MISTA 2013

Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem

Shahriar Asta · Daniel Karapetyan · Ahmed Kheiri · Ender Özcan · Andrew J. Parkes[*]

1 Introduction

This paper briefly describes our submission to the project scheduling challenge[1]. The full description of this problem domain can be found on the competition website and elsewhere; however, for completeness we briefly summarise it here. The broad aim is to schedule a set of different and partially interacting projects. Each project consists of a set of activities. The activities must respect a set of (hard) precedence constraints and project release times. Also the activities use resources and the appropriate resource limits are also hard constraints. There are basically two different ways to distinguish resources. Firstly, they can be either renewable or non-renewable. Renewable resources are ones that are available again at their full capacity whenever current activities stop using them, for example, they could be some machine. Non-renewable ones disappear on usage — an example could be fuel where one can take any amount of fuel, but only until the tank is empty. The second distinction between resources is that of 'local' resources that are associated with one of the projects, and 'global resources' that are shared between different projects. In the competition, there were no global non-renewable resources, and so the only interaction between projects is from the global renewable resource(s). A complication is that each activity actually can be performed using any one of a set of 'modes'. The mode determines the set of resources used by the activity and the duration of the activity (though note that the modes do not affect the set of precedences). A solution consists of an assignment of mode and starting time to every activity and that satisfies all the precedence and resource constraints.

Given a solution then each project i has an associated makespan MS_i which is the time from it being released to the time the last activity is completed. The primary objective is to minimise the "Total Project Delay" (TPD), which (up to constant terms) is the sum of the makespans MS_i for each project i. The tie-breaking secondary objective is to minimise the overall "Total Makespan" (TMS), which (up to a constant term) is the finishing time of the last activity.

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
{ sba, dxk, axk, exo, ajp }@cs.nott.ac.uk
∗ Contact Author

[1] `http://allserv.kahosl.be/mista2013challenge/`

In designing an algorithm, there are two 'natural' representations to be used in the search for an assignment of activities to times:

Schedule-based: A direct representation using the assignment times of activities.
Sequence-based: This is based on a selecting a total order on all the activities. Given such a sequence then a time schedule is constructed by taking the activities one at a time in the order of the sequence and placing each one at the earliest time that it will go in the schedule.

The schedule based method is direct, and (perhaps) most natural for a mathematical programming approach, but we believed it could make a metaheuristic method difficult, in particular, it becomes more challenging to generate many feasible solutions. Hence, we used the sequence method as it has the advantage of being easier to produce schedules that are both feasible and for which no activity in such a schedule can be moved to an earlier time without moving some other activities. Our overall approach is in two phases in a "construct and improve" fashion. Firstly, a heuristic constructor is used to create initial sequences. The novelty here is that we generate it using Monte-Carlo Tree Search (MCTS) methods [1]. Secondly, an improvement phase is applied using a large variety of heuristic neighbourhood moves. This phase is carefully controlled by a combination of methods arising from standard metaheuristics, memetic algorithms, and also an extension of an existing hyper-heuristic [2].

2 Construction Phase

On inspecting good solutions, we found many cases had an approximate ordering of the projects: There would be times in the schedule when the general focus would be on one or few projects and during the schedule this focus would change. This is natural when the primary objective is the delay-based TPD rather than on the overall makespan, and it suggested that a good constructor should attempt to create initial sequences that mimic such project (partial) orderings. The problem then is how to quickly compare between different project orderings. We use an MCTS method in which the fast "rollout" is done by means of a randomised schedule constructor. As standard in MCTS, the aim is not to directly produce good solutions but to use an unbiased sampling to make good decisions. There is also an option as to whether to generate total or partial orders between the projects. We settled for a 3-way partition of the projects taken to correspond to 'start', 'middle' and 'end' of the overall project time. The MCTS is then used to select the projects constituting each of the partitions. In the rollout, the order of activities within each project partition is taken as a random one that respects the precedences.

3 Improvement Phase

The improvement phase uses an extension of [2] together with a large set of moves, or "Low Level Heuristics". Some of which are given below; all moves are restricted, as needed, to only generate sequences that respect all the precedence constraints.

1. SwapJobs: Swap two activities within the sequence.
2. InsertJob: Insert a given activity into a new location in the solution sequence.
3. SetMode: Change the mode of a activity to a new randomly chosen mode.

4. MoveUniform: A set of activities is selected uniformly at random. The move has three options: moving activities, changing modes or both. When moving the selected activities they are reshuffled randomly. When changing the modes, random modes are assigned to the selected activities.
5. MoveLocal: Similar to MoveUniform, with the difference being that the activities are selected using a specific distribution that in not uniform but that is centered around a controllable position, and with a controllable width, within the sequence.
6. MoveOneProject: Similar to MoveUniform, but the activity selection is biased towards a randomly selected project.
7. MoveBiasedGlobalResource: Similar to MoveUniform, but favours selection of activities which have a bigger remaining capacity on the global resources.
8. MoveEndBiased: Similar to MoveBiasedGlobalResource, but favours the activities with a position close to the end of their project.
9. FILS swapJobs/insertJobs/setMode: these three moves are First Improvement Local Search (FILS) procedures based on the swapping, inserting or mode changing limited neighbourhoods.
10. SwapTwoProjects: Swaps two randomly selected projects in the sequence.
11. MutationOneExtreme: the activities of a randomly selected project are all collected and squeezed into a randomly-selected position in the sequence.
12. MutationOne: Shifts all the activities of a randomly selected project by a fixed number of positions in the sequence.
13. MoveProjects: Extracts the sequence of the projects in the solution (based on the positions of the last activities in each project), selects several consequent projects, and then moves them to either the beginning or end of the sequence.

The moves are controlled by a combination of meta-heuristics and a hyper-heuristic, all in the context of a multi-threaded population-based approach that uses ideas from memetic algorithms.

4 Implementation Experiments and Results

As might be expected, the many of our experiments were concerned with selecting the algorithms and then (partially) tuning the many parameters that are possible within the overall search control algorithm. It was also important to a careful implementation so that the construction of the schedule from a sequence was as fast as possible. For example, it helped to use “prefix sharing” in which the construction from the current sequence can reuse the results of a previous construction, at least up until the point at which the two activity sequences differ.

We provide some preliminary experimental results in Table 1. For reference purposes, we include the results from the competition website of the best results of the qualification phase. We then give our average performance of the submitted code under conditions similar to the competition, and, for purposes of comparison, the best values that we ever encountered during all our experiments. It can be seen that under competition conditions, we were generally able to beat (or equal) the results from the challenge website, but that there is still room for improvement on most of the instances.

Instance	Quals best		Our Mean		Our Best	
A-1	1	23	1	23	1	23
A-2	2	41	2	41	2	41
A-3	0	50	0	50	0	50
A-4	65	42	65	42	65	42
A-5	153	105	152	104	150	104
A-6	147	96	136	91	133	91
A-7	596	196	602	202	589	201
A-8	302	155	280	151	272	151
A-9	223	119	203	126	196	125
A-10	969	314	850	308	840	306
B-1	—	—	352	128	345	125
B-2	—	—	441	169	424	165
B-3	—	—	547	211	526	208
B-4	—	—	1279	285	1254	277
B-5	—	—	831	247	809	251
B-6	—	—	936	233	897	224
B-7	—	—	796	231	782	229
B-8	—	—	3315	534	3088	533
B-9	—	—	4197	755	4097	745
B-10	—	—	3238	454	3136	445

Table 1 Experimental results summary. The objective values are given as ordered pairs of "TPD", total project delay, and "TMS", total makespan. The 'Quals best' values are from the competition website for all the entrants in the qualification round. 'Our Mean' values are average values for our algorithm under time and machine conditions intended to match those of the competition. 'Our Best' is the best solution encountered over a large number of runs with a variety of runtimes and parameter settings.

5 Conclusions

Our algorithm consists of a two-phase construct-and-improve method working on the sequence in which activities are given to a schedule constructor. The construction of an initial activity sequence is done by a (novel) hybrid of MCTS and partitioning of the projects. The improvement phase uses a large number of neighbourhood moves, in a multi-threaded fashion, and controlled by a mix of ideas from metaheuristics, memetic algorithms, and hyper-heuristics.

References

1. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. Computational Intelligence and AI in Games, IEEE Transactions on **4**(1), 1–43 (2012). DOI 10.1109/TCIAIG.2012.2186810
2. Kheiri, A., Özcan, E., Parkes, A.J.: HySST: Hyper-heuristic search strategies and timetabling. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012) (2012)