A Heuristic Approach for the Multi-Project Scheduling Problem with Resource Transition Constraints

Markus Berg, Tobias Fischer, and Sebastian Velten

Abstract A resource transition constraint models sequence dependent setup costs between activities on the same resource. In this work, we propose a heuristic for the multi-project scheduling problem with resource transition constraints, which relies on constraint programming and local search methods. The objective is to minimize the project delay, earliness and throughput time, while at the same time reducing setup costs. In computational results, we demonstrate the effectiveness of an implementation based on the presented concepts using instances from practice.

Keywords Project scheduling · Transition constraints · Setup costs

1 Introduction

Project scheduling problems have been the subject of extensive research for many decades. A well-known standard problem is the *Multi-Project Scheduling Problem with Resource Constraints (RCMPSP)*. It involves the issue of determining the starting times of project activities under satisfaction of precedence and resource constraints. As an extension of RCMPSP, we consider the *Multi-Project Scheduling Problem with Resource Transition Constraints (MPSPRTC)*, where setup costs and times depend on the sequence in which activities are processed.

The applications of MPSPRTC are abundant, e.g. in production processes with cleaning, painting or printing operations. In many of these applications, the presence of parallel projects and scarce resources makes scheduling a difficult task. In addition, there is competition between activities for planning time points with lowest

M. Berg
proALPHA Business Solutions GmbH, Weilerbach, Germany
e-mail: markus.berg@proalpha.de

T. Fischer (✉) · S. Velten
Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany
e-mail: tobias.fischer@itwm.fraunhofer.de; sebastian.velten@itwm.fraunhofer.de

J. S. Neufeld et al. (eds.), *Operations Research Proceedings 2019*, Operations Research Proceedings, https://doi.org/10.1007/978-3-030-48439-2_70

setup costs and times. Often, project due dates can only be met if the activities are scheduled accurately and the available resources are used optimally.

The appearance of resource transitions in project scheduling is already investigated in Krüger and Scholl [1]. They formulate MPSPRTC as an integer program and present a priority-based heuristic with promising computational results. Within their framework, there are no due dates on the projects and the aim is to minimize the average project ends. This goal goes hand in hand with minimizing the sequence-dependent setup times of the activities and therefore there is no multi-objective oriented optimization framework.

In this work, we present a priority-based heuristic for MPSPRTC using constraint programming that we extend by a local search and solution refinement procedure. The proposed algorithm is a multi-criteria approach for determining a good compromise between low setup costs, adherence to project due dates, and short project throughput times. In our model, we restrict us to the case that all setup times are zero and there only exist sequence dependent setup costs.

The algorithm is divided into 3 steps: The first step is to find a practicable initial solution using a constructive heuristic (see Sect. 2.1). The heuristic relies on priority rules that include the goals of the objective functions. Starting from the initial solution, we apply a local search in the second step of our algorithm (see Sect. 2.3), where we try to improve the setup costs by permuting the activity sequences on the setup resources. Finally, in the third step (see Sect. 2.3), we refine the solution calculated in the previous steps by a forward-backward improvement heuristic. The goal is to bring the activities closer to the due dates of the corresponding projects, while keeping the sequence of activities on the setup resources essentially unchanged. In a computational study in Sect. 3, we demonstrate the effectiveness of an implementation based on the presented concepts using instances from practice.

1.1 Problem Definition

We consider a set of projects $P = \{p_1, \ldots, p_m\}$ with due dates $d_1, \ldots, d_m$. Each project $p \in \mathcal{P}$ is composed of a set $\mathcal{A}_p$ of activities with a fixed working time w_a for each $a \in \mathcal{A}_p$. By $\mathcal{A} := \bigcup_{p=1}^{m} \mathcal{A}_p$, we denote the set of all activities. The start and end time points of these activities are variable and should be determined in such a way that all constraints are met and the target criteria are optimized.

The constraints required by MPSPRTC are listed below:

1. *Precedence Constraints:* Each activity can have several predecessors and may not be started until all its predecessors are complete. We assume that precedence constraints only exist between activities of the same project. The precedence constraints of each project $p \in \mathcal{P}$ are represented by the directed edges $\mathcal{E}_p$ of a precedence graph $G_p = (\mathcal{A}_p, \mathcal{E}_p)$.
2. *Resource Requirements:* Let $\mathcal{R}$ be the set of resources, which are assumed to be renewable and to have a varying integer capacity over time. The execution of

every activity $a \in \mathcal{A}$ requires a constant number of resource units u_{ar} of each resource $r \in \mathcal{R}$. By $\{a \in \mathcal{A} : u_{ar} > 0\}$ we denote the set of activities that are assigned to r.

3. *Resource Transition Constraints:* These constraints are used to model sequence dependent setup costs on a resource $r \in \mathcal{R}$. Assume that every activity assigned to r has a certain setup type of a set $\mathcal{T}_r$. Activities of different types may not overlap, i.e, they cannot be allocated in parallel on r. A nonnegative integer matrix M_r of order $|\mathcal{T}_r| \times |\mathcal{T}_r|$ contains the setup costs required to change the setup state of r to another setup state. In most applications, the diagonal entries of M all are 0.

The (multi-)objective is to minimize the delay, earliness and throughput time of the projects, while at the same time reducing setup costs.

2 Scheduling Heuristic for MPSPRTC

In the following sections, we describe the three steps of our scheduling heuristic.

2.1 Initial Planning

We use a greedy heuristic to find a feasible solution for MPSPRTC. The heuristic relies on priority rules taking the characteristics of the problem into account. Projects (respectively, activities) with highest priority are scheduled first towards their specific goals. We define priority rules for the following aspects:

1. *Scheduling sequence of projects:* Projects are scheduled in the order of a user-defined priority. If projects have an equal user-defined priority, then those with the earliest due date are scheduled first.
2. *Scheduling sequence of activities:* Activities of a project are sorted with first priority by the partial order induced by G_p and as second priority by the order of their current latest end time; latest activities are scheduled first.
3. *Point in time to schedule activities:* Activities are scheduled as close as possible to their intended point in time: For non-setup activities, this is the point x that is as close as possible to the due date d of the corresponding project. For setup activities we search in an interval $[x - \delta, x + \delta]$ with $\delta > 0$ for a point that leads to the lowest setup costs w.r.t. the current schedule. If two points lead to the same cost, then we take one that is closest to d.

All precedence constraints $(a, \bar{a}) \in \mathcal{E}_p$, $p \in \mathcal{P}$, in which either a or $\bar{a}$ is a setup activity, are modeled with an additional time offset of ε, i.e., a ends at least ε time units before the start of $\bar{a}$. Here, ε is a parameter that can be selected by the user. The extra time between activities is not necessary for the feasibility of the solution,

but beneficial for the performance of the local search in Sect. 2.2, where we need enough freedom to perform iterative permutations of the setup sequences. Then the aim of the solution refinement in Sect. 2.3 is to remove the extra offset from the schedule. Reasonable choices of ε are discussed in Sect. 3.

2.2 Local Search

The algorithm of Sect. 2.1 does not necessarily result in local optimal setup costs—even if the other goals (earliness, tardiness and throughput time) are fixed in their goal levels. To find improved setup costs, we propose a local search procedure that is applied to the activity order of each setup resource.

Starting from the activity sequence S_0 found in the initial planning (Sect. 2.1), we iteratively move from one sequence S_i to another S_{i+1}. The sequence S_{i+1} is selected from a neighborhood set $\mathcal{N}(S_i)$, where we choose the sequence with the best improvement w.r.t. some priority rule R. The feasibility of the current sequence is checked every iteration (or less frequently for further speedup). If an infeasibility is detected, we backtrack to the last feasible sequence and try the next candidate. This process is continued until no improvement occurs anymore or a maximum iteration number is met.

From a theoretical point, the feasibility of a sequence S could be checked by rescheduling all activities subject to the condition that the ordering specified by S must be satisfied. However, since feasibility has to be checked quite often, this can be very time consuming. Therefore, we restrict MPSPRTC to the subproblem where all activities except the ones of S are fixed. Of course, this has a restrictive effect, which is however partly compensated by adding the additional time offset ε to precedence constraints in the initial planning (see Sect. 2.1). Generally it can be said that, the larger we choose ε, the more freedoms we receive for rescheduling the setup activities and it is more likely that a valid sequence can be detected as such. On the other hand, if we choose ε small, then the earliness, tardiness and throughput time of the projects tends to be smaller.

It remains to specify the neighborhood set $\mathcal{N}(S)$ and the priority rule R: Two sequences S and S' are neighbours if S can be transformed into S' by either a pairwise exchange of two sequence elements, or a shift of one sequence element at another position, or a shift of a group of consecutive sequence elements with the same setup type at another position. Moreover, the priority rule R relies on the sum of setup costs and a measure based on the Lehmer mean [3] that prefers large numbers of consecutive occurrences of the same setup type.

2.3 Solution Refinement

In the third step of our method, we refine the solution calculated in the previous steps. This is done by a forward-backward improvement (FBI) heuristic. The goal is to move the activities closer to the due dates of the corresponding projects, while keeping the sequence of the setup activities essentially unchanged.

The FBI heuristic consists of two steps: In the forward step, the activities are considered from left to right (based on the order of the current schedule) and scheduled to their earliest feasible point in time. Similarly, in the backward step, the activities are considered from right to left (according to the order of the forward schedule) and scheduled as close as possible to their due dates. The approach can be repeated multiple times until no improvement occurs anymore.

3 Computational Experience

In this section, we report on computational experience with our implementation of the presented algorithm. The implementation is written in C++ using the framework ILOG CP Optimizer V12.6.2 [2], which allows to express relations between interval variables and cumulative variables in the form of constraints.

We use four different kind of instance classes which arise from rolling horizon ERP data of three customers. The instances of classes 3–4 correspond to the same customer, however different kind of resources are marked as setup resources. Table 1 gives statistics on all four instance classes. Column "#" denotes the number of instances of the given class. Moreover, in arithmetic mean over all instances of each class, we list the number of projects in "Projs", the number of activities in "Acts", the number of precedences in "Precs", the number of resources in "Ress", the number of setup resources in "S-Ress", the number of setup activities in "S-Act", the number of different setup types in "S-Types", and the definition of the setup cost matrices in "M_{ij}". We defined a planning horizon in which setup costs are optimized. The horizon is useful for controlling the complexity of the problem. For our purposes, we decided that a large horizon of 60 days would be appropriate, since the activities can be multiple days long.

Our computational results are organized into two experiments. In the first experiment, we use the default settings of Table 2, but vary the parameter ε from

Table 1 Statistics of the instance classes

Instances	#	Projs	Acts	Precs	Ress	S-Ress	S-Acts	S-Types	M_{ij}
Class 1	6	5287	30818	27591	750	2	1249	153	$0\ (i = j), 1\ (i \neq j)$
Class 2	1	4367	79318	82602	111	1	2490	87	$\lvert i - j \rvert$
Class 3	7	8699	67247	66064	919	9	676	317	$0\ (i = j), 1\ (i \neq j)$
Class 4	7	8699	67247	66064	919	2	8366	11	$0\ (i = j), 1\ (i \neq j)$

Table 2 Settings for test runs

Shortcut	Explanation
Default	Default settings (all methods enabled)
Initial-off	Initial planning (Sect. 2.1) is done without taking care of resource transfers
ls-off	Turn local search off (Sect. 2.2)
Refine-off	Turn solution refinement off (Sect. 2.3)
Setup-off	Combination of "ls-off", "Refine-off", and "Initial-off"

Table 3 Experiments on the 4 instance classes (in arithmetic mean)

Setting		Class 1					Class 2				
Shortcut	ε	Costs	thr	earl	tard	Time	Costs	thr	earl	tard	Time
Setup-off	0	0.96	5.66	2.14	2.73	32	31.08	7.55	0.64	11.34	73
Default	1	0.59	5.81	2.23	2.73	310	5.57	7.70	0.58	12.39	568
	2	0.54	5.86	2.29	2.86	295	4.91	7.83	0.64	12.15	630
	4	0.49	5.90	2.39	3.04	281	4.08	8.16	0.78	12.32	534
	6	0.46	5.93	2.57	3.07	266	4.60	8.00	0.84	12.74	625
	8	0.43	5.97	2.57	3.25	167	4.67	8.27	0.87	12.89	600
Initial-off	4	0.59	5.88	2.56	2.91	201	5.30	7.76	0.69	11.14	558
ls-off	4	0.62	5.95	2.36	3.09	135	18.21	8.14	0.68	12.35	442
Refine-off	4	0.47	6.16	2.29	3.34	82	3.52	8.20	0.82	13.03	180
Setting		Class 3					Class 4				
Shortcut	ε	Costs	thr	earl	tard	Time	Costs	thr	earl	tard	Time
Setup-off	0	0.87	6.61	0.64	9.69	142	0.388	6.61	0.64	9.69	151
Default	1	0.77	6.57	0.75	9.51	507	0.041	6.89	0.87	9.84	1178
	2	0.76	6.57	0.76	9.55	496	0.036	7.17	0.96	10.21	1081
	4	0.73	6.59	0.77	9.64	501	0.032	7.53	1.12	10.75	1036
	6	0.71	6.61	0.79	9.71	503	0.033	7.69	1.22	11.16	1086
	8	0.68	6.63	0.81	9.78	491	0.033	7.94	1.33	11.64	1142
Initial-off	4	0.77	6.61	0.69	9.67	493	0.048	7.29	1.02	10.76	1169
ls-off	4	0.79	6.59	0.73	9.79	491	0.132	7.63	0.82	11.14	591
Refine-off	4	0.72	6.66	6.70	9.93	145	0.025	8.53	1.04	12.03	627

Sect. 2.1 between 1 and 8 days. In the second experiment, we evaluate the effect of the three basic steps (Sects. 2.1–2.3) of our algorithm by switching different components off. The data of Table 3 shows aggregated results of these experiments. For each instance class, the table reports on the mean number of setup costs per setup activity (column "Costs"), the mean number of days of the throughput time, earliness and tardiness per project (columns "thr", "earl" and "tard"), and the CPU time in seconds (column "Time").

The results can be summarized as follows: A larger value for ε tends to result in lower setup costs. The reason is that ε controls the degrees of freedom we get for optimizing the setup sequences in the local search. On the other hand, if we choose ε small, we obtain better values for the earliness, tardiness and throughput time of the

projects. This reflects the fact that minimal setup costs and optimal project dates are contrary targets. Comparing the default settings with "Setup-off", it turns out that our algorithm is able to significantly improve the setup costs without worsening the project statistics too much. On the other hand, the solution time for optimizing setup costs is significantly increased.

We proceed with an evaluation of the three basic steps of our algorithm, see rows "Initial-off", "ls-off" and "Refine-off'. For these test runs, we decided to choose $\varepsilon = 4$, since this was a good trade-off in our last experiment. If setup cost optimization is deactivated either in the initial planning or the local search step, then we observe a significant deterioration in this goal, however for the sake of the project statistics. Moreover, solution refinement turns out to successfully remove a lot of free space from the schedule, which originates from the additional precedence offset ε in the initial planning. This is demonstrated by the fact that the project statistics tend to get worse when solution refinement is switched off.

4 Conclusion and Outlook

In this paper, we discussed and computationally tested a heuristic approach based on constraint programming for the MPSPRTC. The algorithm particularly addresses the issue of finding a good trade-off between low setup costs and compliance with project due dates. We presented a standard constructive heuristic which we extended by a local search and solution refinement procedure. The algorithm was tested on large real-world data of different customers. Our computational results demonstrate that this extension clearly outperforms the setup costs without worsening the throughput time, earliness and tardiness of the projects too much. Future research is necessary to develop more efficient heuristics to speed up the whole solving process.

References

1. Krüger, D., Scholl, A.: A heuristic solution framework for the resource constrained multi-project scheduling problem with sequence-dependent transfer times. Eur. J. Oper. Res. **197**(2), 492–394 (2009)
2. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP optimizer for scheduling. Constraints **23**(2), 210–250 (2018)
3. Lehmer, D.H.: On the compounding of certain means. J. Math. Anal. Appl. **36**, 183–200 (1971)