An Update on the Comparison of MIP, CP and Hybrid Approaches for Mixed Resource Allocation and Scheduling

Philippe Laborie

IBM, 9 rue de Verdun 94250 Gentilly, France
laborie@fr.ibm.com

Abstract. We consider a well known resource allocation and scheduling problem for which different approaches like mixed-integer programming (MIP), constraint programming (CP), constraint integer programming (CIP), logic-based Benders decompositions (LBBD) and SAT-modulo theories (SMT) have been proposed and experimentally compared in the last decade. Thanks to the recent improvements in CP Optimizer, a commercial CP solver for solving generic scheduling problems, we show that a standalone tiny CP model can out-perform all previous approaches and close all the 335 instances of the benchmark. The article explains which components of the automatic search of CP Optimizer are responsible for this success. We finally propose an extension of the original benchmark with larger and more challenging instances.

Keywords: Constraint Programming, Resource Allocation, Scheduling, CP Optimizer

1 Introduction

We consider the well known resource allocation and scheduling problem proposed in [6]. After recapping the problem definition (Sect. 2) and giving an overview of the state-of-the-art methods that have been proposed for solving it (Sect. 3), we present a very concise formulation of the problem in CP Optimizer (Sect. 4). Experimental results (Sect. 5) show that, using this model together with a parameter focusing the search on optimality proofs, CP Optimizer 12.7.1 outperforms all existing approaches and closes all the 335 instances of the benchmark. We finally propose an extension of the original benchmark (Sect. 6) with larger and more challenging instances.

2 Problem Definition

The problem proposed in [6] is defined by a set of jobs $\mathcal{J}$ and a set of facilities $\mathcal{I}$. Each job $j \in \mathcal{J}$ must be assigned to a facility $i \in \mathcal{I}$ and scheduled to start after its release date r_j, end before its due date d_j, and execute for p_{ij} consecutive time units. Each job j has a facility assignment cost f_{ij} and a resource requirement

c_{ij} when allocated to facility i. Each facility $i \in \mathcal{I}$ has a capacity C_i and the constraint that the resource capacity must not be exceeded at any time. The problem is to minimize the total facility assignment cost.

A time-indexed MIP formulation of the problem where Boolean variable x_{ijt} is one if job j starts at discrete time t on facility i is:

$$\begin{aligned}
\min \; & \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} \sum_{t\in\mathcal{T}} f_{ij} x_{ijt} & \\
\text{s.t.} \; & \sum_{i\in\mathcal{I}} \sum_{t\in\mathcal{T}} x_{ijt} = 1, & \forall j \in \mathcal{J} \\
& \sum_{j\in\mathcal{J}} \sum_{t'\in\mathcal{T}_{ijt}} c_{ij} x_{ijt'} \leq C_i, & \forall i \in \mathcal{I}, t \in \mathcal{T} \\
& x_{ijt} = 0, & \forall i \in \mathcal{I}, j \in \mathcal{J}, t \in \mathcal{T}, \\
& & t < r_j \text{ or } t > d_j - p_{ij} \\
& x_{ijt} \in \{0,1\}, & \forall i \in \mathcal{I}, j \in \mathcal{J}, t \in \mathcal{T}.
\end{aligned}$$

In the above formulation, $\mathcal{T}_{ijt}$ denotes the set of time values t' such that if job j is allocated to resource i and starts at t' then it is executing at time t:

$$\mathcal{T}_{ijt} = \{t' \in \mathcal{T} | t - p_{ij} < t' \leq t\}$$

3 State of the Art

Although the problem can be considered as being *simple* compared to many real-world scheduling problem, it is clearly of interest because it mixes resource allocation and scheduling and it is NP-Hard in the strong sense. This problem has received a lot of attention in the combinatorial optimization community and several approaches have been studied and evaluated on the benchmark proposed in [6, 7]. We give a short review of these approaches below.

Mixed Integer Programming (MIP). The time-indexed formulation described in Sect. 2 is often used as a baseline method to compare with [7, 4]. As reported in [1], other MIP formulations were tried (like the event based ones, see [13] for a review) but on this particular problem, time-indexed formulations seem to perform better. In [5], a time-indexed MIP model using some redundant formulation was used and the reported results are, to the best of our knowledge, the best MIP results on the benchmark.

Constraint Programming (CP). Results using an OPL model on top of ILOG Scheduler were reported in [6, 7]. In [4] the authors used an early version of IBM ILOG CP Optimizer (V2.3). More recently [12] has been using IBM ILOG CP Optimizer 12.7 but the details of the model are not provided. All the experiments so far suggested that standalone CP is not really competitive with hybrid methods or even a standalone MIP.

Logic-based Benders decomposition (LBBD). An LBBD approach was originally proposed in [6, 7] and shown to outperform both CP and MIP. In this approach, the master problem consists of the facility allocation problem whereas sub-problems consist in scheduling the jobs given a facility allocation provided by the master problem. A relaxation of the scheduling problem inspired from energetic reasoning [2] is also included in the master problem. In the original work, the scheduling sub-problems were solved using ILOG Scheduler. These results were updated in [1] by using IBM ILOG CP Optimizer (V12.4) for solving the scheduling sub-problems. Other results with LBBD on top of CP (denoted LBBD-CP in this paper) are also reported in [5].

Constraint Integer Programming (CIP). CIP is a resolution paradigm that exploits both the constraint propagation traditionally used in CP and the linear relaxations used in MIP in the same search tree. CIP models were studied in [3–5] showing that they were competitive with LBBD.

Satisfiability Modulo Theories (SMT). More recently, an SMT approach combining SAT with the theory of real arithmetics has been studied in [12] for solving the problem, together with a more efficient hybrid LBBD approach that uses SMT for solving the sub-problems (denoted LBBD-SMT).

4 CP Optimizer Model

CP Optimizer extends classical CP on integer variables with a few mathematical concepts (intervals, functions) that make it easier to model scheduling problems while providing interesting problem structure for its automatic search algorithm [10]. The complete CP Optimizer formulation of the resource allocation and scheduling problem proposed in [6] is shown on Fig. 1 (using OPL).

Lines 2-8 read data using similar notations as the ones introduced in Sect. 2. Line 10 creates one interval variable `job[j]` for each job j that is constrained to start after r_j and end before d_j. For each possible allocation of a job j on a facility i, line 11 creates an optional interval variable `mode[i][j]` of size p_{ij}. This interval variable will be present if and only if job j is allocated to facility i. The objective at line 13 is to minimize the weighted sum of the selected modes. Constraints on line 15 state that a job j is allocated to one facility: only one of the interval variables `mode[i][j]` is present and the start and end value of interval `job[j]` are the same as the ones of the selected mode. Finally at line 16, a cumul function expression on the left hand side represents the facility usage over time and is constrained to be lower than the facility capacity C_i.

5 Results

5.1 Benchmark Description

The benchmark proposed in [7] is composed of 4 families of instances (c,e,de,df) with slightly different characteristics resulting in a total of 335 instances. For

```
using CP;
tuple ModeData { int p; int c; int f; }
tuple JobData { int r; int d; }
int n = ...; range J = 1..n;
int m = ...; range I = 1..m;
ModeData M[I,J] = ...;
JobData Job[J] = ...;
int C[I] = ...;

dvar interval job[j in J] in Job[j].r..Job[j].d;
dvar interval mode[i in I][j in J] optional size M[i][j].p;

minimize sum(i in I, j in J) (M[i][j].f * presenceOf(mode[i][j]));
subject to {
  forall(j in J) { alternative(job[j], all(i in I) mode[i][j]); }
  forall(i in I) { sum(j in J) pulse(mode[i][j],M[i][j].c) <= C[i]; }
}
```

Fig. 1. Complete CP Optimizer model

all instances the cost is such that faster facilities tend to be more expensive. In each families there are 5 instances for every combination (n, m) where n is the number of jobs and m the number of facilities.

Results in this section were obtained on an IBM blade with 20GB RAM and an Intel® Xeon® X5570 2.93GHz running GNU/Linux with CP Optimizer V12.7.1. Unless stated otherwise, we are using 4 parallel workers (`Workers=4`).

5.2 Experimental Evaluation of CP Optimizer Search Components

As explained in [10], the automatic search of CP Optimizer consists of two components run in concurrence: (1) a Large Neighborhood Search (LNS) heuristic that tries to improve the current solution by successively unfreezing and re-optimizing a part of the solution [8] and (2) a Failure-directed Search (FDS) that aims at proving optimality of the current solution [15].

CP Optimizer targets in priority industrial scheduling problems that are in general much larger than the ones of the benchmark. This explains why, by default, the automatic search spends more effort on LNS than on FDS but, as explained in [15], this behavior can be changed thanks to a search parameter `FailureDirectedSearchEmphasis`.

In a preliminary experimental study, using a 1h time limit, we compare three variants of the CP Optimizer search: the default version (`default`), a version with a strong focus on FDS with 3.5 out of the 4 workers dedicated to FDS[1] (`fds`) and a version that do not use FDS[2] (`no_fds`). Results are shown on Table 1.

First, it is to be noted that, whatever variant is used, CP Optimizer finds an initial feasible solution or prove infeasibility[3] for all 335 instances of the bench-

[1] Using `FailureDirectedSearchEmphasis=3.5`.

[2] Using `FailureDirectedSearch=Off`.

[3] All instances of the benchmark are feasible except for 5 instances of the `de` family.

Table 1. Comparison of different CP Optimizer variants.

	no_fds	default	fds
Number of feasible solutions found	335/335	335/335	335/335
Maximal time for finding a feasible solution	0.33s	0.33s	0.33s
Number of optimal solutions found	305/335	320/335	334/335
Maximal gap of best solution found	1.18%	0.92%	0.16%
Number of optimality proofs	143/335	309/335	330/335

mark in less than 0.5s. This is to be compared with LBBD approaches that, by construction, do not provide any feasible solution before the optimal solution is eventually found. For 309 instances, the `default` version finds the optimal solution and proves optimality, for the remaining 26 instances, the gap with respect to the optimal solution is less than 1%. The main difference between the 3 variants concerns the capability to prove optimality: as expected, the more FDS is used and the more optimality proofs we get. Within the 1h time limit, the variant with a strong focus on FDS closes all but 5 instances of the benchmark. In next section, we compare the results of this variant with state-of-the-art approaches.

5.3 Comparison with Previous Results

Table 2 compares the results of CP Optimizer 12.7.1 (with a focus on optimality proofs through FDS) on the most challenging instances of the 'c' family using a time limit of 1h with the best known results compiled for the main four approaches (MIP, LBBD-CP, LBBD-SMT, CIP-CP). For MIP and LBBD-CP, we compare with the results of [5] that are on average better than the ones reported in [7, 1, 12]. For LBBD-SMT we compare with the results of [12][4] and for CIP-CP with the ones of [5]. As in [5], the columns `geom` denote the shifted geometrical mean with a shift of 10s over the 5 instances, taking any time lower than 1s as being equal to 1s. Experiments in [5] were run using a single thread with a 2h time-limit on a slightly slower machine (2.50 GHz v.s. 2.93GHz) so for a more fair comparison we also provide results of our model using one worker but still using a 1h time-limit. CP Optimizer easily solves all the instances of the other families.

As we see, CP Optimizer generally outperforms existing approaches, both in terms of number of instances solved to optimality and in terms of solve time. With 130 instances solved out of 135, it is better than the virtual best solver of the 4 approaches studied in [5] (127 solved instances). In fact, the 5 remaining open problems could be closed with the same model by using a larger time limit (up to 160h for the hardest one). Detailed list of optimal costs can be found at `http://ibm.biz/AllocSched`.

We think that an important ingredient of the success of Failure-Directed Search on these instances is its capability to opportunistically mix allocation

[4] Computed from the detailed results the authors gratefully sent us.

Table 2. Comparison with state-of-the-art approaches.

		MIP		LBBD-CP		LBBD-SMT		CIP-CP		CPO (fds) 1W		CPO (fds) 4W	
#I	#J	opt	geom	opt	geom	opt	geom	opt	geom	opt	geom	opt	geom
2	16	5	8.0	5	**1.0**	5	2.82	5	4.7	5	1.05	5	**1.00**
	18	5	16.9	5	**1.3**	5	1.64	5	1.7	5	2.72	5	1.60
	20	5	29.0	5	3.7	5	1.47	5	**1.5**	5	2.01	5	1.62
	22	*4*	*812.4*	5	51.4	5	72.06	*3*	*382.5*	5	**7.37**	5	**4.06**
	24	*3*	*883.0*	*4*	*214.8*	5	196.72	*2*	*573.4*	5	**10.17**	5	**6.54**
	26	*4*	*1069.2*	5	209.0	*3*	*554.03*	*4*	*464.9*	5	**22.33**	5	**11.28**
	28	*4*	*378.9*	5	536.5	*4*	*38.58*	*4*	*42.0*	5	**48.39**	5	**17.00**
	30	*3*	*861.2*	*3*	*401.2*	*1*	*1147.84*	*2*	*587.6*	**5**	**122.21**	**5**	**92.30**
	32	*3*	*792.1*	*0*	-	*3*	*332.85*	*2*	*1140.5*	**5**	**235.81**	**5**	**120.14**
	34	*3*	*879.7*	*2*	*1745.1*	*3*	*509.45*	*1*	*1995.3*	*3*	***419.24***	*3*	***253.09***
	36	*2*	*1534.1*	*1*	*4770.2*	*2*	*450.68*	*3*	***548.4***	*3*	*1199.91*	*3*	***491.11***
	38	*2*	*4980.2*	*1*	*5848.7*	*4*	*428.51*	*2*	*1334.0*	*4*	***390.86***	*4*	***127.07***
3	18	5	46.0	5	5.8	5	2.43	5	4.8	5	**2.08**	5	**1.56**
	20	*4*	*98.5*	5	1.5	5	**1.33**	5	6.9	5	2.21	5	1.75
	22	*4*	*554.6*	5	2.3	5	**2.17**	5	6.6	5	4.55	5	2.90
	24	5	304.5	5	**6.7**	5	9.41	5	78.6	5	11.05	5	**6.24**
	26	*3*	*1652.8*	5	**19.8**	5	44.50	5	40.2	5	21.71	5	**10.28**
	28	*3*	*987.6*	5	35.4	5	70.54	*3*	*194.9*	5	**34.14**	5	**15.13**
	30	*3*	*3100.2*	*4*	*178.3*	*3*	*540.18*	*4*	*520.9*	5	**158.48**	5	**54.17**
	32	*2*	*3601.3*	*4*	*1951.8*	*2*	*665.42*	*3*	*559.0*	5	**220.02**	5	**117.26**
4	20	5	25.3	5	1.8	5	**1.15**	5	4.3	5	1.65	5	**1.09**
	22	5	60.0	5	3.7	5	**2.48**	5	15.0	5	3.22	5	**2.29**
	24	*4*	*1399.0*	5	12.1	5	19.22	5	42.9	5	**9.58**	5	**4.95**
	26	*3*	*2787.8*	5	**14.9**	5	17.12	5	112.7	5	20.56	5	**12.44**
	28	*3*	*2124.2*	5	**9.6**	5	29.15	5	200.0	5	17.30	5	10.32
	30	*2*	*3253.6*	5	**31.7**	5	110.23	5	581.1	5	153.54	5	53.10
	32	*1*	*4691.0*	5	118.3	5	450.84	5	1519.1	5	**93.22**	5	**44.09**

and scheduling decisions in the same decision tree based on its decision rating system. In the CP Optimizer model, both allocation decisions (presence status of interval variables) and scheduling decisions (interval variables start and end values) hold on the same decision variables that are efficiently pruned by constraint propagation on optional intervals. The search space explored by the FDS is reduced by using strong pruning (like timetable edge finding [14]) and efficient propagation of conditional bounds on alternative constraints [11].

6 Benchmark Extension

Given the progress achieved until now on the original benchmark, we propose an extension with more challenging instances. In particular we want to address the following limitations:

- The current instances are small (up to 50 jobs and 10 facilities) and not really representative of the typical size of actual scheduling problems. We propose to generate new problems with a size up to 1000 jobs and 20 facilities.
- The time granularity is coarse, as the maximal duration of jobs is less than a few tens of units. We propose a grain 100 times finer.
- On a similar line, the granularity of facilities capacity (maximal capacity is 10) is also made 100 times finer.
- In the current benchmark, a given job j always requires the same quantity c_{ij} of the different facilities i. In our extension they can use different capacity. Instead of being roughly inversely proportional to the job duration, the facility cost is roughly inversely proportional to the job energy (product of the duration by the demanded quantity).
- As in the original version of the benchmark, we generated some precedence constraints between jobs. These precedences can be used optionally. Precedence constraints are ubiquitous in real-life scheduling but, as highlighted in [5], they destroy the independent sub-problem structure that LBBD and, to a lesser extend, the other approaches (CP Optimizer apart) exploit.

In the new instances, the capacity of each facility is 1000. As in the existing instances, we suppose that the facilities get in average slower as their index i increases. The duration p_{ij} is drawn uniformly from $[100\sqrt{i}, 1000\sqrt{i}]$. The required capacity c_{ij} is drawn uniformly from $[1, 1000]$. The cost f_{ij} for executing job j on machine i is roughly proportional to the inverse of the energy $e_{ij} = p_{ij}c_{ij}$ with a variability α_j that depends on the job j and is drawn uniformly from $[0.5, 1]$. More precisely, the cost f_{ij} is drawn from $[\alpha_j F_{ij}, F_{ij}]$ where $F_{ij} = 10^7\sqrt{m}/e_{ij}$. As for the original instances of the 'c' family, the release dates are all 0 and the deadlines are all equal to αL where $\alpha = 1/3$ and $L = \sum_{ij} p_{ij}/m^2$ is the average total processing time per facility. A set of precedence constraints can also optionally be considered: we generated $n/2$ precedence constraints between random pairs of predecessor/successor jobs while ensuring the resulting precedence graph is acyclic. We selected 20 different combinations $(n, m) \in [20, 1000] \times [2, 20]$. As in the original benchmark, 5 instances are generated for each different combinations (n, m) they are denoted `f`n`j`m`m`k`.dat` for $k \in [1, 5]$, so the benchmark extension consists of 100 new instances. In this paper we only report results without precedence constraints. The new benchmark is available at `http://ibm.biz/AllocSched` together with the detailed results discussed in this section as well as results with precedence constraints.

We experimented with 3 variants of the CP Optimizer search with a time limit of 1h: the one with a focus on FDS (col. `fds`), the default search without any parameter change (col. `default`) and a version not using the temporal linear relaxation [9] (col. `no_tlr`).[5] The results are summarized on Table 3. We see that the first feasible solutions are produced, even for the largest instances, in less than 1s (column `fst`). Some optimality proofs could be provided only for the smallest instances with 20 or 30 jobs and 2 facilities. The `s_gap` columns (for 'scheduling gap') measure the average gap of the produced solutions with

[5] Using `TemporalRelaxation=Off`.

respect to the lower bound of a MIP formulation of the allocation part of the problem with energetic resource relaxation (this is basically the MIP of the initial iteration of the master problem in LBBD approaches). As we see for problems with 20 jobs and 2 facilities for which CP Optimizer proves optimality of the global problem, this gap is significant (21.8%) meaning that the allocation part of the problem does not dominate the scheduling part.[6] As expected, when the size of the problem grows, the FDS becomes less useful and the performance is better when not using it.[7] Interestingly, for these problems, the temporal linear relaxation does not seem to be very helpful and, for the largest instances, it is penalized by the cost of running the LP relaxation at the root node of LNS moves. This would deserve a more detailed analysis.

Table 3. Comparison of CP Optimizer variants on the new instances.

		fst	fds		default	no_tlr			fst	fds		default	no_tlr
#J	#I	(s)	opt	s_gap	s_gap	s_gap	#J	#I	(s)	opt	s_gap	s_gap	s_gap
20	2	0.0	5	21.8%	21.8%	21.8%	200	10	0.1	0	20.4%	18.2%	**16.8%**
30	2	0.0	1	**10.6%**	11.0%	11.3%	200	20	0.2	0	25.2%	23.3%	**20.2%**
40	2	0.0	0	10.7%	8.9%	**8.7%**	500	2	0.1	0	**12.8%**	13.2%	14.1%
50	2	0.0	0	11.3%	9.8%	**9.0%**	500	5	0.1	0	**19.8%**	22.6%	20.3%
50	5	0.1	0	15.8%	**14.4%**	14.5%	500	10	0.2	0	27.0%	26.3%	**23.7%**
100	2	0.1	0	10.9%	8.5%	**8.1%**	500	20	0.4	0	33.2%	35.0%	**24.4%**
100	5	0.1	0	14.7%	13.1%	**13.0%**	1000	2	0.2	0	18.8%	19.2%	**17.8%**
100	10	0.1	0	18.5%	16.3%	**15.3%**	1000	5	0.3	0	30.6%	28.1%	**22.8%**
200	2	0.1	0	14.5%	9.8%	**9.3%**	1000	10	0.5	0	35.4%	36.2%	**33.4%**
200	5	0.1	0	17.2%	14.4%	**13.5%**	1000	20	1.0	0	35.7%	35.1%	**35.0%**

7 Conclusion

This paper provides an update on the comparison of different approaches for solving a well known allocation and scheduling problem. We show that with the recent advances in the automatic search algorithm of CP Optimizer, a standalone simple CP model outperforms all existing approaches. This simple declarative model allows to close the benchmark. We proposed an extension of the original benchmark with larger and more challenging instances for future research and provide a preliminary analysis of the results of CP Optimizer on the new instances.

[6] As a comparison, this scheduling gap is only 0.77% in average for the instances of the 'c' family with 20 jobs and 2 facilities.

[7] Note that FDS is automatically switched off for large problems. Here, it is not being used for problems with 500 and 1000 jobs.

References

1. Ciré, A., Çoban, E., Hooker, J.N.: Logic-based Benders decomposition for planning and scheduling: a computational analysis. The Knowledge Engineering Review 31(5), 440–451 (2016)
2. Erschler, J., Lopez, P.: Energy-based approach for task scheduling under time and resources constraints. In: Proc. 2nd international workshop on project management and scheduling. pp. 115–121 (1990)
3. Heinz, S., Beck, J.C.: Solving resource allocation/scheduling problems with constraint integer programming. In: Proc. Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2011). pp. 23–30 (2011)
4. Heinz, S., Beck, J.C.: Reconsidering Mixed Integer Programming and MIP-Based Hybrids for Scheduling. In: Proc. 9th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2012). pp. 211–227 (2012)
5. Heinz, S., Ku, W.Y., Beck, J.C.: Recent Improvements Using Constraint Integer Programming for Resource Allocation and Scheduling. In: Proc. 10th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2013). pp. 12–27 (2013)
6. Hooker, J.N.: A Hybrid Method for Planning and Scheduling. In: Proc. 10th International Conference on Principles and Practice of Constraint Programming (CP 2004). pp. 305–316 (2004)
7. Hooker, J.N.: Planning and scheduling by logic-based benders decomposition. Operations Research 55(3), 588–602 (2007)
8. Laborie, P., Godard, D.: Self-adapting large neighborhood search: Application to single-mode scheduling problems. In: Baptiste, P., Kendall, G., Munier-Kordon, A., Sourd, F. (eds.) Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2007). pp. 276–284. Paris, France (28-31 August 2007) (28-31 August 2007 2007)
9. Laborie, P., Rogerie, J.: Temporal Linear Relaxation in IBM ILOG CP Optimizer. Journal of Scheduling 19(4), 391–400 (2016)
10. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP Optimizer for Scheduling. Constraints Journal (2018)
11. Laborie, P., Rogerie, J.: Reasoning with Conditional Time-intervals. In: Proc. 21th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2008). pp. 555–560 (2008)
12. Mistry, M., D'Iddio, A.C., Huth, M., Misener, R.: Satisfiability Modulo Theories for Process Systems Engineering. Optimization Online (2017)
13. Tesch, A.: Compact MIP Models for the Resource-Constrained Project Scheduling Problem. Master's thesis, Technische Universität Berlin (2015)
14. Vilím, P.: Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources. In: Achterberg, T., Beck, J. (eds.) Proc. 8th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-2011). Lecture Notes in Computer Science, vol. 6697, pp. 230–245. Springer Berlin / Heidelberg (2011)
15. Vilím, P., Laborie, P., Shaw, P.: Failure-directed Search for Constraint-based Scheduling. In: Proc. 12th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2015). pp. 437–453 (2015)