Two Clause Learning Approaches for Disjunctive Scheduling

Mohamed Siala[1,2,3], Christian Artigues[2,4], and Emmanuel Hebrard[2,4]

[1] Insight Center for Data Analytics. University College Cork. Ireland
[2] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[3] Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[4] Univ de Toulouse, LAAS, F-31400 Toulouse, France
{siala, artigues, hebrard}@laas.fr

Abstract. We revisit the standard hybrid CP/SAT approach for solving disjunctive scheduling problems. Previous methods entail the creation of redundant clauses when lazily generating atoms standing for bounds modifications. We first describe an alternative method for handling lazily generated atoms without computational overhead. Next, we propose a novel conflict analysis scheme tailored for disjunctive scheduling. Our experiments on well known Job Shop Scheduling instances show compelling evidence of the efficiency of the learning mechanism that we propose. In particular this approach is very efficient for proving unfeasibility.

1 Introduction

Disjunctive scheduling refers to a large family of scheduling problems having in common the *Unary Resource Constraint*. This constraint ensures that a set of tasks run in sequence, that is, without any time overlap. The traditional constraint programming (CP) approaches for this problem rely on tailored propagation algorithms (such as *Edge-Finding* [6, 16, 23]) and search strategies (such as *Texture* [19]). The technique of *Large Neighborhood Search* [21] (LNS) was also extensively used in this context [7, 24].

A different type of approaches emerged recently, trading off strong propagation-based inference for a way to learn during search. For instance, a hybrid CP/SAT method, *Lazy Clause Generation* (LCG) [9, 17, 8] was shown to be extremely efficient on the more general *Resource Constrained Project Scheduling Problem* [20] (*RCPSP*). Even simple heuristic weight-based learning was shown to be very efficient on disjunctive scheduling [12, 10, 11]. The so called *light model* combines minimalist propagation with a slight variant of the *weighted degree* [5] heuristic.

In this paper, we propose a hybrid CP/SAT method based on this light model. Similarly to LCG, our approach mimics *CDCL* [15] with arbitrary propagators. However, it differs from LCG in two main respects: First, as the time horizon can be large, literals representing changes in the bounds of the tasks domains should be generated "lazily" during conflict analysis as it was proposed in [8]. However, handling domain consistency through clauses entails redundancies and hence suboptimal propagation. We use a dedicated propagation algorithm, running in constant amortized time, to perform this task. The second

contribution is a novel conflict analysis scheme tailored for disjunctive scheduling. This technique could be applied to any problem where search can be restricted to a predefined set of Boolean variables. This is the case of disjunctive scheduling since once every pair of tasks sharing the same resource is sequenced, we are guaranteed to find a complete solution in polynomial time. Most methods therefore only branch on the relative order of tasks sharing a common resource. We propose to use this property to design a different conflict analysis method where we continue *resolution* until having a nogood with only Boolean variables standing for task ordering. As a result, we do not need to generate domain atoms.

We compare the two methods experimentally and show that the benefit of not having to generate new atoms during search outweigh in many cases the more expressive language of literals available in traditional hybrid solvers. The novel approach is very efficient, especially for proving unfeasibility. We were able to improve the lower bounds on several well known *Job Shop Scheduling Problem* (*JSSP*) benchmarks. However, a method implemented within IBM CP-Optimizer studio recently found, in general, better bounds [24].

The rest of the paper is organized as follows: We give a short background on hybrid CP/SAT solving in Section 2. Next, we briefly describe the baseline CP model and strategies that we use in Section 3. We introduce in Section 4 our new lazy generation approach. In Section 5, we present our novel conflict analysis scheme. Last, we give and discuss the experimental results in Section 6.

2 Hybrid CP/SAT Solving

In this section, we briefly describe the basic mechanisms and notations of hybrid CP/SAT used in Lazy Clause Generation [17, 9], and in particular the notions of *backward* explanations and *lazy generation* introduced in [8].

Let $\mathcal{X} = [x_1, .., x_n]$ be a sequence of variables. A *domain* $\mathcal{D}$ maps every variable $x \in \mathcal{X}$ to a finite set $\mathcal{D}(x) \subset \mathbb{Z}$. $\mathcal{D}(x)$ is a *range* if $\max(\mathcal{D}(x)) - \min(\mathcal{D}(x)) + 1 = |\mathcal{D}(x)|$. A *constraint* C is defined by a relation over the domains of a sequence of variables. In this context, each constraint is associated to a *propagator* to reduce the domains with respect to its relation. Propagators are allowed to use only *domain operations* of a given type (often $[\![x = v]\!]$, $[\![x \neq v]\!]$, $[\![x \leq v]\!]$, and $[\![x \geq v]\!]$). Every domain operation is associated to a *literal* p that can have an *atom* (i.e. Boolean variable) generated "eagerly" before starting the search, or "lazily" when learning a conflict involving this change. Let p be a literal corresponding to a domain operation. $level(p)$ is the number of nodes in the current branch of the search tree when p becomes true (i.e., when this domain operation takes place). Moreover, $rank(p)$ is the number of such domain operations that took place at the same node of the search tree before p. Last, a literal p is associated to a logical implication of the form $\wp \Rightarrow p$ where $\wp$ is a conjunction of literals sufficient for a given propagator to deduce p. We call $\wp$ an *explanation* of p. We assume that the function $explain(p)$ returns the explanation $\wp$ for p. The explanations can be computed at the moment of propagation (i.e., *clausal* and

Algorithm 1: 1-UIP-with-Propagators

```
1 Ψ ← explain(⊥) ;
2 while |{q ∈ Ψ | level(q) = current level}| > 1 do
      p ← arg max_q({rank(q) | level(q) = current level ∧ q ∈ Ψ}) ;
3     Ψ ← Ψ ∪ {q | q ∈ explain(p) ∧ level(q) > 0} \ {p} ;
  return Ψ ;
```

forward explanations in [8]), or on demand during conflict analysis (*backward*). The notion of explanation is extended to a failure $\bot$ in the natural way.

Whenever a failure occurs during search, a conflict analysis procedure is used to compute a nogood, that is, an explanation of the failure which is not yet logically implied by the model. Algorithm 1 depicts the conflict analysis procedure based on the 1-UIP scheme [26]. The nogood Ψ is initialized in Line 1 to the explanation for the failure. It is returned once it contains a single literal that has been propagated at the current level (Line 2). When this is not the case, the last propagated literal is selected and its explanation is resolved with the current nogood in Line 3. Last, after computing the final nogood Ψ, a *backjump* to the level second to maximum propagation level of any literal in Ψ is performed and $\neg\Psi$ is stored as a learnt clause.

Consider now the lazy generation of atoms (i.e. after computing Ψ). When $[\![x \leq u]\!]$ has to be generated, the clauses $\neg[\![x \leq a]\!] \vee [\![x \leq u]\!]$; $\neg[\![x \leq u]\!] \vee [\![x \leq b]\!]$ where a and b are the nearest generated bounds to u with $a < u < b$ must be added to maintain the consistency of $[\![x \leq u]\!]$ with respect to previously generated atoms. In this case, the clause $\neg[\![x \leq l]\!] \vee [\![x \leq u]\!]$ becomes redundant.

3 A Simple CP Model for Job Shop Scheduling

A JSSP consists in sequencing the tasks from n jobs on m machines. The jobs $\mathcal{J} = \{J_i \mid 1 \leq i \leq n\}$ define sequences of tasks to be scheduled in a given order. Moreover, each of the m tasks of a job must be processed by a different machine. We denote T_{ik} the task of job J_i requiring machine M_k. Each task T_{ik} is associated to a processing duration p_{ik}. Let t_{ik} be the variable representing the starting time of task T_{ik}. The processing interval of a task T_{ik} is $[t_{ik}, t_{ik} + p_{ik})$. For all $k \in [1, m]$, the Unary Resource Constraint for machine M_k ensures that for any $i \neq j \in [1, n]$, there is no overlap between the processing interval of T_{ik} and T_{jk}. We use a simple decomposition with $O(n^2)$ Boolean variables δ_{kij} ($i < j \in [1, n]$) per machine M_k using the following Disjunctive constraints:

$$\delta_{kij} = \begin{cases} 0 \Leftrightarrow t_{ik} + p_{ik} \leq t_{jk} \\ 1 \Leftrightarrow t_{jk} + p_{jk} \leq t_{ik} \end{cases} \tag{1}$$

A JSSP requires also a total order on the tasks of each job. We enforce this order with simple precedence constraints. Last, the objective is to minimize the total scheduling duration, i.e., the makespan. To this purpose, we have an objective variable C_{max} subject to precedence constraints with the last task of each job.

We use the same search strategy as that proposed in [12, 10]. First, we compute a greedy upper bound on the makespan. Then we use a dichotomic search to reduce the gap between lower and upper bounds. Each step of this dichotomic phase corresponds to the decision problem with the makespan variable C_{max} fixed to the mid-point value. Each step is given a fixed time cutoff, and exceeding it is interpreted as the instance being unsatisfiable. Therefore, the gap might not be closed at the end of this phase, so a classic branch and bound procedure is used until either closing the gap or reaching a global cutoff.

We branch on the Boolean variables of the DISJUNCTIVE constraints following [12]. Recall that it is sufficient to have all these Boolean variables assigned to decide the problem. We use the weighed degree heuristic $taskDom/tw$ in a similar way to [12, 10] in addition to VSIDS [15]. As for the value ordering, we use an ordering that is based on the solution guided approach [3].

4 Lazy Generation of Atoms

In this section we describe an efficient propagator to maintain the consistency of lazily generated atoms. Recall that domain clauses are likely to be redundant in this case (Section 2). Instead of generating clauses to encode the different relationships between the newly generated atoms, we propose to encode such relations through a dedicated propagator in order to avoid this redundancy and hence the associated overhead. We introduce the DOMAINFAITHFULNESS constraint. Its role is twofold: firstly, it simulates *Unit-Propagation* (UP) as if the atoms were generated eagerly; secondly it performs a complete channeling between the range variable and all its generated domain atoms.

We consider only the lazy generation of atoms of the type $[\![x \leq u]\!]$ since all propagators in our models perform only bound tightening operations. Nevertheless, the generalization with $[\![x = v]\!]$ is quite simple and straightforward. Let x be a range variable, $[v_1, \ldots, v_n]$ be a sequence of integer values, and $[b_1 \ldots b_n]$ be a sequence of lazily generated Boolean variables s.t. b_i is the atom $[\![x \leq v_i]\!]$. We define DOMAINFAITHFULNESS$(x, [b_1 \ldots b_n], [v_1, \ldots, v_n])$ as follows: $\forall i, b_i \leftrightarrow x \leq v_i$.

We describe now the data structures and the procedures that allow to maintain Arc Consistency [4] (AC) with a constant amortized time complexity down a branch of the search tree. Algorithm 2 shows the $O(n)$ naive version.

Algorithm 2: AC(DOMAINFAITHFULNESS$(x, [b_1 \ldots b_n], [v_1, \ldots, v_n])$)

```
1 ub ← min(max(D(x)), min(v_i | D(b_i) = {1})) ;
2 lb ← max(min{D(x)}, 1 + max(v_i | D(b_i) = {0}));
3 if ub < lb then return ⊥ ;
4 D(x) ← D(x) ∩ [lb, +∞[ ;
5 D(x) ← D(x) ∩ ]−∞, ub] ;
6 for i ∈ [1, n] do
      if v_i ≥ ub then D(b_i) ← {1} ;
      if v_i < lb then D(b_i) ← {0} ;
```

We assume that $n \geq 1$. The first step is to look for the tightest possible bounds for x (Lines 1 and 2). The new upper bound ub is the minimum between $\max(\mathcal{D}(x))$ and the minimum value v_i s.t. b_i is assigned to 1. Similarly, the new lower bound lb is the maximum between $\min(\mathcal{D}(x))$ and the maximum value $v_i + 1$ s.t. b_i is assigned to 0. Clearly, the only situation to trigger a failure corresponds to the case when $ub < lb$ (Line 3). Otherwise, the domain of x, b_1, $\ldots$, b_n is updated in the natural way (Lines 4, 5, and 6).

In order to make Algorithm 2 incremental we first need to keep the atoms sorted: we assume that $v_i < v_{i+1}$. The filtering is then organized in two parts:

1. Simulating UP as if the atoms were eagerly generated with all domain clauses. That is, whenever an atom b_i becomes assigned to 1 (respectively 0), the atom b_{i+1} (respectively b_{i-1}) is assigned to 1 (respectively 0).
2. Performing a complete channeling between x and $b_1, \ldots, b_n$: We sketch the process related to the upper bound of x. A similar process in applied with the lower bound. There are two cases to distinguish:
   (a) Changing the upper bound of x w.r.t. newly assigned atoms: When an atom $b_i \leftrightarrow [\![x \leq v_i]\!]$ becomes assigned to 1, we check if v_i can be the new upper bound of x. Note that a failure can be triggered if $v_i < \min(x)$.
   (b) In the case of an upper bound propagation event, one has to assign some atoms to be consistent with the new upper bound u of x. Every atom b_i such that $v_i \geq u$ has to be assigned to 1. Let $i_{ub} = \arg\max_k(v_k \mid \mathcal{D}(b_k) = \{1\})$. Observe first that the part simulating UP (1) guarantees that all atoms b_i where $i > i_{ub}$ are already assigned to 1. Consider now the set $\varphi = \{b_{i_{ub}}, b_{i_{ub}-1}, b_{i_{ub}-2} \ldots, b_{last_{ub}}\}$ where $last_{ub} = \arg\min_k(v_k \mid v_k \geq u))$. It is now sufficient to assign every atom in φ to 1 in order to complete the filtering.

Finally, since down a branch of the search tree the upper bound can only decrease, we can compute the current value of i_{ub}, that is, $\arg\max_k(v_k \mid \mathcal{D}(b_k) = \{1\})$ by exploring the sequence of successive upper bounds from where we previously stopped. Therefore, each atom is visited at most once down a branch. This filtering can thus be performed in constant amortized time, that is, in $O(n)$ time down a branch, however, we must store i_{ub} and i_{lb} as "reversible" integers.

5 Learning Restricted to Task Ordering

Here we introduce a learning scheme as an alternative to lazy generation for disjunctive scheduling problems. Recall that we branch only on Boolean variables coming from the DISJUNCTIVE constraints. It follows that every bound literal p s.t. $level(p) > 0$ has a non-empty explanation. We can exploits precisely this property in order to avoid the generation of any bound atom.

The first step in this framework is to perform conflict analysis as usual by computing the 1-UIP nogood Ψ. Next, we make sure that the latest literal in Ψ is not a bound literal. Otherwise, we keep explaining the latest literal in Ψ until having such UIP. We know that this procedure terminates because the worst

Algorithm 3: Substitute($\Im, \Psi$)

```
   visited ← ∅ ;
   while |ℑ| > 0 do
1      p ← choose p ∈ ℑ ;
       visited ← visited ∪ {p} ;
2      Ψ ← Ψ ∪ {q | q ∈ explain(p) ∧ level(q) > 0} \ visited ;
3      ℑ ← {q | q ∈ Ψ ∧ q is a bound litteral};
   return Ψ ;
```

case would reach the last decision which corresponds to a UIP that is not a bound literal. Let Ψ^* be the resulting nogood.

Consider now $\Im$ to be the set of bound literals in Ψ^*. Instead of generating the corresponding atoms, we start a second phase of conflict analysis via the procedure Substitute($\Im, \Psi$) with ($\Im, \Psi^*$) as input. Algorithm 3 details this procedure. In each step of the main loop, a bound literal p from $\Im$ is chosen (Line 1) and replaced in Ψ with its explanation (Line 2). $\Im$ is updated later at each step in Line 3. The final nogood Ψ involves only Boolean variables. Note that the backjump level remains the same as in Ψ^* since for every p there must exists a literal in $explain(p)$ with the same level of p.

The advantage of this approach is that since no atom need to be generated during search, we do not need to maintain the consistency between tasks' domains and bounds atoms. In particular, it greatly reduces the impact that the size of the scheduling horizon has on the space and time complexities. Note, however, that there may be a slight overhead in the conflict analysis step, compared to the lazy generation mode, since there are more literals to explain. Moreover, since the language of literal is not as rich in this method, shorter proofs may be possible with the standard approach.

6 Experimental Results

We first compare the two approaches described in this paper, that is, the implementation of lazy generation using DomainFaithfulness ($lazy$) as well as the new learning scheme ($disj$) against the the CP model described in [11] on two well known benchmarks for the JSSP: Lawrence [13] and Taillard [22]. Then, we compare the lower bounds found by our approaches with the best known lower bounds in the literature.

All models are implemented within Mistral-2.0 and are tested on Intel Xeon E5-2640 processors. The source code, the detailed results, and the experimental protocol are available at **http://homepages.laas.fr/msiala/jssp/details.pdf**.

We denote CP($task$) the CP model using the $taskDom/tw$ heuristic. For the hybrid models, we use the notation H(θ, σ) where $\theta \in \{vsids, task\}$ is the variable ordering and $\sigma \in \{lazy, disj\}$ indicates the type of learning used.

6.1 Empirical Evaluation on the Job Shop Scheduling Problem

We ran every method 10 times using randomized geometric restarts [25]. Each dichotomy step is limited to 300s and 4×10^6 propagation calls. The total runtime for each configuration is limited to 3600s. The results are summarized in Table 1.

Table 1 is divided in two parts. The first part concerns instances that are mostly proven optimal by our experiments. We report for these instances the average percentage of calls where optimality was proven $\%O$, the average CPU time T, and the number of nodes explored by second (nds/s). The second part concerns the rest of the instances (i.e. hard instances). We report for each data set the speed of exploration (nds/s) along with the average *percentage relative deviation* (*PRD*) of each model. The PRD of a model m for an instance C is computed with the formula: $100 * \frac{C_m - C_{best}}{C_{best}}$, where C_m is the minimum makespan found by model m for this instance (among the several randomized runs); and C_{best} is the minimum makespan known in the literature [1, 2, 24]. Notice that the bigger the PRD, the less efficient a model is.

Table 1. Summary of the results

Instances	CP($task$)			H($vsids, disj$)			H($vsids, lazy$)			H($task, disj$)			H($task, lazy$)		
Mostly proven optimal															
	%O	T	nds/s	%O	T	nds/s	%O	T	nds/s	%O	T	nds/s	%O	T	nds/s
la-01-40	87	522	8750	**91.5**	437	6814	88	632	2332	90.50	434	5218	88.75	509	2694
tai-01-10	89	768	5875	**90**	517	4975	88	1060	1033	90	634	3572	84	1227	1013
Hard instances															
	PRD		nds/s	PRD		nds/s	PRD		nds/s	PRD		nds/s	PRD		nds/s
tai-11-20	1.8432		4908	**1.1564**		3583	1.3725		531	1.2741		2544	1.2824		489
tai-21-30	1.6131		3244	0.9150		2361	1.0841		438	0.9660		1694	**0.8745**		409
tai-31-40	5.4149		3501	4.0210		2623	**3.7350**		580	4.0536		1497	3.8844		510
tai-41-50	7.0439		2234	4.8362		1615	**4.6800**		436	4.9305		1003	5.0136		390
tai-51-60	3.0346		1688	3.2449		2726	3.7809		593	**1.1156**		1099	1.1675		575
tai-61-70	6.8598		1432	6.5890		2414	5.4264		578	3.9637		866	**3.6617**		533

Consider first small and medium sized instances, i.e., la-01-40, tai-01-10, and tai-11-20. Table 1 shows clearly that the learning scheme that we introduced (i.e. H($vsids/task, disj$)) dominates the other models on these instances. For example H($vsids, disj$) proves 91.5% of Lawrence instances to optimality whereas CP($task$) and H($vsids, lazy$) achieve a ratio of 87% and 88%, respectively. Moreover, VSIDS generally performs better than weighted degree on these instances, although this factor does not seem as important as the type of learning.

Consider now the rest of instances (tai-21-30 to tai-61-70). The impact of clause learning is more visible on these instances. For example, with tai-31-40, CP($task$) has a PRD of 5.4149 while the worst hybrid model has a PRD of 4.0536. The choice of the branching strategy seems more important on the largest instances. For instance, on tai-51-60, the PRD of H($task, disj$) is 1.1156 while H($vsids, disj$) has a PRD of 3.2449.

Regarding the exploration speed, CP($task$) is often the fastest model (w.r.t. nds/s). This is expected because of the overhead of propagating the learnt clauses in the hybrid models. It should be noted that lazy generation ($lazy$) slows the search down considerably compared to the mechanism we introduced ($disj$).

Overall, our hybrid models outperform the CP model in most cases. The novel conflict analysis shows promising results especially for small and medium sized instances. It should be noted that we did not find new upper bounds for hard instances. However, our experiments show that our best model deviates only of a few percents from the best known bounds in the literature. Note that the state-of-the-art CP method [24] uses a timeout of 30000s (similarly to [18]) along with a double threading phase, whereas we use a single thread and a fixed time limit of 3600s per instance.

6.2 Lower Bound Computation

We ran another set of experiments with a slightly modified dichotomy phase. Here we assume that the outcome of step i is satisfiable instead of unsatisfiable when the cutoff is reached. Each dichotomy step is limited to 1400s and the overall time limit is 3600s. We used the set of all 22 open Taillard instances before the results of [24] and we found 7 new lower bounds. However, most of our bounds were already improved by Vilím et al. bottom-up method in [24], here again with a 30000s cutoff per call to the CP engine. Moreover, their algorithm uses the best known bounds as an additional information before starting search. Our models, however, are completely self-contained in the sense where search is started from scratch (see Section 3).

We computed the average PRD w.r.t. the best known lower bound including those reported in [24]. Excluding the set tai40-50, the best combination, H($vsids, disj$) using Luby restarts [14], finds lower bounds that are 2.78% lower, in average, than the best known bound in the literature.

Finally, in order to find new lower bounds, we launched again the model H($vsids, disj$) with 2500s as a timeout for each dichotomy step and 7200s as a global cutoff. We found a new lower bound of 1583 for tai-29 (previously 1573) and one of 1528 for tai-30 (previously 1519).

7 Conclusion

We introduced two hybrid CP/SAT approaches for solving disjunctive scheduling problems. The first one avoids a known redundancy issue regarding lazy generation without computational overhead, whereas the second constitutes a 'hand-crafted' conflict analysis scheme for disjunctive scheduling. Our approach showed very promising results on Taillard instances. In particular the novel learning scheme is extremely efficient for proving unfeasibility. Indeed, we improved the best known lower bounds of two Taillard instances.

References

1. Best known lower/upper bounds for Taillard Job Shop instances. http://optimizizer.com/TA.php. Accessed: April 15, 2015.
2. Éric Taillard homepage. http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/jobshop.dir/best_lb_up.txt. Accessed: April 15, 2015.
3. J. Christopher Beck. Solution-guided multi-point constructive search for job shop scheduling. *Journal of Artificial Intelligence Research*, 29(1):49–77, 2007.
4. Christian Bessiere. Constraint propagation. In Peter van Beek Francesca Rossi and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 29 – 83. Elsevier, 2006.
5. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting Systematic Search by Weighting Constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'04, Valencia, Spain*, pages 146–150, 2004.
6. Jacques Carlier and Éric Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.
7. E. Danna and L. Perron. Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs. In *Proceedings of Principles and Practice of Constraint Programming - CP'03. LNCS No. 2833*, pages 817–821, 2003.
8. Thibaut Feydy, Andreas Schutt, and Peter J. Stuckey. Semantic Learning for Lazy Clause Generation. In *Proceedings of TRICS Workshop: Techniques foR Implementing Constraint programming Systems, TRICS'13, Uppsala, Sweden*, 2013.
9. Thibaut Feydy and Peter J. Stuckey. Lazy Clause Generation Reengineered. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, CP'09, Lisbon, Portugal*, pages 352–366, 2009.
10. Diarmuid Grimes and Emmanuel Hebrard. Job Shop Scheduling with Setup Times and Maximal Time-Lags: A Simple Constraint Programming Approach. In *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR'10, Bologna, Italy*, pages 147–161, 2010.
11. Diarmuid Grimes and Emmanuel Hebrard. Models and Strategies for Variants of the Job Shop Scheduling Problem. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, CP'11, Perugia, Italy*, pages 356–372, 2011.
12. Diarmuid Grimes, Emmanuel Hebrard, and Arnaud Malapert. Closing the Open Shop: Contradicting Conventional Wisdom. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, CP'09, Lisbon, Portugal*, pages 400–408, 2009.
13. Stephen R. Lawrence. Supplement to Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1984.
14. Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173 – 180, 1993.
15. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Annual Design Automation Conference, DAC'01, Las Vegas, Nevada, USA*, pages 530–535, 2001.

16. Wim Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Eindhoven University of Technology, 1994.
17. Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via Lazy Clause Generation. *Constraints*, 14(3):357–391, 2009.
18. Panos M. Pardalos and Oleg V. Shylo. An algorithm for the job shop scheduling problem based on global equilibrium search techniques. *Computational Management Science*, 3(4):331–348, 2006.
19. Norman M. Sadeh. *Lookahead techniques for micro-opportunistic job-shop scheduling*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1991.
20. Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Solving rcpsp/max by lazy clause generation. *Journal of Scheduling*, 16(3):273–289, 2013.
21. P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Proceedings of Principles and Practice of Constraint Programming - CP'98. LNCS No. 1520*, pages 417–431, 1998.
22. Éric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278 – 285, 1993. Project Management anf Scheduling.
23. Petr Vilím. Edge Finding Filtering Algorithm for Discrete Cumulative Resources in $\mathcal{O}(kn \log(n))$. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, CP'09, Lisbon, Portugal*, volume 5732, pages 802–816, 2009.
24. Petr Vilím, Philippe Laborie, and Paul Shaw. Failure-directed Search for Constraint-based Scheduling. In *Proceedings of the 12th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR'15, Barcelona, Spain*, page to appear, 2015.
25. Toby Walsh. Search in a Small World. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence, IJCAI'99, Stockholm, Sweden*, pages 1172–1177, 1999.
26. Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design, ICCAD'01, San Jose, California*, pages 279–285, 2001.