Augmenting the Cumulative Overload Check with Integral Resource Usage Reasoning

Samuel Cloutier
Université Laval, Québec, Canada

Claude-Guy Quimper
Université Laval, Québec, Canada

1 Extended Abstract

The Resource-Constrained Project Scheduling Problem (RCPSP) consists of scheduling tasks under both resource and precedence constraints. Constraint programming, with the Cumulative constraint, shows notable success in solving this problem [11]. Enhancing the filtering of the Cumulative constraint may lead to better solutions in the industrial context.

This research presents a checker for the Cumulative constraint, based on the overload check, that is augmented by reasoning on the consumptions as integral quantities. We show how general explanations for this checker can be generated in a lazy clause generation solver. This extended abstract summarizes our accepted paper for the CP 2026 main conference.

Let $\mathcal{I}$ be a set of n tasks, and let p_i and c_i for $i \in \mathcal{I}$ be the task's processing time and resource usage. The energy of a task is the product between its processing time and resource usage, i.e., $e_i = p_i \times c_i$. Let S_i be its starting time variable. Each task i has an earliest (latest) start time noted est_i (lst_i). These values are associated with the current bounds of the domain of S_i. The earliest (latest) completion times of i are defined by $\text{ect}_i = \text{est}_i + p_i$ ($\text{lct}_i = \text{lst}_i + p_i$). The Cumulative$([S_1, \ldots, S_n], [p_1, \ldots, p_n], [c_1, \ldots, c_n], C)$ constraint asserts that the total consumption from tasks in execution never, at any time, overloads a resource of capacity C [1]. Formally, for every time t, $\sum_{i \in \mathcal{I} | t \in [S_i, S_i + p_i)} c_i \leq C$.

Few filtering rules for the Cumulative directly reason about the integrity of the consumptions. Rules seldom evaluate the actual available resource, at a specific time point, given the resolution state. They rather consider the whole capacity as available or only remove what is used by compulsory parts. Recent work augments the energetic reasoning rule through the resolution of a tri-partition problem [2]. By computing the maximum amount of energy that tasks can use outside a given time interval, they can better determine the energy required in that interval, which strengthens the filtering.

Our main contribution is an enhancement to the time-table horizontally elastic overload check [5] by defining a stronger evaluation of the resource availability.

▶ **Example 1.** Consider $C = 3$ and $\mathcal{I} = \{1, 2, 3, 4\}$ where, $\forall i \in \mathcal{I}$, $\text{est}_i = 0$, $\text{lct}_i = 7$, $p_i = 2$, and $c_i = 2$. When we apply the standard overload check [12], the available energy is $3 \times (7 - 0) = 21$. Since $\sum_{i \in \mathcal{I}} e_i = 16$, no conflict is detected. It is the same if we instead apply the stronger time-table horizontally elastic overload check [5] or augmented energetic reasoning [2]. However, a parity test shows that the third unit of resource is never used, that the available energy can be reduced to $2 \times (7 - 0) = 14$, and that a conflict must occur.

Example 1 illustrates a case in which the resource available to a set of tasks can be better approximated based on divisibility. However, there are cases that such a basic reasoning misses. It is why we consider the quantity of resource still available to a subset of tasks $\Theta \subseteq \mathcal{I}$ at time t, not counting what is already used by compulsory parts ($f(\mathcal{I}, t) = \sum_{i \in \mathcal{I} | t \in [\text{lst}_i, \text{ect}_i)} c_i$),

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:3
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to be the solution of a knapsack problem. Let $h^K_{\max}(\Theta, t)$ be that quantity:

$$h^K_{\max}(\Theta, t) := \max\left\{ \sum_{i\in\Theta | t\in[\text{est}_i,\text{lct}_i)\setminus[\text{lst}_i,\text{ect}_i)} c_i x_i \;\middle|\; \sum_{i\in\Theta | t\in[\text{est}_i,\text{lct}_i)\setminus[\text{lst}_i,\text{ect}_i)} c_i x_i \leq C - f(\mathcal{I}, t), x_i \in \{0,1\} \right\}$$

Using this new definition of the resource availability, we adapt the time-table horizontally elastic overload check to take into account the integrity of the resource consumption. This new *knapsack augmented overload check* (KAOC) dominates the previous checker.

Using bitsets to compute the resource availability queries in amortized $\mathcal{O}\left(\frac{C}{w}\right)$ time (where w is the size of a memory word), we implement a propagator for this rule which runs in $\mathcal{O}\left(\frac{C}{w}n^2\right)$. On usual instances, C is small and this complexity collapses back to $\mathcal{O}(n^2)$, the complexity of the dominated checker. In the context of lazy clause generation solvers, we also present greedy algorithms to generate general explanations that run in $\mathcal{O}\left(\frac{C}{w}n^3\right)$.

We evaluate the performance of our new checker by solving various instances of the RCPSP. We model the problem in MiniZinc [8] and integrated our propagator to Chuffed version 0.13.0 [4]. We compare against 4 other propagators of the Cumulative constraint from the Chuffed solver, as well as the propagator of the time-table horizontally elastic overload check in Choco version 4.10.10 [9]. We perform our experiments on the Pack [3], Pack_d [7] and PSPLIB [6] benchmarks. On these benchmarks, two machine words (of 64-bits) are always sufficient to encode the bitsets (the greatest C is 122). All the options are tested twice on each instance of the benchmarks: once on the base instances and a second time on the instances transformed using pre-solving rules that can apply knapsack-based reasoning to transform the capacities and consumptions to enhance energy-based filtering [10]. We thus compare 12 approaches. In order to show the difference in robustness between our new propagator and the pre-solving method, we introduce the new Pack+1, Pack_d+1, and PSPLIB+1 benchmarks. Each is generated, respectively, from Pack, Pack_d, and PSPLIB by adding to each instance one task of processing time 1 with a resource consumption of 1. Our code, instances, and raw results are available[1].

The results for the PSPLIB instances show little interest due to their disjunctive nature that stronger energy-based rules are not designed for. For most reference methods, pre-solving the instances permits the resolution of about 15 more instances out of 110 Pack/Pack_d instances. KAOC is mostly unaffected by pre-solving, is not dominated by the other options, and solves instance pack040 in 2 minutes (to our knowledge for the first time in the literature), while the other approaches fail to prove optimality before the 30 minute time-out. However, on Pack+1/Pack_d+1, we see that the addition of the task in these benchmarks neutralized the positive effect of pre-solving. All methods are now mostly unaffected by it. However, KAOC's performance remains unchanged, making it the dominant approach for solving these instances. This highlights the fragility of pre-solving and the robustness of our new approach.

In conclusion, we added reasoning based on the integrality of the resource consumptions to the horizontally elastic overload check and defined our new knapsack augmented overload check for the Cumulative constraint. We showed how this new rule can be applied in $\mathcal{O}\left(\frac{C}{w}n^2\right)$, only adding a small pseudo-polynomial factor due to the resolution of knapsack sub-problems. We also presented how to explain our new propagator in lazy clause generation solvers. Experiments showed that the new propagator helps solve more instances for the Pack and Pack_d while being more robust than some pre-solving methods. Applying our enhancement to the edge-finding rule represents a potential avenue for future work.

[1] Available at: https://github.com/Samclou/chuffed/releases/tag/KAOC-cp2026

References

1 Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17:57–73, 1993.

2 Jacques Carlier, Antoine Jouglet, Kristina Kumbria, and Abderrahim Sahli. More powerful energetic reasoning for the cumulative scheduling problem. *Discrete Applied Mathematics*, 378:187–209, 2026.

3 Jacques Carlier and Emmanuel Néron. On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, 149:314–324, 2003.

4 Geoffrey Chu. *Improving Combinatorial Optimization*. PhD thesis, The University of Melbourne, 2011.

5 Roger Kameugne, Sévérine Fetgo Betmbe, Thierry Noulamo, and Clémentin Tayou Djamegni. Improved timetable edge finder rule for cumulative constraint with profile. *Computers & Operations Research*, 172:106795, 2024.

6 Rainer Kolisch and Arno Sprecher. PSPLIB - A project scheduling problem library: OR Software - ORSEP Operations Research Software Exchange Program. *European Journal of Operational Research*, 96:205–216, 1997.

7 Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers and Operations Research*, 38:3–13, 2011.

8 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, pages 529–543. Springer Berlin Heidelberg, 2007.

9 Charles Prud'homme and Jean-Guillaume Fages. Choco-solver: A Java library for constraint programming. *Journal of Open Source Software*, 7:4708, 2022.

10 Jens Schulz. *Hybrid Solving Techniques for Project Scheduling Problems*. PhD thesis, Technischen Universität Berlin, 2013.

11 Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Explaining the cumulative propagator. *Constraints*, 16:250–282, 2011.

12 Armin Wolf and Gunnar Schrader. $\mathcal{O}(n \log n)$ Overload Checking for the Cumulative Constraint and Its Application. In *Declarative Programming for Knowledge Management*, pages 88–101. Springer Berlin Heidelberg, 2006.