Constraint Programming applied to the Multi-Skill Project Scheduling Problem

No Author Given

No Institute Given

Abstract The Multi-Skill Project Scheduling Problem is a variant of the well-studied Resource Constrained Project Scheduling Problem, in which the resources are assumed to be multi-skilled. Practical applications of this problem occur when the resources considered are a multi-skilled workforce or multi-purpose machines. This variant introduces a set of assignment decisions between the resources and activities, further to the usual scheduling decisions. This additional layer of complexity results in the problem becoming far more difficult to solve. We investigate different constraint programming models and searches tailored for solvers with nogood learning. These models and searches are then evaluated on instances available from the literature as well as newly generated ones. Using the best performing model and search, we are able to close at least 87 open instances from the literature.

1 Introduction

The Resource Constrained Project Scheduling Problem (RCPSP) is one of the most widely studied combinatorial optimization problems [10,26] and is the basic problem for the herein studied Multi-Skill Project Scheduling Problem (MSPSP). Solving RCPSP involves finding the optimal schedule to perform a set of non-preemptive activities which satisfies the given precedence relations. Limitations on the available resources are also imposed and must be respected throughout the project's duration. In most cases, the objective when solving this problem is to find the shortest possible duration of the project, which we call the makespan.

In RCPSP and many of its variants, it is assumed that each resource only has one capability or *skill*. However, when the resources considered are multi-skilled workers or multi-purpose machines then each could have a variety of skills. MSPSP is the extension of RCPSP, in which each resource can have multiple skills. In addition, it is assumed that each resource can use only one skill at each time. For example, a logistic company has a workforce of truck drivers holding different licenses for driving vehicles ranging from vans to large trucks.

MSPSP consists of non-preemptive activities, which require time and skills for their execution, some precedence relations between pairs of activities, and renewable multi-skilled resources. We assume that all resources are unary, because multi-skilled resources are typical human resources in real scenarios. In a solution, all precedence relations are satisfied, no resource is overloaded at any time, and no resource uses more than one skill at any time. In addition, each

Table 1: Resources' Skills

	R_1	R_2	R_3	R_4
S_1	-	✓	✓	✓
S_2	✓	-	-	-
S_3	✓	✓	-	✓

Table 2: Activity Skill Requirement

	A_0	A_1	A_2	A_3	A_4	A_5
S_1	-	-	1	2	1	-
S_2	-	1	-	-	1	-
S_3	-	1	1	-	-	-

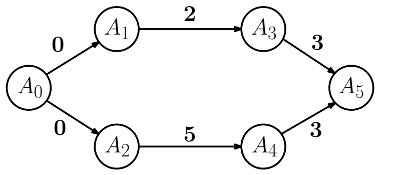

Figure 1: Precedence Graph

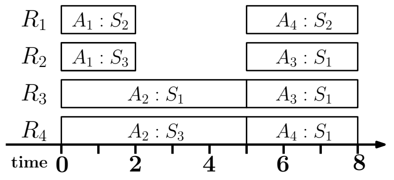

Figure 2: Feasible schedule and assignment

activity must be fully processed by the same resources from the start to the end. Naturally, a solution can be divided into the scheduling decisions, which assign each activity to a start time, and the assignment decisions, which assign each activity to one or more resources having the required skills. As noted by Correia and Saldanha-da-Gama [7], adding these assignment decisions to the unit-skilled RCPSP presents a non-trivial layer of complexity. Hence, the MSPSP is NP-hard [3].

Example 1 (Adapted from [16]). The small example is made up of activities A_0, A_1, ..., A_5, for which A_0 (A_5) is a fictitious source (sink) activity having a zero duration, and resources R_1, R_2, ..., R_4 having one or more of the three skills S_1, S_2, and S_3. Table 1 defines the skills which each resource has mastered and Tab. 2 defines the skill requirements of each activity. Figure 1 shows the precedence relations which must be respected, with the processing times of each activity shown on the directed edges. Figure 2 contains a solution showing the start times of the non-fictitious activities, assignment of the resources to activities and skill contribution of each resource. This solution has a makespan of 8. □

In the past two decades Constraint Programming (CP) has been used extensively to tackle a variety of combinatorial optimization problems. One of its successes lies in solving resource-constrained scheduling problems, which was further boosted since the introduction of Lazy Clause Generation (LCG) [18]. LCG is a CP solver with nogood learning facility from Boolean Satisfiability Solving (SAT). It has been shown to effectively prune the search space with conflict driven search to guide the solver [25,12,22,21,19,20,23]. In line with these works, we investigate the different CP models and searches for MSPSP using the MiniZinc modeling language and show the effectiveness of LCG by beating the state-of-the-art methods on instances from the literature and newly generated ones. When using the best performing model and search, which utilizes a recent extension of the search facilities of MiniZinc, the LCG solver `Chuffed` [6] was able to close at least 87 open instances that were made available to us from the literature.

2 Literature Review

To the best of our knowledge, the first contribution to the literature concerning the MSPSP was by Néron [16] who proposed two lower bounds on the makespan. These bounds were later enhanced in [9]. [15] provides strong lower bounds using Lagrangian relaxation and column generation. Exact branch-and-bound methods were developed in [5,4] based on the reduction of the slack of one activity at each node. More recently, the authors of [13,14] propose an approach using column generation and a branch-and-price framework.

Correia et al. [8] propose a mixed-integer linear program for solving MSPSP. They strengthen their model by adding many valid cuts, which are computed in a pre-processing step. For evaluating their method, they created a data set which was inspired by the data generator for RCPSP in [11]. Building upon this work, [1] develop an instance generator for MSPSP and create a larger data set. Later the same author [2] proposed a constructive heuristic using a priority-based parallel scheduling scheme. Correia and Saldanha-da-Gama [7] design a generic modeling framework for the MSPSP as well as preprocessing and enhancement methods. Their framework provides the basis of our CP model in Sect. 4.

A closely related problem is the Multi-Mode RCPSP, in which each activity has one or more execution modes and a solver has to decide, in which mode an activity is processed. Recently all open instances from standard benchmark sets were closed by [25] using the LCG solver `Chuffed`. As noted in [4], MSPSP can be reduced to Multi-Mode RCPSP by creating one execution mode for each possible activity-resource assignment potentially leading to a large number of modes.

3 Problem Definition

The MSPSP consists of *non-preemptive activities* $V = \{0, 1, \ldots, n+1\}$, *a set of precedence relations* $E \subseteq V \times V$, *unary renewable resources* $R = \{1, 2, \ldots, m\}$, and *skills* $S = \{1, 2, \ldots, l\}$. An activity $i \in V$ is characterized by a *duration* (processing time) $p_i \in \mathbb{N}^0$ and the *skill requirement* $sr_i^s \in \mathbb{N}^0$ for each skill $s \in S$, where $sr_i^s = 0$ means the skill s is not required. The activities 0 and $n+1$ are fictitious marking the start and the end of the project. Both activities have a zero duration and no skill requirements. A resource $r \in R$ is defined by the *skill mastery* $mast_r^s \in \{0, 1\}$ for each skill $s \in S$, where $mast_r^s = 1$ ($mast_r^s = 0$) means that the resource (does not) masters the skill. A solution assigns each activity i to not only a start time $s_i \in \mathbb{N}^0$, but also the resource-skill combinations $y_{ir}^s \in$

$\{0,1\}$, that are used for executing i, so that the following constraints are satisfied.

$$s_i + p_i \leq s_j \qquad \forall (i,j) \in E \tag{1}$$

$$\sum\nolimits_{r \in R} y_{ir}^s = sr_i^s \qquad \forall i \in V, \forall s \in S \tag{2}$$

$$\sum\nolimits_{s \in S} \sum\nolimits_{i \in V : s_i \leq t < s_i + p_i} y_{ir}^s \leq 1 \qquad \forall r \in R, \forall t \in \mathbb{N}^0 \tag{3}$$

$$y_{ir}^s \leq mast_r^s \qquad \forall i \in V, \forall r \in R, \forall s \in S \tag{4}$$

Constraint (1) models the precedence relations, (2) ensures that required resource-skills are assigned to each activity, (3) states that a unary resource only can use one skill at any time, and (4) ensures that a resource only contributes with a skill that it mastered. The goal is to find a solution that minimizes the makespan, *i.e.*, $\min_{i \in V} s_i + p_i$ or, simply, $\min s_{n+1}$.

4 Constraint Programming Model

The inherent similarities between the Multi-Skill and the Multi-Mode RCPSP motivated us to employ a similar approach to Szeredi and Schutt [25] using CP with the modeling language MiniZinc [17]. MiniZinc allows the user to define application-tailored search strategies that guide the solver's branching strategy when exploring the solution space which proved helpful for this problem.

4.1 Basic Variables and Constraints

As the previous section indicated, we have a start time variable s_i and resource-skill assignment variables y_{ir}^s for each activity i. They are linked with the input data via (2), (4), and the following ones, which ensures that a resource only contributes at most one skill for the execution of an activity.

$$\sum\nolimits_{s \in S} y_{ir}^s \leq 1 \qquad \forall i \in V, \forall r \in R \tag{5}$$

Precedence relations are modeled as stated in constraint (1). Note that all non-fictitious activities have at least one predecessor and one successor, whereas the fictitious activity 0 ($n+1$) has no predecessor (successor). We set the start time $s_0 = 0$ and, thus, the start time of $n+1$ equals the project makespan if started immediately after the completion of all its predecessors.

4.2 Unary Resource Constraints

The modeling of the unary resource constraints (3) is an important ingredient for efficient solving of MSPSP. We investigate different model choices tailored to the LCG solver `Chuffed` and other CP solvers with nogood learning. Note that some model choices are clearly weaker for CP solvers without nogood learning, due to the weaker propagation strength, but for CP solvers with nogood learning the weaker propagation strength might be more than compensated by the stronger learning that is provided by the model choice.

Time-indexed Decomposition The straightforward way is to model it as a time-indexed decomposition, as stated in the constraint (3). Since the size of the decomposition also depends on the size of the planning horizon, the resulting model size quickly became prohibitive, which was supported by bad results in preliminary experiments. Thus, we did not consider this option further.

Global Constraints The standard way in CP is to use the global constraint `disjunctive` or `cumulative`, if the first one is not supported by the solver, for each resource and re-using the variables y_{ir}^{s} for modeling the optionality of each activity on the resource. The solver `Chuffed` does not support the constraint `disjunctive` in the form that is needed for MSPSP. Thus, we model the resource constraints by `cumulative`.

$$\texttt{cumulative}((s_i)_{i \in V, s \in S}, (p_i)_{i \in V, s \in S}, (y_{ir}^{s})_{i \in V, s \in S}, 1) \qquad \forall r \in R$$

The parameters of `cumulative` represent the start times of activities, the durations of activities, the resource requirements of activities, and the resource capacity. This resource model creates nl "optional" tasks with a variable resource requirement, which equals to 1 if the activity uses a skill of the resource; otherwise it is 0 and the activity is absent.

This model choice works better than the time-indexed decomposition, and can be further improved as the number of "optional" tasks created can be reduced to n by introducing auxiliary variables x_{ir} for each activity $i \in V$ and each resource $r \in R$. These variables are reflecting the fact whether a resource r contributes with any of its skills to the execution of the activity i.

$$y_{ir}^{s} \leq x_{ir} \qquad \forall i \in V, r \in R, s \in S \qquad (6)$$
$$\texttt{cumulative}((s_i)_{i \in V}, (p_i)_{i \in V}, (x_{ir})_{i \in V}, 1) \qquad \forall r \in R \qquad (7)$$

Constraint (6) links the auxiliary variables to the resource-skill assignment variables, whereas (7) uses these auxiliary variables as resource requirement ones in the constraint `cumulative`. Note that these cumulative constraints will perform the same propagation when using the variables y_{ir}^{s}, but the learning would be stronger, because it relaxes the exact skill contribution for the resources.

Order Constraints Instead of using the `cumulative` constraint, which provides the strongest propagation on the start time variables and lets the solver learn about the time resource usage connection, we can use ordering constraints that enforce a non-overload of any resource. In order to reduce the number of those ordering constraints, we introduce the concept of *unrelated* activity pairs.

Let $clo(E)$ be the transitive closure of the set of the precedence relations in E, *i.e.*, $\forall (i,j) \in clo(E), \exists (k_1, k_2), (k_2, k_3), \ldots, (k_o, k_{o+1}) \in E, o \geq 1$ with $k_1 = i$ and $k_{o+1} = j$. Two activities i and j are *unrelated* if $(i,j), (j,i) \notin clo(E)$. Let $U = \{(i,j) \mid i, j \in V, i < j, \{(i,j),(j,i)\} \cap clo(E) = \emptyset\}$ be the set of all unrelated activity pairs. Only the execution of unrelated activities can overlap in time; all others cannot due to the precedence relations. Even if unrelated activities run concurrently they might not cause a resource overload if different resources are assigned to them. For each pair of unrelated activities $(i, j) \in U$, we introduce an order variable $o_{ij} \in \{0, 1\}$, which takes the value 1 if their executions are overlapping and otherwise 0.

$$\neg o_{ij} \Leftrightarrow (s_i + p_i \leq s_j) \vee (s_j + p_j \leq s_i) \qquad \forall (i, j) \in U \tag{8}$$

$$(x_{ir} \wedge x_{jr}) \Rightarrow \neg o_{ij} \qquad \forall (i, j) \in U, r \in R \tag{9}$$

Constraint (8) uses an equality for linking the order variables to whether one of the activity runs before the other one. Constraint (9) enforces when two unrelated activities are assigned to the same resource that then their execution cannot overlap. We note that the order variables allow the solver to learn about the relative position of activities, which is stronger than the time-dependent learning that happens with the above described models. Thus, the stronger learning might compensate the weaker propagation.

Moreover, for some unrelated activities, we may know a priori that they cannot overlap due their combined requirements of a given skill exceeding the number of available resources mastering that skill. For such unrelated activities, we can replace the corresponding constraints in (8) and (9) by this conjunction.

$$(o_{ij} \Rightarrow s_i + p_i \leq s_j) \wedge (\neg o_{ij} \Rightarrow s_j + p_j \leq s_i)$$
$$\forall (i, j) \in U, \exists s \in S : sr_i^s + sr_j^s > \sum\nolimits_{r \in R} mast_r^s \tag{10}$$

4.3 Redundant Constraints

Similar to the valid inequalities presented in [8], we add redundant constraints to enhance our formulation. Since the assignment of activities to resources is unknown at the beginning of any search, the unary resource constraints will propagate poorly, potentially resulting in poor early search decisions, which are hard to recover for any solver. Two ways to allow more propagation earlier in the search is to relax the skill from the resources and the resource from the skills.

$$\texttt{cumulative}((s_i)_{i \in V}, (p_i)_{i \in V}, (sr_i^s)_{i \in V}, |\{r \in R \mid mast_r^s = 1\}|) \qquad \forall s \in S \tag{11}$$

$$\texttt{cumulative}((s_i)_{i \in V}, (p_i)_{i \in V}, (\sum\nolimits_{s \in S} sr_i^s)_{i \in V}, |R|) \tag{12}$$

Constraint (11) ensures that at any time no more than the available number of resources having a particular skill are taken, whereas (12) states that at any time no more than the available resources can be used.

4.4 Search Procedures

Preliminary experiments showed that the basic search procedures, *i.e.*, `int_search` and `bool_search`, in combination with the compositional procedure `seq_search` in MiniZinc performed poorly, because of the disconnection of branching on the resource assignment and start time variables related to one activity. However,

Table 3: Data sets summary

					Best Known Results	
set	#instances	n	l	m	source	%optimal
1a	216	22	4	10-30	[8]	93.98
1b	216	42	4	20-60	[2]	2.31
2a	110	20-51	2-8	5-14	[14]	43.64
2b	77	32-62	9-15	5-19	[14]	66.20
2c	91	22-32	3-12	4-15	[14]	51.11

we made use of the new search facilities of MiniZinc `priority_search`, which is supported by `Chuffed`. We grouped the start time variables s_i and the resource-skill assignment variables y^s_{ir} by activities $i \in V$. The search procedures then branch over each activity group. For each group, the search assigns the smallest possible start time of s_i before assigning the maximal value of its corresponding resource skill variables y^s_{ir} in input order. We investigate three different branching orders of activity groups, which depend on the domain of their start time variables s_i. The first one `priority-sm` picks the group having the smallest value in the domain of s_i, the second one `priority-sml` the smallest largest value, and the last one `priority-ff` the smallest domain size.

5 Computational Experiments

All experiments were run on a PC with an Intel i7 2600 CPU 3.4GHz and 8GB of memory. The model was compiled to the `Chuffed` FlatZinc format using MiniZinc 2.1.2. We used `Chuffed` [6] from `https://github.com/chuffed` (branch: develop, commit 1f37fde). All experiments were run with a time limit of 10 minutes. We evaluated our models and search strategies on two data sets detailed in Tab. 3. Set 1 was proposed in [8,1], but we were unable to get in contact with the authors and access their data sets. Thus, we created a new data set $1'$ using the same instance generator and parameters as they did. [14] selected a subset of 271 instances from the available 278 MSPSP instances in set 2. To the best of our knowledge, their exact method provides the best results for this subset. We run preliminary experiments on set $1'a$ to inform our decision on the best model formulation and choice of search strategy.

Table 4 presents a comparison between the different unary constraints and the search strategies tested on set $1'a$ as well as the influence of the redundant constraints (11) and (12). We used the time-tabling filtering as described in [21] for all `cumulative` constraints (7), (11–12) and, additionally, the time-tabling-edge-finding filtering [24] for (12). The base model for this comparison includes constraints (1), (2), and (4–6). First each unary constraint formulation is tested with the default search, *i.e.*, an activity-based search with restarts. Constraint (10) is superior so we then test it again using the search strategies from Sect. 4.4, which are alternated by the default search on restarts. We conclude that the

Table 4: Model and search comparison – set $1'a$

unary cons.	redundant cons.	search	#nodes	%optimal	runtime
(7)	(11–12)	default	370,174	100.00	10.23s
(8–9)	(11–12)	default	97,085	100.00	2.73s
(8–10)	(11–12)	default	54,282	100.00	1.30s
(8–10)	(11–12)	`priority-ff`	41,762	100.00	1.25s
(8–10)	(11–12)	`priority-sml`	20,786	100.00	0.68s
(8–10)	(11–12)	`priority-sm`	13,241	100.00	**0.51s**
(8–10)	(11)	`priority-sm`	847,879	85.19	94.81s
(8–10)	(12)	`priority-sm`	13,953	100.00	0.67s

Table 5: Benchmark results – set $1'b$ and set 2

set	#nodes	%gap	#opt	%opt	mean opt runtime	mean runtime
$1'b$	7,584,577	49.32	27/216	12.50	77.12s	534.64s
$2a$	2,223,060	185.20	81/110	73.64	50.29s	195.22s
$2b$	816,068	22.42	63/77	81.82	16.88s	122.90s
$2c$	14,035	0.00	91/91	100.00	1.20s	1.20s

priority-based search is superior to `Chuffed`'s default search and that prioritizing the activity selection by smallest possible start time is best. If we remove one of the redundant constraints then the performance deteriorates, especially when constraint (12) is removed. Constraints (8–12) using the `priority-sm` search was used for all remaining experiments.

Table 5 presents the results of testing the CP model on all remaining benchmark instances. We include the mean nodes explored, mean optimality gap for unsolved instances, optimal solutions found, mean runtime on solved instances and the mean runtime across all instances. The gap has been calculated using a naïve lower bound defined by the length of the critical path in the precedence graph. The results of set $2a$ make it clear that this is a very loose bound.

The CP model performed well on the instances of set 2 previously tackled by the literature as we see that all instances of set $2c$ have been solved to optimality in an average of 1.20 seconds. Previously, half of this set was unsolved.

6 Conclusion

We have investigated different CP models and searches for the MSPSP and evaluated them on 710 instances. These models were tailored for CP solvers deploying nogood learning, for which the best trade-off between propagation and learning strength needs to be found. The key ingredients for a successful solution procedure was the combination of problem tailored search with the use of redundant resource constraints and learning on the order between activities by not modeling unary resources with global constraints. The CP solver `Chuffed` was able to optimally solve 67.3% of the instances with a time limit of 10 minutes. In total, we closed at least 87 open instances from the literature.

References

1. Almeida, B.F., Correia, I., Saldanha-da Gama, F.: An instance generator for the multi-skill resource-constrained project scheduling problem (2015), `https://ciencias.ulisboa.pt/sites/default/files/fcul/unidinvestig/cmaf-cio/SGama.pdf`, last accessed on 26 April 2017
2. Almeida, B.F., Correia, I., Saldanha-da Gama, F.: Priority-based heuristics for the multi-skill resource constrained project scheduling problem. Expert Systems with Applications 57, 91–103 (2016)
3. Artigues, C., Demassey, S., Néron, E.: Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. ISTE/Wiley (2008)
4. Bellenguez-Morineau, O.: Methods to solve multi-skill project scheduling problem. 4OR 6(1), 85–88 (2008)
5. Bellenguez-Morineau, O., Néron, E.: A branch-and-bound method for solving multi-skill project scheduling problem. RAIRO - Operations Research 41(2), 155–170 (6 2007)
6. Chu, G.G.: Improving Combinatorial Optimization. Ph.D. thesis, The University of Melbourne (2011), `http://hdl.handle.net/11343/36679`
7. Correia, I., Saldanha-da Gama, F.: A modeling framework for project staffing and scheduling problems. In: Schwindt, C., Zimmermann, J. (eds.) Handbook on Project Management and Scheduling, Vol. 1, pp. 547–564. Springer International Publishing (2015)
8. Correia, I., Loureno, L.L., Saldanha-da Gama, F.: Project scheduling with flexible resources: Formulation and inequalities. OR Spectrum 34(3), 635–663 (2012)
9. Dhib, C., Kooli, A., Soukhal, A., Néron, E.: Lower bounds for a multi-skill project scheduling problem. In: Klatte, D., Lüthi, H.J., Schmedders, K. (eds.) Operations Research Proceedings 2011: Selected Papers of the International Conference on Operations Research (OR 2011), August 30 - September 2, 2011, Zurich, Switzerland. pp. 471–476. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
10. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. Working Paper Series 02/2008, Hamburg School of Business Administration (HSBA) (2008), `https://ideas.repec.org/p/zbw/hsbawp/022008.html`
11. Kolisch, R., Sprecher, A.: Psplib - a project scheduling problem library. European Journal of Operational Research 96(1), 205 – 216 (1997)
12. Kreter, S., Schutt, A., Stuckey, P.J.: Modeling and solving project scheduling with calendars. In: Pesant, G. (ed.) Proceedings of Principles and Practice of Constraint Programming – CP 2015. pp. 262–278. Springer International Publishing, Cham (2015)
13. Montoya, C.: New methods for the multi-skills project scheduling problem. Ph.D. thesis, Ecole des Mines de Nantes (2012)
14. Montoya, C., Bellenguez-Morineau, O., Pinson, E., Rivreau, D.: Branch-and-price approach for the multi-skill project scheduling problem. Optimization Letters 8(5), 1721–1734 (2014)
15. Montoya, C., Bellenguez-Morineau, O., Pinson, E., Rivreau, D.: Integrated column generation and lagrangian relaxation approach for the multi-skill project scheduling problem. In: Schwindt, C., Zimmermann, J. (eds.) Handbook on Project Management and Scheduling, Vol. 1, pp. 565–586. Springer International Publishing (2015)

16. Néron, E.: Lower bounds for the multi-skill project scheduling problem. In: Proc. 8th International Workshop on Project Management and Scheduling. pp. 274–277 (2002)
17. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming – CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings. pp. 529–543. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
18. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints 14(3), 357–391 (2009)
19. Schutt, A., Chu, G., Stuckey, P.J., Wallace, M.G.: Maximising the net present value for resource-constrained project scheduling. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems. pp. 362–378. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
20. Schutt, A., Feydy, T., Stuckey, P.J.: Scheduling optional tasks with explanation. In: Schulte, C. (ed.) Proceedings of Principles and Practice of Constraint Programming – CP 2013. pp. 628–644. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
21. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. Constraints 16(3), 250–282 (2011)
22. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving rcpsp/max by lazy clause generation. Journal of Scheduling 16(3), 273–289 (2013)
23. Schutt, A., Stuckey, P.J., Verden, A.R.: Optimal carpet cutting. In: Lee, J. (ed.) Proceedings of Principles and Practice of Constraint Programming – CP 2011. pp. 69–84. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
24. Schutt, A., Feydy, T., Stuckey, P.J.: Explaining time-table-edge-finding propagation for the cumulative resource constraint. In: Gomes, C.P., Sellmann, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Lecture Notes in Computer Science, vol. 7874, pp. 234–250. Springer Berlin Heidelberg (2013)
25. Szeredi, R., Schutt, A.: Modelling and solving multi-mode resource-constrained project scheduling. In: Rueher, M. (ed.) Proceedings of Principles and Practice of Constraint Programming – CP 2016. pp. 483–492. Springer International Publishing, Switzerland, Cham (2016)
26. Węglarz, J., Józefowska, J., Mika, M., Waligóra, G.: Project scheduling with finite or infinite number of activity processing modes a survey. European Journal of Operational Research 208(3), 177–205 (2011)