ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/221048818

Teaching Assignment Problem Solver

Conference Paper · June 2010

DOI: 10.1007/978-3-642-13025-0_32 · Source: DBLP

CITATIONS
6

READS
624

2 authors, including:

Malek Mouhoub
University of Regina

157 PUBLICATIONS **603** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Natural Language Processing View project

Project Combinatorial Optimization View project

All content following this page was uploaded by Malek Mouhoub on 16 July 2014.

The user has requested enhancement of the downloaded file.

Teaching Assignment Problem Solver

Ali Hmer and Malek Mouhoub

University of Regina
Wascana Parkway, Regina, SK, Canada, S4S 0A2
{hmer200a,mouhoubm}@cs.uregina.ca

Abstract. In this paper, we describe an extension approach to the backtracking with look-ahead forward checking method that adopts weighted partial satisfaction of soft constraints that has been implemented to the development of an automated teaching assignment timetabling system. Determining the optimal solution for a teaching assignment problem is a challenging task. The objective is to construct a timetable for professors from already scheduled courses that satisfy both hard constraints (problem requirements such as no teacher should be assigned two courses at the same time) and soft constraints (teacher preferences) based on fairness principle in distributing courses among professors. The approach is done mainly to modify the variable selection method and the value assignment technique taking into account preferences and based on fairness principle. The optimized look-ahead backtracking method applied to the solution is presented and discussed along with computational results.

Keywords: Teaching Assignment Problem, Soft Constraints, Constraint Optimization.

1 Introduction

In this paper, we describe a constraint programming system, with a web site as front-end to demonstrate a suggested solution technique of the Teaching Assignment Problem. The timetabling problem in general is mostly, if not always, an over constrained combinatorial optimization problem and hence it is considered one of the most difficult problems to solve. The Teaching Assignment Problem in essence is a branch of the timetabling problem and it is the problem of assigning professors to time slots that is occupied by courses in a specific week. The resulted weekly timetable is to be used to organize the teaching process at a university or any educational institute given that the courses have already been scheduled over the time slots and rooms. Each professor is assigned a total number of courses to be taught that should not be violated. Each professor is allowed to express interest or dislike in certain courses through weighed preferences that can or cannot be satisfied. Some professors can be assigned some courses in advance. The final solution should be constructed based on distributing the given courses over professors based on fairness principle as well as maximizing the total weight of the solution. The total weight of a solution is the sum of all satisfied preferences. The literature is very rich on the topic of university

© Springer-Verlag Berlin Heidelberg 2010

timetabling in general as there are different ways to solve the problem; most of them depend on specific needs considered by the institution that the timetabling is designed for. However, to the knowledge of the authors, no literature is dealing with the teaching assignment problem. In our case, we considered the problem as two-fold stages. The first is to assign courses to rooms and time slots and the second is to assign professors to the resulting time slots with courses. In this study, we only tackled the second one. Timetabling problems, in general, are usually over constrained as it is not always possible to satisfy all requirements. User preferences can be used to relax these requirements. In our study case, we have used a more specific model with preferences which utilizes weight for each constraint and try to maximize the total weight of satisfied soft constraints. As a development approach, our work includes a development of a solver for soft constraints. The solver was implemented by the authors as an extension of a well-known CSP solver named "Java Cream" [1] to include soft constraints in the backtracking mechanism which the Java Cream Solver is lacking. The solver itself was re-coded entirely, by the author, using Microsoft C# language from Java language. Some of the optimized technologies introduced in C# and in .NET framework, such as LINQ, were used to enhance and optimize the local search.

The next section of this paper provides a related work for the problem. Section 3 provides a description to the teaching assignment problem. The added soft-constraint approach that was implemented within the solver along with the modified search algorithm developed for this problem is detailed in section 4. This includes a description of how the problem has been solved as well as the representation of soft and hard constraints. Furthermore, a discussion on how the search is done is provided at the end of this section. Section 5 provides a description for the web based system used to implement the solver. Computational results are discussed in Section 6. The final section reviews the results of our work and looks to future extensions of the problem solution and soft-constraint solver improvements.

2 Related Work

Over the last 30 years, the timetabling problem is considered to be one of the broadly studied scheduling problems in Artificial Intelligence and Operations Research literature [2]. Educational timetabling, to be specific, has been the main topic of quite few papers in various scientific journals and the topic of many theses in academia society. The course timetabling problem deals effectively with courses and time slots which have to be scheduled during the academic term. The problem basically is the scheduling of a known number of courses into a known number of time slots spread all over the week in such a way that constraints are satisfied.

As there are many versions of the Timetabling Problem, a variety of techniques have been used to solve it [3], [4]. Most of these techniques range from graph colouring to heuristic algorithms. Another focus of research in the timetabling problem was on the application of a single solution approach which in effect a large variety of such approaches have been tried out, such as an integer programming approach [4], Tabu search [3], and Simulated Annealing [5]. Recently, some researchers have attempted to combine several approaches, such as hybridization of exact algorithms and Meta-heuristics. One of the most primitive methods used to solve this problem is graph colouring in which vertices represent events where two vertices are connected if and only if there is a conflict. [5], [6], [7], [8] and [9] proposed a number of formulations by graph colouring for a set of class teacher timetabling problems and discussed the inherent complexity. In [10], graph colouring has been used to solve course and exam timetabling. Linear programming models were also used to formulate the course time-tabling problem usually with binary variables [11], [12], [13], and [14]. An Integer Programming approach [15] was also used to model the timetabling problem as assignment problem with numerous types of constraints and large number of binary or integer variables. Rudov and Murray introduced an extension of constraint logic programming [16] that allows for weighted partial satisfaction of soft constraints is implemented to the development of an automated timetabling system. In [17], an Evolution Strategy to generate the optimal or near optimal schedule of classes is used to determine the best, or near best timetable of lecture/courses for a university department. Case Based Reasoning is another approach that has recently been applied to university timetabling [18], [19], [20], and [21]. Case Based Reasoning is believed to be studied as early as 1977 with the study of Schank and Abelson [22]. Case Based Reasoning has also been successfully applied to scheduling and optimization problems. Burke et al. [23] also, in a published article, developed a graph-based hyper-heuristic (GHH) which has its own search space that operates in high level with the solution space of the problem generated by the so-called low level heuristics.

3 Problem Description

In general, the timetabling problem is the assignment of time slots to a set of events. These assignments usually include many considerable constraints of different types. At the department of Computer Science, University of Regina, in any term, the timetabling process currently consists of constructing a class schedule prior to student registration. The professors and classes timetabling problem [24] and [25] is NP-Complete. The teaching assignment problem is the problem of assigning courses, scattered over time slots, to professors. In our case, the teaching assignment problem is described as follows.

1. There is a finite set of courses $C = \{c_1,\ c_2,\ \ldots,\ c_{|C|}\}$ and a finite set of time slots $T = \{t_1,\ t_2,\ \ldots,\ t_{|T|}\}$, which already have been assigned to courses C. This is typically provided as courses occupy time slots. So each course could occupy just one time slot, usually 3 hours; two time slots, usually an hour and half each; or 3 time slots, usually an hour each. For any course, the time slots assigned to it must not overlap. t_i can be assigned to different

courses as long as they are in different rooms and different professors. Our approach is nothing to do with these assignments as these assignments are considered as input for the problem to solve.

2. There is a finite set of professors $P = \{p_1,\ p_2,\ \ldots,\ p_{|P|}\}$.
3. In this scheduling problem, courses represent variables while professors represent variables domain values.
4. The problem is to schedule P to C in a way such that no professor p_i is in more than one place at a time t_i.
5. The constraints for this problem are soft and hard. The soft constraints should not all be satisfied and on the contrary all hard constraints must be satisfied so a possible solution to the problem is one that satisfies all the hard constraints but not necessary soft constraints.
6. Soft Constraints are preferences that do not deal with time conflicts and have weight (or Cost) associated with them. Our goal is to maximize the total weight of a solution (or minimize the total cost). We have two types of soft constraints; the first is count, where a professor has a maximum number of courses assigned to him that should not be exceeded. The second is preferences that any professor can express as interest or dislike in certain courses which have weights (or costs). This type of constraints can or cannot be satisfied.
7. In our case, we have two types of soft constraints, both of them related to professors preferences. These preferences named equal and not equal, which is indicate if a professors provided an interest or dislike in a that course (variable).
8. Hard Constraints are typically constraints that physically cannot be violated. This includes time slots that must not overlap in time, and in our problem time slots that overlap in time must not be taught by the same professor. There is another type of hard constraints where time slots represent a course can be assigned to a professor in advance prior to starting the search for a solution.
9. There is a total weight function that measures the quality of the current solution. The object of this function is to return the sum of all weights/costs associated with the satisfied preferences. The aim of the optimization technique is to maximize the total weight function or minimize the total cost.

4 Algorithm Description

As mentioned above, the solver, used in solving the problem, is re-coded from a well-known solver named *"Java Cream"* using Microsoft C# language. The original solver can be used to model any constraint satisfaction or optimization on finite domains problems [26]. However, it lacks any proper handling of soft constraints. As known, any timetabling/scheduling problem would be mostly over constrained and therefore it cannot be solved unless constraints are relaxed. Hence, the necessity came to add soft constraints as part of the re-coded solver to solve timetabling problems. The backtracking method is the one that was modified to take into account soft constraints. Although we describe only the modified backtracking method, which is used by two of the five methods that the solver adopts: Branch and Bound; and Iterative Branch and Bound. However, because preliminary tests showed that performance is not significantly improved in our application when using the other three methods; Taboo Search, Random Walk and Simulated Annealing, we only consider the backtracking method to meet our requirements. Furthermore, all solver search methods use the same variable/value ordering methods and hence would use the same approach mentioned here. We think that because the application study case variables and values are relatively small, the tests performance using other methods has not improved but might be better if another application is implemented which might have more complex variables and values.

Basically, the backtracking algorithm [27] has two phases. The first stage is what is called "a forward phase" in which the variables are selected sequentially and the current partial solution is extended by assigning a consistent value for the next variable if one exists. The second phase is known as "a backward phase" in which the algorithm returns to the previous assigned variable when no consistent solution exists for the current variable. For the forward phase, the adopted solver originally decides which variable to instantiate next by selecting the one that has minimum number of domain values (i.e. the one that its domain size is minimum). Then the solver decides which value to assign to the next variable by assigning the maximum value in the variable domain. By assigning a value to a variable, this value is eliminated from all other variables' domains. This is known as look-ahead backtracking.

The variables in the original solver are ordered according to their domain size and the variable with the highest domain size is first. In the modified solver, you still can use the same mechanism, but when "soft constraints approach" is used, variables are ordered by the highest weight on soft constraints and then on highest domain size.

The values in the original solver are ordered incrementally. However, in the modified solver that uses soft constraints approach, values are ordered by values associated to equal soft constraints that least have been assigned to any variable before first and then other values incrementally and last values that are associated with not equal soft constraints.

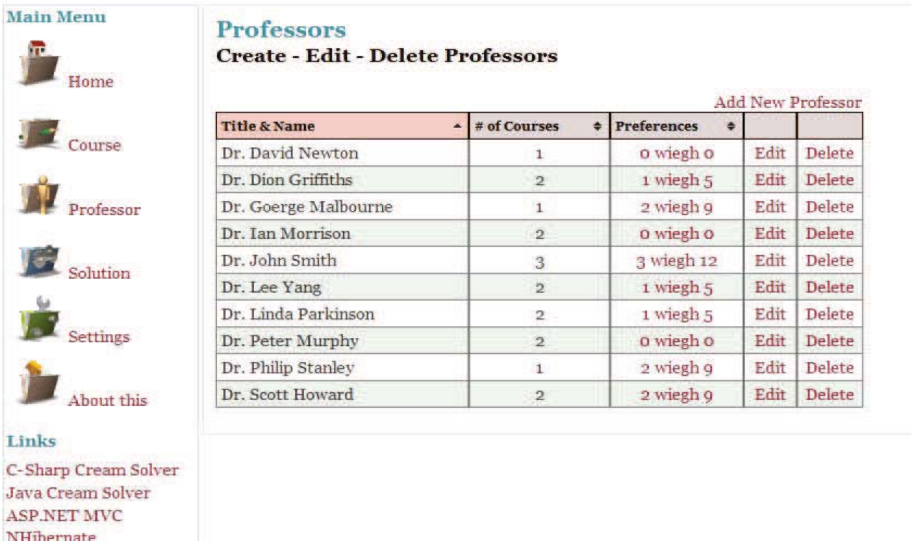
Because of the soft constraints that have been added to the solver, we have improved the two backtracking phases as follows if "soft constraint approach" method is selected in solving the problem:

1. On deciding which variable (Course) to instantiate next, the solver tries primarily to select the variable with the highest weight on equal soft constraints that have not been assigned a value (Its domain size is greater than one); if not then it will return randomly one of the variables (courses). The idea behind this is to try to select a variable that has soft constraints associated with it first, if not found then it will act on the other types of variables.
2. If the previous hint is not implemented, it will give the chance to assign values to variables that do not have soft constraints with them where they should

have been at least trying to be assigned to variables with soft constraints. In this case, variables with preferences will miss the chance to get their preferences assigned to them.
3. On deciding which value to assign to the variable selected in the previous step, a method, first, checks if there are equal soft constraints associated to that variable. If there are not any, then it will randomly select a value from its domain (i.e. domain values represent professors).
4. It is worth mentioning that even if there are no soft constraints associated with it, the method tries to not to choose a value that is associated with another variable that has an equal soft constraints as it might be needed in a later stage.
5. If there are indeed equal soft constraints associated to that variable, then it assigns the value that least has been assigned to any variable before. This is in compliance with the "fairness" principle. Furthermore, the value is selected randomly if there is more than one value.

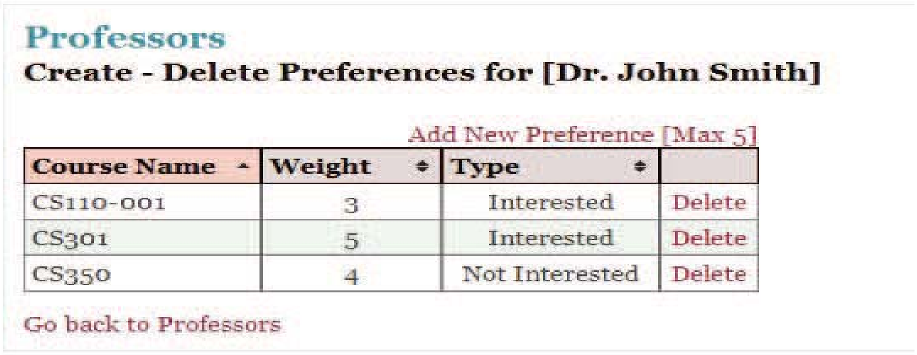
5 Web Based Interface for the Teaching Assignment Problem Solver System (TAPS)

We have implemented a web-based application for solving the timetabling problem. The idea behind this approach is to get professors to enter their preferences through the web site. Web based applications generally are more convenient for users. For instance, every professor can enter his/her preferences from office/home and there is no need to provide this information to application operator to enter their data. The Teaching Assignment Problem Solver (TAPS) was



Title & Name	# of Courses	Preferences		
Dr. David Newton	1	0 wiegh 0	Edit	Delete
Dr. Dion Griffiths	2	1 wiegh 5	Edit	Delete
Dr. Goerge Malbourne	1	2 wiegh 9	Edit	Delete
Dr. Ian Morrison	2	0 wiegh 0	Edit	Delete
Dr. John Smith	3	3 wiegh 12	Edit	Delete
Dr. Lee Yang	2	1 wiegh 5	Edit	Delete
Dr. Linda Parkinson	2	1 wiegh 5	Edit	Delete
Dr. Peter Murphy	2	0 wiegh 0	Edit	Delete
Dr. Philip Stanley	1	2 wiegh 9	Edit	Delete
Dr. Scott Howard	2	2 wiegh 9	Edit	Delete

Fig. 1. Professors information page



Professors

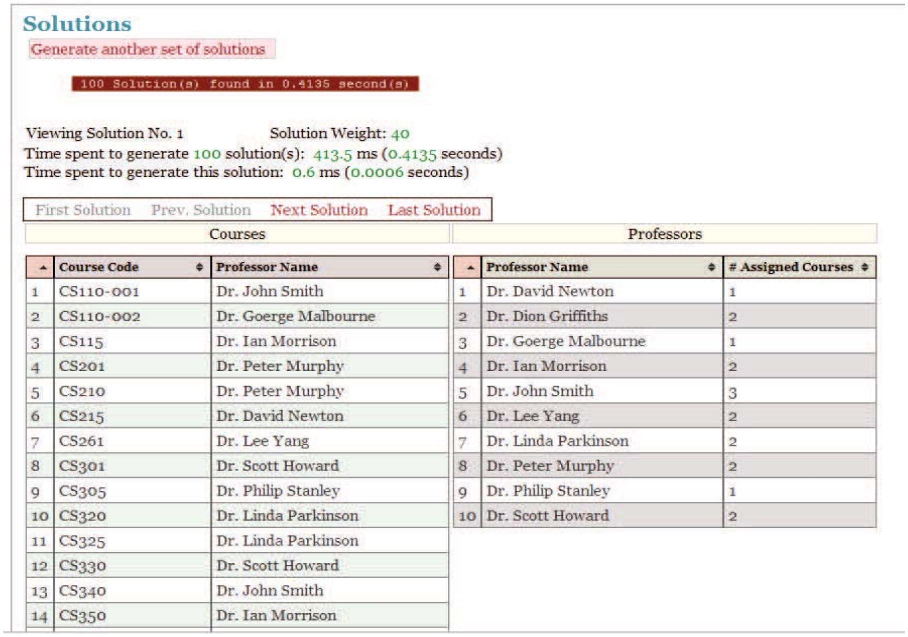
Create - Delete Preferences for [Dr. John Smith]

Add New Preference [Max 5]

Course Name	Weight	Type	
CS110-001	3	Interested	Delete
CS301	5	Interested	Delete
CS350	4	Not Interested	Delete

Go back to Professors

Fig. 2. Professors' preferences information page



Solutions

Generate another set of solutions

100 Solution(s) found in 0.4135 second(s)

Viewing Solution No. 1 Solution Weight: 40
Time spent to generate 100 solution(s): 413.5 ms (0.4135 seconds)
Time spent to generate this solution: 0.6 ms (0.0006 seconds)

First Solution Prev. Solution Next Solution Last Solution

Courses

	Course Code	Professor Name
1	CS110-001	Dr. John Smith
2	CS110-002	Dr. Goerge Malbourne
3	CS115	Dr. Ian Morrison
4	CS201	Dr. Peter Murphy
5	CS210	Dr. Peter Murphy
6	CS215	Dr. David Newton
7	CS261	Dr. Lee Yang
8	CS301	Dr. Scott Howard
9	CS305	Dr. Philip Stanley
10	CS320	Dr. Linda Parkinson
11	CS325	Dr. Linda Parkinson
12	CS330	Dr. Scott Howard
13	CS340	Dr. John Smith
14	CS350	Dr. Ian Morrison

Professors

	Professor Name	# Assigned Courses
1	Dr. David Newton	1
2	Dr. Dion Griffiths	2
3	Dr. Goerge Malbourne	1
4	Dr. Ian Morrison	2
5	Dr. John Smith	3
6	Dr. Lee Yang	2
7	Dr. Linda Parkinson	2
8	Dr. Peter Murphy	2
9	Dr. Philip Stanley	1
10	Dr. Scott Howard	2

Fig. 3. Solution page

developed using Microsoft ASP.NET MVC (Model-View-Controller) as an interface, Microsoft SQL Server 2005 as database engine and Internet Information Server (IIS) as web server. The MVC model provides a rich graphic user interface using HTML and JQuery (Java script based library). There are four main parts for the web site. The first is devoted to courses management and its information can be entered by administrator. The information includes the course assigned week days, assigned time slots, and assigned professor (if needed). The later will be treated by the solver as hard constraint. The interface also displays the number of professors interested and not interested in that course. The whole courses are displayed as a table where there is an option for inserting, editing and deleting a course. The second is dedicated to professors' information management and their information can be entered by professors themselves. The information includes the number of courses that can be assigned to each professor (i.e. count constraint) and the professors' preferences. Both preferences will be dealt by the solver as soft constraints. The third is for searching for a solution using the information provided and using the C# cream as a background solver. The solution section provides an interface for searching for solutions using the information provided in the previous two sections. If solutions are found, they will be displayed in this section's web page. Among the information displayed, there is time spent for generating all solutions and the time spent for each solution along with the solution weight. There is also the option to display the next and previous solution. There are three tables; the first one is the main table where courses are displayed with professors. The second table displays professor's names and the number of assigned courses. The displays all constraints (hard and soft) used in finding the solutions. The last section is for Web site settings. This includes the number of hours per course, maximum break minutes per session, maximum number of courses per professor, number of preferences per professors, maximum number of generated solutions, the option to generate only better solutions in terms of solution weight and the maximum timeout that should be used in the solver to generate solutions. The screen shots below illustrate the professors and solutions sections.

6 Experimental Tests and Results

The experimental tests compare between our proposed approach and the original backtracking method.

In order to do tests on the designed Web site and the proposed solver, we used data from the Computer Science department at the University of Regina for both courses and professors information including courses time slots. Then we assigned randomly some of the courses to some professors and assigned professors to show some interest and dislikes in some of the courses.

Overall, we used 17 courses as solver variables and 10 professors as solver values. From these courses we entered interest in 4 courses for different professors and disinterest in just one course. Experimental computations were done with a number of objectives in mind. The main goal was to provide a table of courses assigned to professors where all hard constraints are satisfied and the weight of soft constraints is maximized.

The second experiment involved using the same solver to solve a problem with the same variables and domain values but using non preference approach (The original backtracking method).

We have also set the solver to generate the first 100 solutions considering the first solution is the most optimized one and to generate only same or better weighted solutions and the solver has only 100 seconds to generate any solution at any given time. We used a PC with the following capabilities: Core 2 Duo Quad processor (2.4 GHz) with 6 GB ram. We have asked the solver to search for solutions 10 times to get a bigger picture of the search time spent in finding solutions.

The results showed that the average time to find a solution was between 1.92 ms and 2.567 ms. When using non-preference approach (i.e. original solver's ordinary variable/value ordering), we were unable to find a feasible solution that can satisfy the maximum number of soft constraint in the first 100 solutions. On the contrary, when using the preference approach, the first 10 solutions were optimal for our problem that satisfied hard and soft constraints.

7 Conclusion

We have successfully applied modified back tracking method to solve the teaching assignment problem. Feasible schedules were obtained for real data sets, including professors' preferences without the need for a huge computational effort. The original solver was meant to solve integer based variables problems but for problems with hard constraints. We have extended the solver to adopt soft constraints and we think that it has been a success. In conclusion, this application of Teaching Assignment Problem Solver appears to be quite successful and we are satisfied and ready to implement it to generate actual schedules for future terms. We also think that this approach can be implemented in similar problems like Exam Supervision scheduling. Based on the gathered experience of this test, we concluded that this approach is computationally feasible.

References

1. Tamura, N.: Cream: Class Library for Constraint Programming in Java. Kobe University (2009), `http://bach.istc.kobe-u.ac.jp/cream/`
2. Valdes, R.A., Crespo, E., Tamarit, J.M.: Design and implementation of a course scheduling system using Tabu Search. European Journal of Operational Research 37, 512–523 (2002)
3. Burke, E.K., Jackson, K., Kingston, J., Weare, R.E.: Automated university timetabling: the state of the art. The computer journal 40, 565–571 (1997)
4. Carter, M.W., Laporte, G.: Recent developments in practical course timetabling. In: Burke, E.K., Carter, M. (eds.) PATAT 1997. LNCS, vol. 1408, pp. 3–19. Springer, Heidelberg (1998)
5. Welsh, D.J.A., Powell, M.B.: An upper bound for the chromatic number of graph and its application to timetabling problems.1. The Computer Journal 10, 360–364 (1967)
6. Wood, D.C.A.: Technique for colouring a graph applicable to large scale timetabling problems. The Computer Journal 12, 317–319 (1969)
7. Selim, S.M.: Split Vertices in Vertex colouring and their application in developping a solution to the faculty timetable problem. The Computer Journal 31, 76–82 (1988)
8. Burke, E.K., Ross, P. (eds.): PATAT 1995. LNCS, vol. 1153, pp. 296–308. Springer, Heidelberg (1996)
9. Miner, S., Elmohamed, S., Yau, H.W.: Optimizing Timetabling Solutions Using Graph Coloring. NY: NPAC REU program, NPAC, Syracuse University (1995)

10. Timothy, A.R.: A Study of university timetabling that blends graph coloring with the satisfaction of various essential and preferential conditions. Rice University: Ph.D. Thesis (2004)
11. Daskalaki, S., Birbas, T., Housos, E.: An integer programming formulation for a case study in university timetabling. European Journal of Operational Research, 117–135 (2004)
12. Daskalaki, S., Birbas, T.: Efficient solutions for university timetabling problem through integer programming. European Journal of Operational Research, 106–121 (2005)
13. Dimopoulou, M., Miliotis: An Automated Course Timetabling System developed in a distributed Environment: a Case Study. European Journal of Operational Research, 153, 136–148 (2004)
14. Dimopoulou, M., Miliotis, P.: Implementation of a University Course and Examination Timetabling System. European Journal of Operational Research 130, 202–213 (2001)
15. Schimmelpfeng, k., Helber, S.: Application of a real-world university-course timetabling model solved by integer programming. Springer, Heidelberg (2006)
16. Rudov, H., Murray, K.: University Course Timetabling with Soft Constraints, pp. 310–327. Springer, Heidelberg (2003)
17. George, T.B., Opalikhin, V., Chung, C.J.: Using an Evolution Strategy for a University Timetabling System with a Web Based Interface to Gather Real Student Data. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2724. Springer, Heidelberg (2003)
18. Burke, E.K., MacCathy, B., Petrovic, S., Qu, R.: Case-based reasoning in course timetabling: an attribute graph approach. In: Aha, D.W., Watson, I. (eds.) ICCBR 2001. LNCS (LNAI), vol. 2080, pp. 90–105. Springer, Heidelberg (2001)
19. Burke, E.K., MacCathy, B., Petrovic, S., Qu, R.: Multiple-retrieval case-based reasoning for course timetabling problems. Journal of the Operational Research Society, 1–15 (2005)
20. Burke, E.K., MacCathy, B., Petrovic, S.: Knowledge discovery in a hyperheuristic for course timetabling using case-based reasoning. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 90–103. Springer, Heidelberg (2003)
21. Burke, E.K., MacCathy, B., Petrovic, S., Qu, R.: Structured case in case-based reasoning-re-using and adapting cases for timetabling problems. Knowledge-Based Systems 13, 159–165 (2000)
22. Schank, R.C., Abelson, R.P.: Scripts, plans, goals and understanding. Erlbaum, New Jersey (1977)
23. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyperheuristic for educational timetabling problem. European Journal of Operational Research, 1–16 (2006)
24. Gislen, L., Soderberg, B., Peterson, C.: Teachers and Classes with Neural Nets. International Journal of Neural Systems 1, 167–168 (1989)
25. Gislen, L., Soderberg, B., Peterson, C.: Complex scheduling with Potts neural networks. Neural Computation 4, 805–831 (1992)
26. Tamura, N.: Calc/Cream: OpenOffice Spreadsheet Front-End for Constraint Programming. In: Umeda, M., Wolf, A., Bartenstein, O., Geske, U., Seipel, D., Takata, O. (eds.) INAP 2005. LNCS, vol. 4369, pp. 81–87. Springer, Heidelberg (2006)
27. Dechter, R.: Constraint Processing, 1st edn., pp. 123–128. Morgan Kaufmann, San Francisco (2003)

View publication stats