Learning Assignment Order in an Ant Colony Optimiser for the University Course Timetabling Problem

James Sakal
js1188@exeter.ac.uk
University of Exeter
UK

Jonathan E. Fieldsend
J.E.Fieldsend@exeter.ac.uk
University of Exeter
UK

Edward Keedwell
E.C.Keedwell@exeter.ac.uk
University of Exeter
UK

ABSTRACT

Previous studies have employed Ant Colony Optimisation to solve the University Course Timetabling task — which requires the order of lecture assignments to be defined for its construction graph. Various heuristic or random ordering techniques have been proposed in the literature, but uncertainty remains regarding the best approach for this. We investigate the effect that permuting assignment order has on the quality of timetable produced. As part of this we develop a novel MAX-MIN Ant System including dynamic constraint handling and partial function evaluations. We also explore algorithm variants with and without Local Search and employ a form of transfer learning to identify appropriate permutations. We find that between smaller problems in the International Timetabling Competition 2007 benchmark, timetabling performance can be improved using such an approach. However we find that we lose this performance gain when attempting to transfer to considerably larger problems — indicating that similar structures are required when using a 'learnt' permutation approach in such a framework.

CCS CONCEPTS

• **Theory of computation → Scheduling algorithms**.

KEYWORDS

Timetabling and scheduling, ant algorithms, combinatorial optimisation, empirical study.

ACM Reference Format:
James Sakal, Jonathan E. Fieldsend, and Edward Keedwell. 2021. Learning Assignment Order in an Ant Colony Optimiser for the University Course Timetabling Problem. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3449726.3459534

1 INTRODUCTION

The University Course Timetabling Problem (UCTP) refers to generating a workable university timetable by assigning lectures to discrete locations in time and space. It is a well studied problem in combinatorial optimisation, known to be computationally hard [6].

A popular metaheuristic for such discrete optimisation problems is Ant Colony Optimisation (ACO). One of the first applications of ACO to the UCTP can be found in [6]. In our study, we develop a system based around a variant of ACO known as MAX-MIN Ant System (MMAS). One unavoidable design choice is the order of lecture assignments in the construction graph. While some consideration has been given to heuristic approaches in the literature ([3], [5]), we employ machine learning techniques. Our experiments are conducted on a standardised problem formulation proposed by the International Timetabling Competition (ITC) 2007 Track 3 [2].

2 METHOD

We develop a system based on the components below.

Representation: Our construction graph is comprised of $|\mathcal{P}| \times |\mathcal{L}| \times |\mathcal{R}| + 2$ nodes, where $\mathcal{P}$, $\mathcal{L}$ and $\mathcal{R}$ are the sets of periods, lectures and rooms respectively. This is encoded as a 3D matrix, plus start and end nodes. Rather than connect nodes by edges, we employ an edgeless graph in which pheromone is deposited directly on nodes. As [5] notes, this treatment, a permutation-based ACO, removes the combinatorial complexity and memory issues associated with handling a large matrix of edge values.

ACO Formulation: At each iteration, k ants independently begin trails at the start node, s, before visiting exactly one node in each sequential lecture plane and terminating at end node e. Every node in the main matrix corresponds to a distinct placing of a lecture, and thus a single ant trail encodes a complete solution. Timetables are scored using a weighted sum of soft and hard constraint violations — a single objective that we seek to minimise. Pheromone values are bounded by parameters τ_{min} and τ_{max} and initialised to the latter. We employ an elite update strategy for adjusting these values, using either the iteration best or global best ant.

Dynamic Constraint Handling (DCH): For hard constraints that are known *a priori*, pheromone values for the relevant nodes are set to zero. Other hard constraints are dependent on previous lecture assignments. These are handled dynamically by assigning the relevant nodes a value that is a fraction of τ_{min}. This guarantees complete assignment of lectures while permitting the search to move through infeasible space.

Partial function evaluations: When evaluating a given solution, we make smart decisions by abandoning at the earliest opportunity those which cannot yield the iteration best score.

Local Search (LS): Our routine attempts to improve on the elite ant solution. We adopt 3 of the specialised neighbourhood operators from chapter 5.3 of [4]: `timeMove`, `roomMove` and `lectureMove`, as well as a swap operator `perRoomSwap` which attempts to swap the room and period of two randomly chosen lectures. When activated, LS is called periodically during a run of iterations.

Course assignment order: Our motivation for determining the order of lecture assignments in a construction graph lies in a desire

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07.
https://doi.org/10.1145/3449726.3459534

to assign 'harder' lectures earlier — while a wider choice of feasible placement options remains open. We first build a training set of eleven 5-course instances using our problem instance generator, all of which have at least one solution with no constraint violations at all. For each course C_i in the set of courses C in an instance, values for eight impactful features are extracted and normalised. The features capture both characteristics inherent to a course such as number of lectures, as well as inter-course relationships such as the conflict density between curriculum memberships. As we are interested in the relative position of courses in a permutation, we cross the 8 features with position indices $1 \ldots 5$ and concatenate these vectors, yielding a single feature vector of length $8 \times 5 = 40$ for each permutation. Without LS, we observe that, over the permutation space of size $5! = 120$, certain clusters of permutations converge to a perfect solution more efficiently than others. Such variation occurred across all training set instances.

For our predictor, the target is the sample mean (over 30 trials) of the number of full function evaluations taken to find a zero violation score. If none has been found within 150s CPU time, we take the number of full function evaluations completed to that point. We train a set of regression models on 5 cross-validated folds using the standard CART node-splitting technique [1]. Minimum leaf size is used for termination and we optimise over this parameter. An average loss (mean squared error) over all folds of 0.0373 is achieved, with a minimum leaf size of 15.

In order to use this model to make predictions about larger unseen problems with $|C| > 5$, we apply a mapping:

$$\frac{1}{|Q_i|} \sum_{C \in Q_i} \texttt{feat} \rightarrow \texttt{ifeat} \tag{1}$$

where Q_i is the set of courses comprising the i^{th} quintile of a permutation, by position, for $i = 1 \ldots 5$, `feat` is the normalised value of a generic feature, and `ifeat` is the subsequent positional feature. By way of this averaging, we obtain a standard 40-feature vector irrespective of $|C|$. Every such vector has an associated fitness — namely the normalised prediction value returned by the model. We may then search the course permutation space for permutations of high fitness. While the size of this space ($|C|!$) is large, the fitness landscape is simplified by the discrete decision tree and mapping inherent to our models.

We locate regions of 'good' permutations using a simple genetic algorithm (GA). This is built on the partially matched crossover (PMX) operator [7], which tends to respect the absolute positions of courses within a permutation, and a mutation operator which switches the positions of two randomly chosen courses. The fittest permutation is returned after 750 generations of the GA, which was found to be sufficient time for convergence.

3 RESULTS AND CONCLUSIONS

Three small problems ($|C| = 30, 30, 47$) and three large problems ($|C| = 79, 99, 74$) were used. Tables 1 and 2 show results for the predictor and a baseline in which perms were chosen at random, over 50 repetitions of 100k full function evaluation runs.

The best 'mean best solution score' (shaded) indicates superior performance of the predictor on the smaller problems - with statistical significance on `comp11` (p-value 6.5×10^{-5}) and `comp18` (3.1×10^{-3}). Complex feature relationships of the larger problems (whose sizes were 15.8×- 19.8× greater than our training problems) proved harder to capture. This may indicate the importance of similarity in size or structure between training problems and those we wish to predict for. Restricting training size to 5-course problems enabled testing of all permutations, with the trade-off that some features could only take a small number of discrete values. In further experiments incorporating LS, we observed a flattening of results across the board. We note that LS can incur a high computational cost and its benefits must be weighed up against this.

Table 1: Performance on small problems

Variant	Value	comp01	comp11	comp18
randPerm	Mean best solution score	78	152	278
	S.d. best solution score	30	23	19
predictor	Mean best solution score	71	134	266
	S.d. best solution score	19	15	17

Table 2: Performance on large problems

Variant	Value	comp04	comp17	comp19
randPerm	Mean best solution score	2873	2860	1425
	S.d. best solution score	117	514	206
predictor	Mean best solution score	2932	3802	1592
	S.d. best solution score	131	243	497

In summary, we developed an MMAS system that was able to find feasible solutions to all problems tested. Smart function evaluations enabled efficient use of budget, while DCH guaranteed complete solutions and helped guide the search towards feasibility. In the non-LS variant, our predictor model was able to improve average scores when compared to picking a random lecture assignment order, provided the problems were not excessively larger in size.

REFERENCES

[1] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone. 1983. *Classification and Regression Trees.* Wadsworth International Group.

[2] Luca di Gaspero, Andrea Schaerf, and Barry McCollum. 2007. The Second International Timetabling Competition : Curriculum-based Course Timetabling (Track 3). *Electrical Engineering (2007)* Track 3 (2007), 1–21.

[3] Vatroslav Dino Matijaš, Goran Molnar, Marko Čupić, Domagoj Jakobović, and Bojana Dalbelo Bašić. 2010. University course timetabling using ACO: A case study on laboratory exercises. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6276 LNAI, PART 1 (2010), 100–110.

[4] T. Müller. 2009. ITC2007 solver description: A hybrid approach. *Annals of Operations Research* 172, 1 (2009), 429–446. https://doi.org/10.1007/s10479-009-0644-y

[5] Clemens Nothegger, Alfred Mayer, Andreas Chwatal, and Günther R. Raidl. 2012. Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research* 194, 1 (2012), 325–339.

[6] Olivia Rossi-Doria, Michael Sampels, Mauro Birattari, Marco Chiarandini, Marco Dorigo, Luca M. Gambardella, Joshua Knowles, Max Manfrin, Monaldo Mastrolilli, Ben Paechter, Luis Paquete, and Thomas Stützle. 2003. *A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem.* Lecture Notes in Computer Science, Volume 2740, pp 329-351.

[7] S. N. Sivanandam and S. N. Deepa. 2008. *Introduction to Genetic Algorithms.* Springer, Berlin, Heidelberg, New York.