A Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for University Course Timetabling

Salwani Abdullah[*] Edmund K. Burke[*] Barry McCollum[†]

[*]Automated Scheduling, Optimisation and Planning Research Group, School of Computer Science & Information Technology, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, United Kingdom
{sqa,ekb}@cs.nott.ac.uk
[†]Department of Computer Science, Queen's University Belfast, Belfast BT7 1NN United Kingdom
b.mccollum@qub.ac.uk

Extended Abstract

1 Introduction

The general course timetabling problem consists of assigning courses to a specific timeslot and room. The goal is to satisfy as many soft constraints as possible while constructing a feasible schedule (i.e. one that satisfies all the hard constraints). In this paper, we present a composite neighbourhood structure with a randomised iterative improvement algorithm. Many relevant publications can be found in the literature. For example, see the volumes of papers from the international conferences on the Practice and Theory of Automated Timetabling (in Burke and Ross 1996, Burke and Carter 1998, Burke and Erben 2001 and Burke and Causmaecker 2003). Comprehensive surveys on timetabling can be found in de Werra (1985), Burke et al. (1997), Schaerf (1999), Burke and Petrovic (2002) and Petrovic and Burke (2004). The approach that we present in this abstract is tested over eleven benchmark datasets. The results demonstrate that our approach is able to produce solutions that are better than others that appear in the literature.

2 The Problem

Course timetabling problem is subject to a variety of hard and soft constraints. Hard constraints need to be satisfied in order to produce a feasible solution. In this paper, we will test our approach on the problem instances introduced by Socha et al. (2002) who present the following hard constraints:

- *No student can be assigned to more than one course at the same time.*
- *The room should satisfy the features required by the course.*
- *The number of students attending the course should be less than or equal to the capacity of the room.*
- *No more than one course is allowed at a timeslot in each room.*

Socha et al. (2002) present the following soft constraints that are equally penalized and we aim to minimise the penalty:

- *A student has a course scheduled in the last timeslot of the day.*
- *A student has more than 2 consecutive courses.*
- *A student has a single course on a day.*

The problem has

- A set of N courses, $e = \{e_1,\ldots,e_N\}$
- 45 timeslots (5 days with 9 timeslots each day)
- A set of R rooms
- A set of F room features
- A set of M students.

The objective of this problem is to satisfy the hard constraints and to minimise the violation of the soft constraints. This extended abstract is organised as follows: a problem description is presented in Section 2. The description of the composite neighbourhood structure applied to the randomised iterative improvement algorithm is presented in Section 3. The experiments and results are discussed in Section 4. Brief conclusions are drawn in Section 5.

3 The Randomised Iterative Improvement Algorithm

3.1 Initial Solution

The initial solution is produced using a constructive heuristic which starts from an empty timetable. This feasible solution is obtained by adding or removing appropriate events (courses) from the schedule based on room availability (we attempt to schedule those courses with the least room availabilities earlier on in the process), without taking into account any of the soft constraints, until the hard constraints are met. This constructive heuristic behaves likes a saturation degree heuristic (see Carter and Laporte, 1996). The schedule is made feasible before starting the algorithms.

3.2 The Neighbourhood Structures

We implemented the following neighbourhood structures to be used in our algorithm. We first presented N1-N8 in Abdullah et al. (2005) but we reproduce them here (along with three extra

structures for completeness and clarity):

N1: Select a course at random and find another course at random which can swap timeslots.

N2: Choose a single course at random and move to another random feasible timeslot.

N3: Select two timeslots at random and simply swap all courses in one timeslot with all courses in the other timeslot.

N4: Move a timeslot. Take 2 timeslots (selected at random), say t_i and t_j (where $j>i$) and the timeslots are ordered t_1, t_2, …, t_{45}. Take all the exams in t_i and allocate them to t_j. Now take the exams that were in t_j and allocate them to t_{j-1}. Then allocate those that were in t_{j-1} to t_{j-2} and so on until we allocate those that were in t_{i+1} to t_i and terminate the process.

N5: Move highest penalty course from a random 10% selection of the courses to a random feasible timeslot.

N6: Move highest penalty course from a random 20% selection of the courses to a random feasible timeslot.

N7: Move the highest penalty course (i.e. a course with the highest number of soft constraint violations). Take 10% of the courses at random. Then select the one with the highest penalty cost and allocate it to the timeslot which generates the lowest penalty and which does not create an infeasibility.

N8: Move highest penalty course from a random 20% selection of the courses (as in (N7)).

N9: Select one course at random, select a timeslot at random (distinct from the one that was assigned to the selected course) and then apply the kempe chain from Thompson and Dowsland (1996).

N10: This is the same as N9 except we select the highest penalty course from 5% selection of the courses at random.

N11: As N10 but with 20% of the courses.

In order to maintain the feasibility of the solution, the room allocation of courses in each timeslot (after a kempe chain operation) cannot exceed the space available and, of course, courses in each timeslot should be scheduled in different rooms.

3.3 The Algorithm

For the approach presented in this paper, we apply a set of the neighbourhood structures as in subsection 3.1. The hard constraints are never violated during the course of the timetabling process. Figure 1 shows a schematic overview of the approach. The algorithm starts with a feasible initial solution which is generated by a constructive heuristic. Let K be the total number of neighbourhood structures to be used in the search (K is set to be 11 in this implementation) and $f(Sol)$ is the quality measure of the solution Sol. At the start, the best solution, Sol_{best} and the

previous solution, Sol_{prev} are set to be *Sol*. In a *do-while* loop each neighbourhood *i* where $i \in \{1,\ldots,K\}$ is applied to the *Sol* to obtain $TempSol_i$. The best solution among $TempSol_i$ is identified, and is set to be the new solution *Sol**. If *Sol** is better than the best solution in hand Sol_{best}, then *Sol** is accepted. Otherwise the exponential monte carlo acceptance criteria is applied (see Ayob and Kendall 2003). Exponential monte carlo is only based on the solution quality. It accepts a worse solution with a certain probability. For example, given the old and new solutions as *Sol* and *Sol**, respectively. The new solution *Sol** is accepted if a generated random number in [0,1] is less than $e^{-\delta}$ where $\delta = f(Sol^*) - f(Sol)$. The increasing value of δ will decrease the probability of accepting worse solutions. The details on the exponential monte carlo can be found in Ayob and Kendall (2003).

```
Set the initial solution Sol by employing a constructive heuristic;
Calculate initial cost function f(Sol);
Set best solution Sol_best ← Sol;
do while (not termination criteria)
    for i = 1 to i= K where K is the total number of neighbourhood structures
        Apply neighbourhood structure on Sol, TempSol_i;
        Calculate cost function f(TempSol_i);
        Find the best solution among TempSol_i where i ∈ {1,…,K} call new solution
        Sol*;
        if (f(Sol*) < f(Sol_best))
            Sol ←  Sol*;
            Sol_best ← Sol*;
        else
            δ = f(Sol*) -  f(Sol));
            Generate RandNum, a random number in [0,1];
            if (RandNum < e^-δ ) then Sol ← Sol*;
    end for
end do;
```

Figure 1. The pseudo code for the randomised iterative improvement algorithm for course timetabling

4 Experiments and Results

The proposed method was tested on the benchmark course timetabling problems presented by Socha et al. (2002). They are grouped into 5 *small* (N = 100, R = 5, F = 5 and M = 80), 5 *medium* (N = 400, R = 10, F = 10 and M = 200) and one *large* problems (N = 400, R = 10, F = 10 and M = 400). The approach is coded in Microsoft Visual C++ version 6 under Windows. The experiments were run on the Athlon machine with a 1.2GHz processor and 256 MB RAM running under Microsoft Windows 2000 version 5. The number of evaluations for our approach

is 200000 (as in Socha et al. 2002). The evaluation will terminate if the penalty cost is zero or the number of evaluations is 200000. The best and average results out of 5 runs obtained are presented. Table 1 shows the comparison of our approach with other available approaches in the literature i.e. a local search method and ant algorithm by Socha et al. (2002), a tabu-search hyperheuristic by Burke et al. (2003), a graph hyperheuristic by Burke et al. (2005) and a variable neighbourhood search (VNS) with a tabu list by Abdullah et al. (2005). The term "x%Inf" in Table 1 indicates a proportion of runs that failed to obtain feasible solutions.

Table 1: Comparison results on course timetabling problem

Dataset	Initial solution	Our method		VNS with tabu list (Best)	Local search (Average)	Ant algorithm (Average)	Tabu based hyper -heuristic (Best)	Graph hyper -heuristic (Best)
		Best	Average					
small1	261	0	**0**	**0**	8	1	1	6
small2	245	0	**0**	**0**	11	3	2	7
small3	232	0	**0**	**0**	8	1	**0**	3
small4	158	0	**0**	**0**	7	1	1	3
small5	421	0	**0**	**0**	5	**0**	**0**	4
medium1	914	242	245	317	199	195	**146**	372
medium2	878	161	**162.6**	313	202.5	184	173	419
medium3	941	265	267.8	357	77.5% Inf	**248**	267	359
medium4	865	181	183.6	247	177.5	**164.5**	169	348
medium5	780	151	**152.6**	292	100% Inf	219.5	303	171
large	100%Inf.	100% Inf	100%Inf	100%Inf	100% Inf	**851.5**	80% Inf 1166	1068

In terms of feasibility, our approach is able to produce ten out of eleven feasible solutions (as in our previous paper in Abdullah et al., 2005), whereas Socha et al.'s local search only produced nine feasible solutions (with two infeasible solutions for the *medium4* and *large* datasets). The best results are presented in bold. In general, we obtained better than or equal results on seven datasets. We believe that we can obtain better solutions in these experiments (particularly on the smaller problems) because the composite neighbourhood structures offer some flexibility for the search algorithm to explore different regions of the solution space.

5 Conclusion and Future Work

The overall goal of this paper was to investigate a composite neighbourhood structure with a randomised iterative improvement algorithm for the course timetabling problem. Preliminary comparisons indicate that our approach is competitive with other approaches in the literature. Our future work will try to tackle real world datasets in course timetabling.

References

[1] Abdullah S, Burke EK and McCollum B (2005) An investigation of variable neighbourhood search for university course timetabling. Accepted for publication in *The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2005)*.

[2] Ayob M and Kendall G (2003) A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. *Proc. Of the International Conference on Intelligent Technologies, InTech'03*, pp 132-141.

[3] Burke EK and Ross P, editors, (1996) *Practice and Theory of Automated Timetabling I*, volume 1153 of Lecture Notes in Computer Science. Springer-Verlag.

[4] Burke EK, Jackson KS, Kingston JH and Weare RF (1997) Automated timetabling: The state of the art, *The Computer Journal*, volume 40, No. 9, pp 565-571.

[5] Burke EK and Carter MW, editors, (1998) *Practice and Theory of Automated Timetabling II*, volume 1408 of Lecture Notes in Computer Science. Springer-Verlag.

[6] Burke EK and Erben W, editors, (2001) *Practice and Theory of Automated Timetabling III*, volume 2079 of Lecture Notes in Computer Science. Springer-Verlag.

[7] Burke EK and De Causmaecker P, editors, (2003) *Practice and Theory of Automated Timetabling IV*, volume 2740 of Lecture Notes in Computer Science. Springer-Verlag.

[8] Burke EK and Petrovic S (2002) Recent research direction in automated timetabling. *European Journal of Operational Research* 140, pp 266-280.

[9] Burke EK, Kendall G and Soubeiga E (2003) A tabu search hyperheuristic for timetabling and rostering. *Journal of Heuristics* 9(6), pp 451-470.

[10] Burke EK, Meisels A, Petrovic S and Qu R (2005) A graph-based hyper heuristic for timetabling problems. Accepted for publication in the *European Journal of Operational Research.*

[11] Carter, M.W. and Laporte, G. (1996) Recent developments in practical examination timetabling. In [3], pp 3-21.

[12] de Werra D (1985) An introduction to timetabling. *European Journal of Operations Research* 19, pp 151-162.

[13] Petrovic S and Burke EK (2004) University timetabling, Ch. 45 in the *Handbook of Scheduling: Algorithms, Models, and Performance Analysis* (eds. J. Leung), Chapman Hall/CRC Press.

[14] Schaerf A (1999) A survey of automated timetabling. *Artificial Intelligence Review* 13(2), pp 87-127.

[15] Socha K, Knowles J and Samples M (2002) A max-min ant system for the university course timetabling problem. *Proceedings of the 3rd International Workshop on Ant Algorithms, ANTS 2002*, Lecture Notes in Computer Science 2463 (10), pp 1-13.

[16] Thompson J and Dowsland K (1996) Various of simulated annealing for the examination timetabling problem. *Annals of Operational Research* 63, pp 105-128.