Improved Filtering of Scheduling Problems using Redundant Table Constraints

Hélène Verhaeghe[1][0000−0003−0233−4656], Roger Kameugne[2][0000−0003−1809−9822], Christophe Lecoutre[3][0000−0002−2205−6545], and Pierre Schaus[1][0000−0002−3153−8941]

[1] UCLouvain, ICTEAM, Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium
{helene.verhaeghe,pierre.schaus}@uclouvain.be
[2] Department of Mathematics and Computer Science, Faculty of Science, University of Maroua, P.O. Box 814, Maroua, Cameroon
rkameugne@gmail.com
[3] CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France
lecoutre@cril.fr

Abstract. Scheduling problems have been studied for a long time in Constraint Programming (CP). As the constraints used to model such scheduling problems are often NP-hard to filter, various relaxation mechanisms (typically, to prune time interval variables) have been imagined and still continue to be proposed. An example of such constraint is the `cumulative` constraint, for which fast and strong filtering algorithms have been conceived during the last two decades, which has permitted to greatly improve constraint-based scheduling solvers [23]. In this paper, we introduce a simple approach to further enhance the filtering process of scheduling problems, in general, by relying on redundant `table` constraints. The idea is to compile all solutions of an abstract version of the problem to be solved as a redundant `table` constraint. To cope with the possibly prohibitive size of the generated table, solutions are collected on task views at a coarser temporal granularity. Another important ingredient to control the size of the redundant table is to only consider a subset of the tasks, for instance, those that are critical according to a heuristic criterion such as the energy. Interestingly, this approach can be easily integrated in constraint solvers and is totally compatible with any existing filtering technique (propagators of the `cumulative` constraint, lazy-clause reasoning, ...). Our experiments on Resource-Constrained Project Scheduling Problem (RCPSP) show that this simple approach can effectively reduce time solving despite the preprocessing stage needed to build the redundant `table` constraint.

Keywords: Scheduling · Table constraint · Relaxation

1 Introduction

The success of CP on scheduling problems comes from the combination of global filtering algorithms to prune the search tree and specialized heuristics to exhibit

good-quality solutions quickly. During the exploration of the search space, filtering algorithms are repeatedly applied (possibly, thousand times) at every step (node of the search tree). The two required qualities for a filtering algorithm is its strength (amount of inconsistent values that it is able to remove) and its speed. Those objectives are in general conflicting, especially for the `cumulative` constraint known to be NP-complete in general [4].

The `cumulative` constraint [1] is a key ingredient for modeling and solving many real-world scheduling problems with Constraint Programming (CP). It allows modeling a capacity limit on some resources required by some tasks (or activities). More precisely, it ensures that at any time, the resource consumption of the overlapping tasks do not exceed the resource capacity. This constraint appears in application domains such as manufacturing [34], production scheduling [17, 37], space operations planning, ... Regarding the strength and the speed of the `cumulative`, a compromise is obtained by using polynomial filtering algorithms based on some relaxation mechanisms, among which timetabling [14], edge-finding [28, 41, 21], energetic reasoning [9], not-first/not-last [36, 35, 20] or disjunctive reasoning [15] are often used. Energetic reasoning interestingly subsumes most of the other rules ((extended) edge-finding, timetable edge-finding [41]) but not the not-first/not-last and disjunctive reasoning rules.

Another kind of constraints that is frequently used when modeling combinatorial problems in CP is the `table` constraint [25, 26, 30, 8, 42]. This constraint allows us to express any relation as a list (table) of tuples; any constraint can thus be theoretically encoded under this form. There is, however, a practical limitation: the size of the tables may grow exponentially with respect to the number of involved variables.

In this paper, our primary goal is to exploit `table` constraints in order to enhance the filtering process of scheduling problems (typically involving `cumulative` constraints). This process is partly inspired by tabling techniques [7], which replace targeted subsets of constraints with their corresponding precomputed sets of solutions. In our approach, for a given problem instance, a redundant table is generated and posted: the introduced table does not completely replace some specific parts of the model but instead represent the solutions of a relaxed (abstract) version of the instance. We show the interest of our approach with some practical results obtained on some representative instances of the Resource-Constrained Project Scheduling Problem (RCPSP), taken from the BL set [3] and PSPLib [22].

The paper is organized as follows. Some technical background is first introduced, before describing our general methodology. Then, we present some experimental results before concluding.

2 Technical Background and Related Work

A Constraint Satisfaction Problem (CSP) is specified by a triplet $P = (X, D, C)$ where X is a finite set of variables, D a set of finite domains, one for each variable, and C a finite set of constraints. A CSP solution is the assignment

of a value to each variable (from its domain) that satisfies each constraint. A Constraint Optimization Problem (COP) is defined by a tuple $P' = (X, D, C, z)$ where (X, D, C) is a CSP and $z : X \rightarrow \mathbb{R}$ is an objective function to be minimized or maximized. A COP solution is a solution of the underlying CSP; it is optimal if there is no other solution with a better value for the objective.

Scheduling problems are combinatorial constrained problems that are ubiquitous in industry. Many methods have been proposed to improve the solving process of such problems, among which, there are some strong filtering algorithms combining advanced data structures (Θ-tree [40], timetabling [41], Profile [18]) with the classic edge finding rule [28, 41, 21]. The decomposition of resource constraints was also proposed in [32] to enhance the search, the lazy clause explanation of various filtering rules was suggested in [33, 31], and an original approach based on the use of Multi-valued Decision Diagrams (MDDs) has been shown to reduce successfully the search tree of scheduling problems [5]. Importantly, propagation alone is usually not sufficient to solve problem instances, and so, it must be combined with some good heuristics to select variables and values during the search (when taking decisions). In [13], it was shown that Conflict Ordering Search (COS) is an adequate search scheme for scheduling problems.

Recently, auto-tabling has been proposed in [7] to reduce the size of search trees. The idea is to compile a sub-problem that involves only a few tightly constrained variables under the form of a `table` constraint. Although it suffices to solve a CSP with the specific constraints, the size of the sub-problem must be adequately controlled in order to limit this compilation pre-processing stage. An automatic discovery of such interesting sub-problem has been proposed in [2]. Compilation alternatives have been proposed in [27] and [39], where `regular` constraints and `mdd` constraints[4], are respectively generated instead of `table` constraints.

Precedence and resource constraints are generally the main components of scheduling problems. The complete tabling of a sub-problem involving the temporal original variables does not seem appropriate because the domain sizes are usually large; the size of the generated table is consequently prohibitive. It is then necessary to think about a form of temporal abstraction: introducing new abstract variables to reason with a coarser temporal granularity, before generating a table based on these new variables. Contrary to tabling, the computed table must be added as a redundant constraint since the table is based on a relaxation of the problem (and does not represent the solutions of a specific sub-problem). This is the methodology we use in this paper. It is related to abstraction techniques that have been introduced in the literature [19]. However, in our approach, we do not reason with separate abstraction levels, as e.g., in [11, 12].

In some respects, the method we propose combines several points of view, but from an abstraction perspective. This is related, although rather different, from [6] where the authors aim at combining model viewpoints for the same

[4] based on Multi-Valued Decision Diagrams (MDDs), and more generally, formulas in Deterministic Decomposable Negation Normal Form (d-DNNFs)

problem. In their paper, they combine several (partly redundant) models, which are linked by *channeling* constraints; this idea is further developed in [24] where a redundant model is introduced using model induction, model channeling, and model intersection.

3 Methodology

To prevent combinatorial explosion, in our context of generating a (redundant) `table` constraint associated with a scheduling problem, we first perform some abstraction of the involved temporal variables by means of a parameter κ, which is a positive integer. The time horizon $0..H$, where H is a pre-computed time upper bound, is then partitioned into a sequence of κ time intervals of similar size. More precisely, if $H+1$ is a multiple of κ, then with $\sigma = (H+1)/\kappa$ the successive time intervals are: $TI_\kappa^H = \langle [0..\sigma-1], [\sigma..2*\sigma-1], \dots \rangle$. In case $H+1$ is not a multiple of κ, some time intervals (the first ones) are 1 time unit greater than the others. For example, if $\kappa = 4$ and $H = 13$, then $TI_\kappa^H = \langle [0..3], [4..7], [8..10], [11..13] \rangle$. Reasoning with these intervals (hence, at a coarser temporal granularity) will allow us to build redundant abstract tables of reasonable size.

Once the temporal abstraction is made, we can introduce some abstract temporal variables. For simplicity, assume that $\langle s_1, s_2, \dots, s_n \rangle$ are the temporal variables involved in a scheduling problem: each variable s_i represents the starting time of a task i, and is such that $dom(s_i) = [0..H]$. We then introduce $\langle s_1^a, s_2^a, \dots, s_n^a \rangle$ with $dom(s_i^a) = TI_\kappa^H$ after having chosen a value for κ. Of course, we need to link original (concrete) variables with the new abstract ones. This is made possible by introducing binary channeling constraints: for each i, we have a constraint involving s_i and s_i^a that is satisfied iff the value assigned to s_i belongs to the time interval assigned to s_i^a. Technically, such channeling constraints can be defined extensionally, i.e., using a `table` constraint.

Finally, a `table` constraint is defined on $\langle s_1^a, s_2^a, \dots, s_r^a \rangle$; the table is generated by collecting all solutions that can be found for the abstract temporal variables within the context of the original problem. More specifically, an easy way to create the table is to consider the initial set of constraints, to add the down-scaled views of the tasks (i.e. the abstract temporal variables) with the corresponding channeling constraints, and to branch only on these abstract down-scaled variables. This `table` constraint can then be posted to the constraint network, and used as a redundant constraint.

Note that an upper bound for the horizon can be initially computed by various means, for example by considering the first feasible solution obtained by diving left in the search tree. Concerning the granularity of the abstraction, where the activity views are given at a coarser level, an example for a scheduling problem, where the time unit is initially the hour, corresponds to a down-scaled view considering now the day as the schedule's time unit, significantly reducing the possible number of values to branch on. One can use various time partition schemes to create the down-scaled views of the original activities: this can be regular (as defined earlier) but this can also be problem specific (some time

intervals being quite larger than others, due to the structure of the problem such as lot of short tasks preceding longer tasks, and/or some temporal variables being not abstracted). For generality's sake, in our experiments, we have only considered a regular time partition systematically applied to all activities.

The final search tree branches only on the initial variables of the problem.

4 Experimentation

We have conducted some experiments on the Resource-Constrained Project Scheduling Problem (RCPSP). In a RCPSP instance, a finite set of n tasks (or activities) T is executed with the help of a finite set of m resources R. Each resource $r \in R$ has a finite capacity C_r, and each task $i \in T$ has a starting time s_i, an ending time e_i, and is executed without interruption during p_i units of time (i.e. we have $s_i + p_i = e_i$) while using c_{ir} units of each resource r. The limited capacity of a resource r is enforced with a `cumulative` constraint: $\texttt{Cumulative}(\{(s_i, p_i, e_i, c_{ir}) \mid i \in T\}, C_r)$. An acyclic network of precedence restrictions between tasks is also given. Each precedence is given by a pair (i, j) meaning that the task i must precede the task j; the task j can only start after the end of the task i (i.e., we have $e_i \leq s_j$). Finally, all tasks should end before a given specified time called the horizon h. Two special tasks, with id 0 and $n+1$, with a duration of 0, are used as starting and ending tasks of the project. The task 0 precedes all the tasks while the task $n+1$ succeeds all of them.

Figure 1 shows an example of the application of our approach. Looking at the generated redundant `table` constraint, Figure 1d, one can see that the domain of the variable s_3^a is directly reduced (and so, by channeling, the domain of the original variable s_3 is also reduced).

For analyzing the results of our experiments, we have used performance profiles based on performance ratios introduced by [10]. For a given set of problems (instances) P and a set of solvers S, if $m_{p,s}$ is the measure (time,...) computed for the solver s on problem p, then the performance ratio is defined by $r_{p,s} = \frac{m_{p,s}}{\min_{i \in S} m_{p,i}}$. The performance profile of a set of solvers $s \in S$ on a set of problems P is a cumulative distribution of the speedup performance compared to the other solvers of S on P: $\rho_s(\tau) = \frac{1}{|P|} \times |\{p \in P \mid r_{p,s} \leq \tau\}|$.

4.1 Settings.

This section indicates which parameter values were used in our experiments so as to make our results reproducible. Our code is available publicly[5] and is based on the publicly available Oscar solver [29].

Benchmarks. In our experiments, we have used two well-known series of instances: J30 and J60 (480 instances each, from the PSPLib library [22], composed of 30 and 60 tasks). The series of instances BL20 and BL25 were also

[5] https://bitbucket.org/helene_verhaeghe/rcpspwithtables

	t_0	t_1	t_2	t_3	t_4	t_5	t_6
durations	0	2	2	3	2	1	0
resource 1 uses	0	1	1	2	1	1	0
resource 2 uses	0	1	3	1	1	1	0

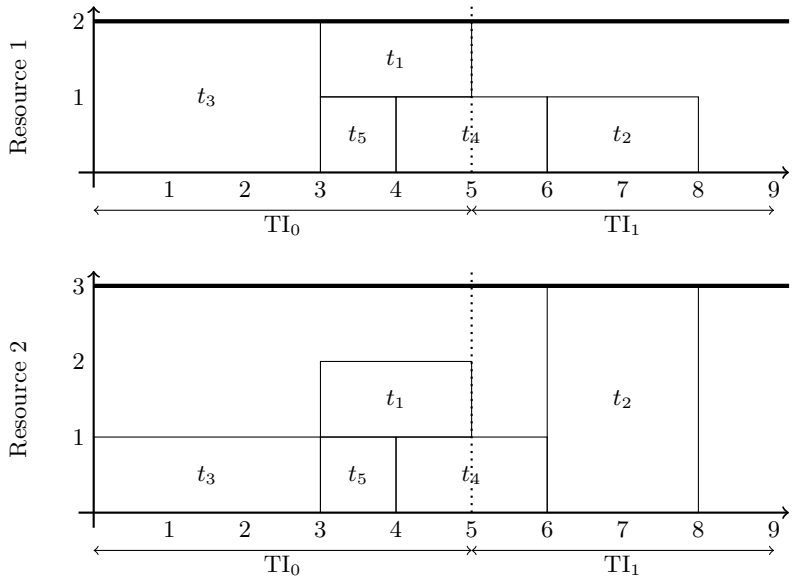
(a) Tasks and Resources

0	1,2,3,4,5	3	4,6
1	2,6	4	6
2	6	5	6

(b) Precedence Relation

	s_0^a	s_1^a	s_2^a	s_3^a	s_4^a	s_5^a	s_6^a
τ_1	0	0	1	0	0	0	1
τ_2	0	1	1	0	0	0	1
τ_3	0	0	1	0	0	1	1
τ_4	0	1	1	0	0	1	1
τ_5	0	0	0	0	1	0	1
τ_6	0	0	0	0	1	1	1
τ_7	0	0	1	0	1	0	1
τ_8	0	0	1	0	1	1	1
τ_9	0	1	1	0	1	0	1

(c) Redundant `table` constraint



(d) Illustration of an optimal solution (corresponding to relaxed solution τ_1). Starts of activities t_1, t_3, t_4 and t_5 belong to TI_0 while start of activity t_2 belongs to TI_1

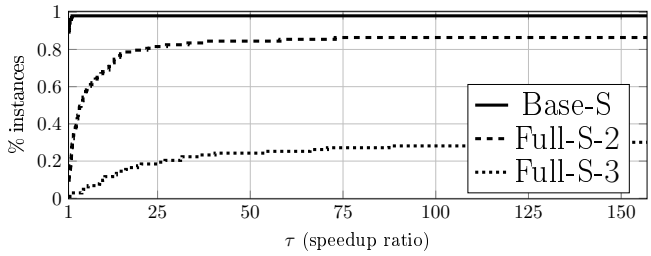
Fig. 1: Illustration of the methodology with a simple instance composed of 5 tasks and 2 resources (whose respective capacities are 2 and 3); t_0 and t_6 are the starting and ending special tasks. The horizon is 9, and the value κ chosen for the temporal abstraction is 2 so that we get $TI = \langle [0..4], [5..9] \rangle$. The new (abstract) temporal variables s_i^a have $\{0, 1\}$ as domain allowing us to index the two intervals in TI. The redundant `table` constraint that can be generated with our approach is given by Figure 1d; note that τ_1 and τ_3 leads to optimal solution with a horizon of 8, whereas the other tuples lead to solutions with a horizon of 9.

considered (20 instances each, from [3], with 20 and 25 tasks). As they take less than 2 seconds to be solved by the baseline model (i.e., they were too easy to solve), an improvement on these is not significant.

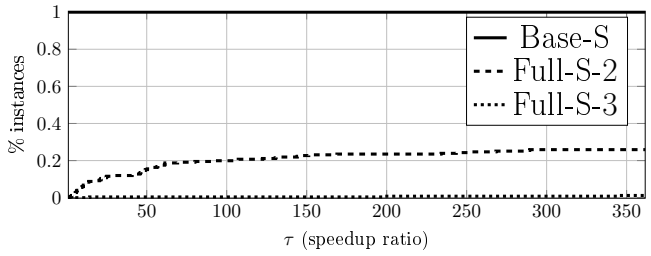
Baseline Solving. As a baseline, we have used two ways of solving the RCPSP instances on the basis of the initial model (i.e., without abstraction): Base-M, which uses the medium filtering level available in Oscar, and, Base-S, which uses the strong(est) one. The medium filtering algorithm combines TimeTable [14], TimeTable Disjunctive Reasoning [16] and TimeTable Edge Finder [31] (without explanation). The strong one adds Energetic Reasoning [9]. For exploring the search space, we have used conflict ordering search [13]. Note that all easy instances have been discarded, that is, the instances requiring less than 2 seconds to complete with the baseline model.

First Solution. The initial horizon used to compute the redundant `table` constraint is set with the left-most first feasible solution that can be found with the Base-M model.

4.2 Case 1: Considering All Tasks



(a) J30, strong filtering



(b) J60, strong filtering

Fig. 2: Performance profiles for Case 1.

In our first experiment, a redundant table is generated while considering all tasks involved in the instance to be solved; this approach is denoted by Full-ϕ-κ

where $\phi \in \{M, S\}$ is the chosen level of filtering (medium or strong), and κ is the temporal granularity (tested with values 2 and 3). Note that for the J60 instances, the arity of the generated tables is thus equal to 60. The methodology we apply is as follows: Step 1: the initial Base-M model is used to find a first solution; Step 2: the redundant table is generated, using Base-S (or -M) model with, additionally, the down-scaled new (abstract) variables and channeling constraints (the search process focusing on those additional variables); Step 3: the redundant table is added to Base-S (or -M) model to find an optimal solution. The time is measured as the methodology total time (i.e., the time to do the three steps).

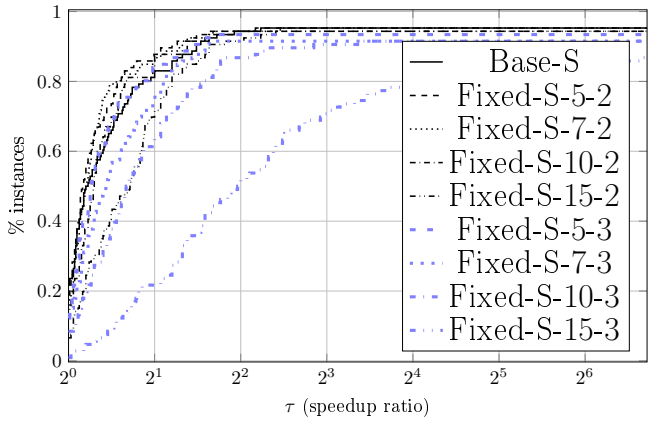
The results that we have obtained with the strong level of filtering for the J30 and J60 instances are given by Figure 2, which clearly shows that computing *full* redundant tables (i.e., tables involving all tasks) significantly increases the total time needed to prove optimality. The impact is even higher on the second data set that involves more tasks (J60), which then suggests, unsurprisingly, that controlling the arity of the tables is important. The results for the medium level of filtering lead to the same conclusion. Hence, we propose to refine the approach by only keeping the most critical tasks when generating redundant tables. Critical tasks are the ones that are not easy to set inside the schedule, because, for instance, they consume more resources and/or are involved in many precedence restrictions. Less critical tasks are more easily set, and, as a consequence, contribute to increasing the size (number of tuples) of the tables without any substantial filtering gain. Therefore, in our next experiment, such non-critical tasks will be discarded when generating redundant tables. Note that a similar technique is used in [38], where the authors select the most critical rectangles (to be packed), generate solutions for them, and finally integrate the remaining rectangles to form complete solutions.

4.3 Case 2: Considering a Fixed Number of Tasks

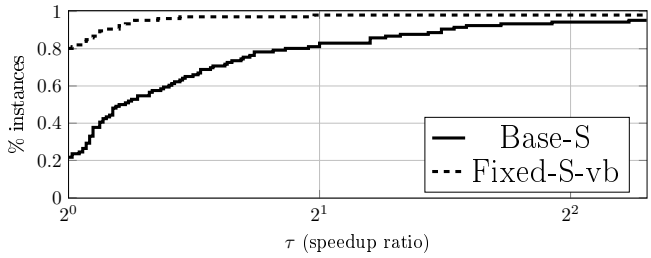
In our second experiment, a redundant table is generated while considering only a fixed number K of tasks; this approach is denoted by Fixed-ϕ-K-κ where κ has again been tested with values 2 and 3. The chosen tasks are the K tasks with the highest total energy values ($p_i \cdot \sum_{r \in R} c_{ir}$). Tested values for K are 5, 7, 10, and 15 for the J30 instances and 5, 10, 15, and 20 for the J60 instances.

Starting with Base-S and the J30 instances, one can observe in Figure 3a that the best combinations of parameters are for K-κ equal to 5-2, 7-2, 10-2 and 5-3, which are rather competitive with Base-S. In Figure 3b, Base-S is compared with the virtual best method (vb) built from all tested combinations of parameters K-κ; vb simulates an approach where every instance would be run with its best-suited parameters. Figure 3c shows a similar comparison with Base-M. This shows that our approach has the capability of solving some instances, unsolved when baseline solving is used, within the timeout.

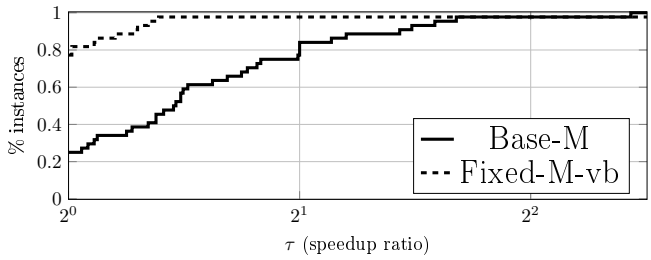
On the 60 instances (Fig. 4a), the runtime is improved by one of our parameter combinations on 50% of the instances as shown in Fig. 4b (as both curves start at around 0.5). A more thorough examination of the results shows that our



(a) J30, all parameters, strong filtering



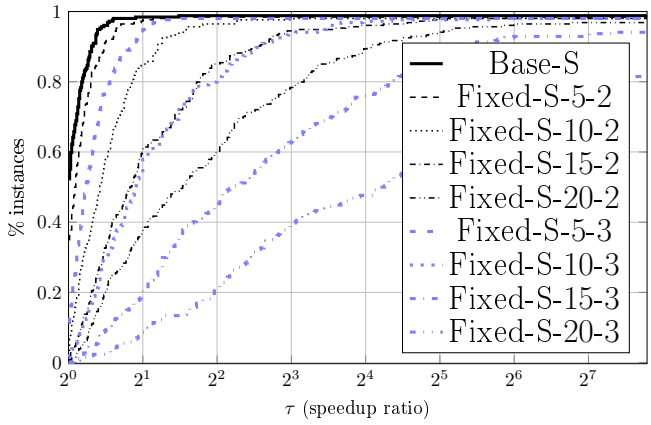
(b) J30, virtual best, strong filtering



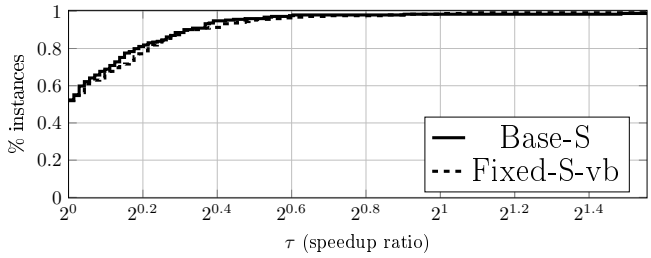
(c) J30, virtual best, medium filtering

Fig. 3: Performance Profiles for Case 2, J30 benchmark.

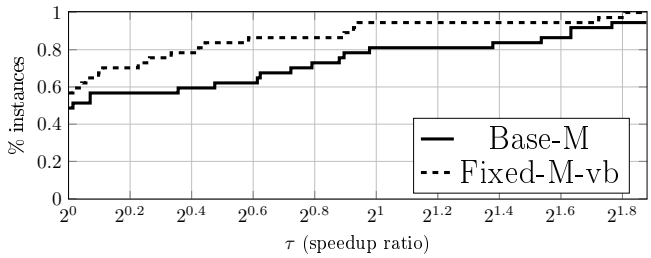
methodology is better on 70% of the instances, with the strongest propagator, when the optimal solution is not the first solution found. Fig. 4c shows Base-M combined with vb: we can see an improvement for almost 60% of the instances. This improvement is even better when only considering the instances for which the optimal solution is not the first one found by the solver.



(a) J60, all parameters, strong filtering



(b) J60, virtual best, strong filtering



(c) J60, virtual best, medium filtering

Fig. 4: Performance Profiles for Case 2, J60 benchmark.

4.4 Case 3: Iterative Table Generation

For our third experiment, a time budget as well as a maximum arity are given as limits during table creation. The table is built in an iterative manner, gradually augmenting the set of tasks. In addition, at step (iteration) i, the table from the previous step $i-1$ is already added to the model. The table creation process stops when the time is consumed or the maximum arity is reached. The last completed table is then used as a redundant constraint.

We tried with the following combinations of parameters on the J60 set: a maximum of 10 seconds allowed for creating the table, a maximum of 5, 10, 15 or 20 tasks per table, and κ set to 2. The iterative approach improved the solving time for 7% of the instances, compared to Base-S, and 65% of the instances, compared to Base-M. The parameters achieving this improvement were 10 seconds with 5 activities maximum. This corresponds to more difficult instances where even a small relaxed table is complicated to compute in a reasonable amount of time.

4.5 Global Comparison and Complementary Results

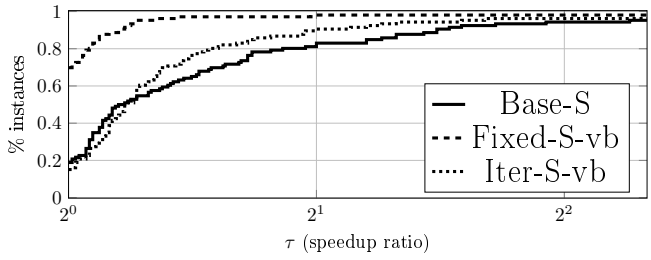
A comparison between Base-S, Fixed-S-vb (Case 2) and Iter-S-vb (Case 3) is displayed in Figure 5c for the J60 series. As one can see, the new approach improves resolution time for around 50% of the instances; most of them were improved when considering a fixed number of tasks. Compared to Base-M, still for the J60 series (Fig. 5b), the improvement is observed on 70% of the instances. Finally, for the J30 series (Fig. 5a), compared to Base-S, 80% of the instances benefits from the technique.

Figure 5d shows the speedup of vb (virtual best) when measuring only the step 3 of our approach (i.e., the time to prove optimality without the time to compute the redundant table). As it can be seen, the redundant constraint almost always contributes improving the solving time. This convinces us that investigating new (automatic) mechanisms for identifying the best tradeoff could be worthwhile. This precomputation step (of generating a redundant table) can also be perceived as being mainly offline in some contexts where the problem instances slightly change, and a form of reoptimization is simply needed. The precomputed table on the stable part of the model could be reused for a series of similar instances in such cases.

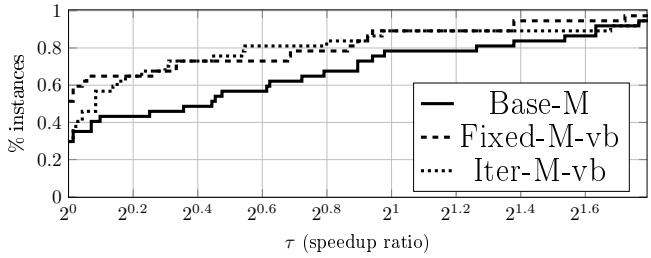
5 Conclusion

We have proposed a methodology for improving the filtering capability of scheduling problems by compiling all solutions of an abstract view of (a subset of) tasks under the form of redundant `table` constraints. Several mechanisms to control the time required for precomputing such tables have been presented: reasoning on task views at an appropriate temporal granularity, selecting only the most impacting tasks, and incrementally creating tables with a time budget. Interestingly, the proposed methodology is rather generic and ready to use in any constraint solver. We have applied this methodology to the RCPSP, showing promising results.

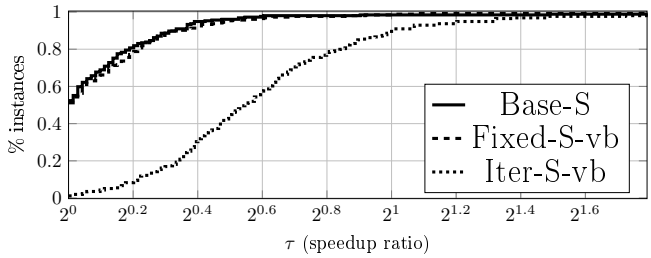
This work must be perceived as a proof of concept, showing that global redundancy controlled by abstraction can pay off. Our approach is a kind of hybridization between well-known techniques of redundant modeling and abstraction reformulation. As a future work, we plan to investigate how to automatically tune parameters so as to find the best trade-off between precomputation time



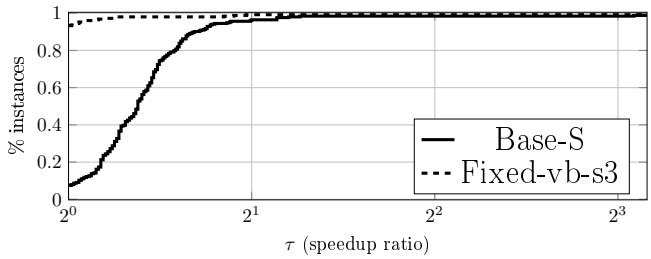
(a) J30, global comparison, strong filtering



(b) J60, global comparison, medium filtering



(c) J60, global comparison, strong filtering



(d) J60, performance profile of the last step only

Fig. 5: Global comparison and evaluation of the last step

and filtering gain (machine learning techniques could be useful), and, of course to test our approach on other problems.

References

1. Aggoun, A., Beldiceanu, N.: Extending CHIP in Order to Solve Complex Scheduling Problems. Journal of Mathematical and Computer Modelling **17**(7), 57–73 (1993)
2. Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P., Salamon, A.Z.: Automatic discovery and exploitation of promising subproblems for tabulation. In: International Conference on Principles and Practice of Constraint Programming. pp. 3–12. Springer (2018)
3. Baptiste, P., Pape, C., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. International Series in Operations Research & Management Science, Springer US (2012), https://books.google.cm/books?id=qUzhBwAAQBAJ
4. Baptiste, P., Le Pape, C.: Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems. Constraints **5**(1), 119–139 (2000)
5. Bergman, D., Ciré, A.A., van Hoeve, W.J.: MDD Propagation for Sequence Constraints. J. Artif. Intell. Res. **50**, 697–722 (2014)
6. Cheng, B.M.W., Choi, K.M.F., Lee, J.H., Wu, J.C.K.: Increasing Constraint Propagation by Redundant Modeling: an Experience Report. Constraints An Int. J. **4**(2), 167–192 (1999)
7. Dekker, J.J., Björdal, G., Carlsson, M., Flener, P., Monette, J.: Auto-Tabling for Subproblem Presolving in MiniZinc. Constraints An Int. J. **22**(4), 512–529 (2017). https://doi.org/10.1007/s10601-017-9270-5, https://doi.org/10.1007/s10601-017-9270-5
8. Demeulenaere, J., Hartert, R., Lecoutre, C., Perez, G., Perron, L., Régin, J.C., Schaus, P.: Compact-Table: Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets. CP (2016)
9. Derrien, A., Petit, T.: A New Characterization of Relevant Intervals for Energetic Reasoning. CP pp. 289–297 (2014)
10. Dolan, E.D., Moré, J.J.: Benchmarking Optimization Software with Performance Profiles. Math. Program. **91**(2), 201–213 (2002). https://doi.org/10.1007/s101070100263, https://doi.org/10.1007/s101070100263
11. Freuder, E., Sabin, D.: Interchangeability supports abstraction and reformulation for multi-dimensional constraint satisfaction. In: Proceedings of AAAI'97. pp. 191–196 (1997)
12. Frisch, A., Hnich, B., Miguel, I., Smith, B., Walsh, T.: Towards csp model reformulation at multiple levels of abstraction. In: Proceedings of CP'2002: Workshop on reformulating CSP (2002)
13. Gay, S., Hartert, R., Lecoutre, C., Schaus, P.: Conflict Ordering Search for Scheduling Problems. In: International conference on principles and practice of constraint programming. pp. 140–148. Springer (2015)
14. Gay, S., Hartert, R., Schaus, P.: Simple and Scalable Time-Table Filtering for the Cumulative Constraint. In: Pesant, G. (ed.) Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9255, pp. 149–157. Springer (2015). https://doi.org/10.1007/978-3-319-23219-5_11, https://doi.org/10.1007/978-3-319-23219-5_11
15. Gay, S., Hartert, R., Schaus, P.: Time-table disjunctive reasoning for the cumulative constraint. In: International Conference on AI and OR Techniques in Constriant

Programming for Combinatorial Optimization Problems. pp. 157–172. Springer (2015)
16. Gay, S., Hartert, R., Schaus, P.: Time-Table Disjunctive Reasoning for the Cumulative Constraint. In: Michel, L. (ed.) Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9075, pp. 157–172. Springer (2015). https://doi.org/10.1007/978-3-319-18008-3_11, https://doi.org/10.1007/978-3-319-18008-3_11
17. Gay, S., Schaus, P., De Smedt, V.: Continuous casting scheduling with constraint programming. In: International conference on principles and practice of constraint programming. pp. 831–845. Springer (2014)
18. Gingras, V., Quimper, C.: Generalizing the Edge-Finder Rule for the Cumulative Constraint. In: IJCAI. pp. 3103–3109. IJCAI/AAAI Press (2016)
19. Giunchiglia, F., Walsh, T.: A theory of abstraction. Artificial Intelligence **56**(2-3), 323–390 (1992)
20. Kameugne, R., Fetgo, B.S., Gingras, V., Ouellet, Y., Quimper, C.G.: Horizontally Elastic Not-First/Not-Last Filtering Algorithm for Cumulative Resource Constraint. CPAIOR pp. 316–332 (2018)
21. Kameugne, R., Fotso, L.P., Scott, J.D., Ngo-Kateu, Y.: A Quadratic Edge-Finding Filtering Algorithm for Cumulative Resource Constraints. Constraints An Int. J. **19**(3), 243–269 (2014). https://doi.org/10.1007/s10601-013-9157-z, https://doi.org/10.1007/s10601-013-9157-z
22. Kolisch, R., Sprecher, A.: PSPLIB - A Project Scheduling Problem Library: OR Software - ORSEP Operations Research Software Exchange Program. European Journal of Operational Research **96**(1), 205–216 (1997). https://doi.org/10.1016/S0377-2217(96)00170-1
23. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP Optimizer for Scheduling. Constraints **23**(2), 210–250 (2018)
24. Law, Y.C., Lee, J.H.: Model Induction: A New Source of CSP Model Redundancy. In: AAAI/IAAI. pp. 54–61. AAAI Press / The MIT Press (2002)
25. Lecoutre, C.: STR2: Optimized Simple Tabular Reduction for Table Constraints. Constraints **16**(4), 341–371 (2011)
26. Lecoutre, C., Likitvivatanavong, C., Yap, H.C.R.: STR3: A Path-Optimal Filtering Algorithm for Table Constraints. Artif. Intell. pp. 1–27 (2015)
27. Löffler, S., Liu, K., Hofstedt, P.: The Regularization of Small Sub-Constraint Satisfaction Problems. In: DECLARE. Lecture Notes in Computer Science, vol. 12057, pp. 106–115. Springer (2019)
28. Mercier, L., Van Hentenryck, P.: Edge Finding for Cumulative Scheduling. INFORMS Journal on Computing **20**(1), 143–153 (2008)
29. OscaR Team: OscaR: Scala in OR (2012), available from `https://bitbucket.org/oscarlib/oscar`
30. Perez, G., Régin, J.C.: Improving GAC-4 for Table and MDD Constraints. In: International Conference on Principles and Practice of Constraint Programming. pp. 606–621. Springer (2014)
31. 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, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7874,

pp. 234–250. Springer (2013). https://doi.org/10.1007/978-3-642-38171-3_16, https://doi.org/10.1007/978-3-642-38171-3_16
32. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.: Why Cumulative Decomposition is not as Bad as it Sounds. In: CP. Lecture Notes in Computer Science, vol. 5732, pp. 746–761. Springer (2009)
33. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the Cumulative Propagator. Constraints An Int. J. **16**(3), 250–282 (2011)
34. Schutt, A., Stuckey, P.J., Verden, A.R.: Optimal carpet cutting. In: International Conference on Principles and Practice of Constraint Programming. pp. 69–84. Springer (2011)
35. Schutt, A., Wolf, A.: A New $o(n^2 logn)$ Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints. In: International Conference on Principles and Practice of Constraint Programming. pp. 445–459. Springer (2010)
36. Schutt, A., Wolf, A., Schrader, G.: Not-First and Not-Last Detection for Cumulative Scheduling in $o(n^3 logn)$. In: International Conference on Applications of Declarative Programming and Knowledge Management. pp. 66–80. Springer (2005)
37. Simonin, G., Artigues, C., Hebrard, E., Lopez, P.: Scheduling scientific experiments for comet exploration. Constraints **20**(1), 77–99 (2015)
38. Simonis, H., O'Sullivan, B.: Search strategies for rectangle packing. In: Stuckey, P.J. (ed.) Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5202, pp. 52–66. Springer (2008). https://doi.org/10.1007/978-3-540-85958-1_4, https://doi.org/10.1007/978-3-540-85958-1_4
39. de Uña, D., Gange, G., Schachte, P., Stuckey, P.J.: Compiling CP Subproblems to MDDs and d-DNNFs. Constraints An Int. J. **24**(1), 56–93 (2019)
40. Vilím, P.: Edge Finding Filtering Algorithm for Discrete Cumulative Resources in $O(kn$ log $n)${\mathcal O}(kn {\rm log} n). In: CP. Lecture Notes in Computer Science, vol. 5732, pp. 802–816. Springer (2009)
41. Vilím, P.: Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources. CPAIOR pp. 230–245 (2011)
42. Yap, H.C.R., Xia, W., Wang, R.: Generalized Arc Consistency Algorithms for Table Constraints - A Summary of Algorithmic Ideas. AAAI pp. 13590–13597 (2020)