategies for Evolutionary Timetabling. Proceedings of the First International Conference on the Theory and Practice of Automated Timetabling,* Napier University, Edinburgh (1995).
7. Corne D., Ross P., Fang H.-L.: *Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation,* Parallel Problem Solving from Nature III, Springer Verlag (1994).
8. 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 496 (1990).
9. *International Timetabling Competition,* on the web `http://www.idsia.ch/Files/ttcomp2002/` (2002).
10 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).
11. Michalewicz Z.: *Genetic Algorithms + Data Structures = Evolution Programs,* Springer Verlag (1996).
12. Myszkowski P., Norberciak M.: *Evolutionary Algorithms for Timetable Problems,* Annales UMCS, Sectio Informatica, vol. I, Lublin (2003)
13. Newall J. P.: *Hybrid Methods for Automated Timetabling,* PhD Thesis, Department of Computer Science, University of Nottingham (1999).
14. Norberciak M.: *Feasible genotype initialization for evolutionary timetabling,* Proceedings of 9 th International Conference on Soft Computing MENDEL 2003, Brno (2003).
15. 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 **993** (1995).
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,* Proceedings of ANTS 2002, Springer-Verlag (2002).
18. Valouxis C., Housos E.: *Hybrid optimization techniques for the workshift and rest assignment of nursing personnel,* Artificial Intelligence in Medicine **20** (2000).
19. Yakhno T., Tekin E.: *Application of Constraint Hierarchy to Timetabling Problems,* Proceedings of EurAsia-ICT 002, Springer-Verlag (2002).