Propagators for Constraint Programming - Energetic Reasoning

Exploring the Effect of Explanations for Energetic Reasoning

Konstantin Kamenov[1]
Supervisor(s): Emir Demirović[1], Imko Marijnissen[1]
[1]EEMCS, Delft University of Technology, The Netherlands

Exploring the Effect of Explanations for Energetic Reasoning

Konstantin Kamenov
Technical University Delft, The Netherlands

Abstract

The cumulative constraint is often used when modeling constraint programming problems, frequently seen in scheduling and planning problems. Energetic reasoning is one of the propagators used to enforce this constraint. However, not much has been done to explore strategies for generating explanations, which are then used by the solver for conflict analysis. This paper addresses this gap by applying strategies used in time-table edge-finding to the energetic reasoning propagator. The strategies are initial bounds relaxations and reducing the overload. Furthermore the paper compares two old strategies (naive and greedy task removal) for reducing the overload and proposes two new ones: greedy task shift and a probabilistic heuristic utilizing the knapsack problem. Results on the MiniZinc RCPSP benchmarks show that the initial bounds adjustments provide great benefit, reducing the number of conflicts by at least twenty-five percent. Reducing the overload provided a small improvement (less than five percent) and results suggest there is not much of a difference between the different strategies.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases Constraint Programming, Energetic Reasoning, Explanations

1 Introduction

Constraint programming (CP) is a powerful paradigm for solving combinatorial problems such as scheduling and planning [22, 18, 11, 1]. It has become a key tool in academic and industrial applications because of its flexibility and expressiveness. In this paper I will be tackling the Resource Constrained Project Scheduling Problem (RCPSP). In this problem we are trying to schedule several jobs, each having a duration and requiring a certain amount of a constrained resource.

The focus of this paper will be on the cumulative constraint. It makes sure that at no point in time we use more resources than the capacity we are given. It models the usage of limited resources well and has many real-world applications in construction, employee scheduling, ship loading, time tabling, processor, and production scheduling [22, 18, 11, 1]. There are many propagation strategies tackling this constraint like Overload Check [15, 29], Time-Tabling [20], Edge-Finding[21], Time-Table Edge-Finding[28], Energetic Reasoning [3] and Not-First/Not-Last Pruning [26]. The propagations of ER is considered strongest, as it implicitly covers the propagations of all these strategies except for not-first/not-last. Yet it tends to have a higher runtime.

Initially, faster but weaker propagation strategies were preferred due to ER's high time complexity of $\mathcal{O}(n^3)$ [3]. However, subsequent research has significantly improved its performance, reaching $\mathcal{O}(n^2)$ [8].

Conflict analysis is a technique that offers significant speedups by allowing us to backtrack to the first decision which caused a conflict, rather than always assuming it was the last decision which caused it. It has gained popularity within constraint programming [23, 16]. In order to ensure maximum backtracking, it is key to generate good explanations for the performed propagations. Explanations are clauses produced when a conflict or propagation occurs allowing us to pinpoint the source of the conflict. Good explanations are more general, allowing us to backtrack to earlier decision levels. There is some work on the usage of explanations to improve the performance of the energetic reasoning propagator by Heinz and Schulz [17]. It is not very in-depth, as it is research on multiple cumulative propagators, and offers only three simple strategies, but it supports the claim that good explanations for the propagator can improve runtime. Schutt [25] did work on explanations for Time Table Edge Finding and the strategies he used for better explanations can also be applied to ER. Yet, no extensive research has been done into the generation of good explanations for the ER propagator.

In this paper I will apply Schutt's strategies to ER, split into two main approaches - initial bounds adjustments and reducing the overload. ER compares the available energy in an interval, to the energy which is required for mandatory parts of tasks in this interval. If the required energy is less than the available energy, there is a conflict. The amount excess energy is called overload. For the most part they can directly be applied since TTEF is a case of ER. Furthermore I will look into specific strategies for reducing the overload and whether there is significant improvement. These strategies are the naive explanation, greedy task removal which primarily aims to reduce the number of clauses (also analyzed by Heinz), greedy task shift which primarily aims to expand the domains as much as possible (proposed in this paper) and selecting an explanation based on the highest likelihood of appearing heuristic which aims to make the explanation, which is most likely to be true if we assume uniform distribution of the variables (proposed in this paper).

The results on the MiniZinc RCPSP benchmarks sets J120(from PSPLIB) [19] and pack [9] show that the techniques from TTEF can indeed improve the ER propagator. The bounds relaxations provide a sizable advantage of more than twenty-five percent. In terms of analyzed strategies for reducing with the overload, the results suggest that they do indeed help, but do not offer a big advantage (they amount to less than five percent improvement). Also when compared to each other, they offer a similar advantage, which is a result also suggested by Schutt in his work on TTEF [25].

To summarize my contributions are as follows. Firstly, I applied the bounds relaxation strategy used in TTEF to the ER propagator. Secondly I analyzed how different strategies for reducing overload energy to get more general explanations affect the number of conflicts to achieve a solution. Finally, when analyzing these strategies I also proposed two new strategies - greedily shifting out the task with the lowest resource consumption and choosing the highest estimated likelihood explanation. The results suggest bounds relaxations offer a reduction of more than twenty-five percent on the number of conflicts, while overload reduction provides very limited advantage of less than five percent. Furthermore the non-naive overload reduction strategies offered very similar advantage.

Section 2 will outline the problem definition for the RCPSP problem. Section 3 will discuss previous work in more detail. Section 4 will act as a preliminary section. In Section 5 the main contributions of this paper will be presented. Section 6 will contain the experiments and results in support of Section 5. Lastly, section 7 will be the conclusion.

2 Problem Definition

The cumulative constraint is a global constraint, on some set of tasks I and some finite resource C. It can be defined as follows:

Let I be the set of tasks, and let $C \in \mathbb{N}_{>0}$ be the total amount of the limited resource available.

For each task $i \in I$ we have defined:

- $S_i \in \mathbb{N}$: the start time of task i,

- $E_i \in \mathbb{N}$: the end time of task i,
- $D_i \in \mathbb{N}_{>0}$: the duration of task i,
- $R_i \in \mathbb{N}_{>0}$: the resource requirement of task i.

The end time of the task is defined by the start time of the task (variable) and duration (constant) as follows:

$$\forall i \in \mathbb{I}, \quad E_i = S_i + D_i \tag{1}$$

Then for each task we will define earliest starting time $est_i = min(S_i)$, latest starting time $lst_i = max(S_i)$, earliest completion time $ect_i = min(E_i)$ and latest completion time $lct_i = max(E_i)$, which will be the smallest and largest values of the domain of S_i and E_i respectively.

The cumulative constraint enforces that the starting times of S_i of all tasks $i \in I$ do not cause an overload, or:

$$\forall t \in [0, T), \quad \sum_{\substack{i \in I \\ t \in [S_i, E_i)}} R_i \leq C \tag{2}$$

where T is the maximum time in which a task can be executed, or $T = \max_{i \in I}(lct_i)$

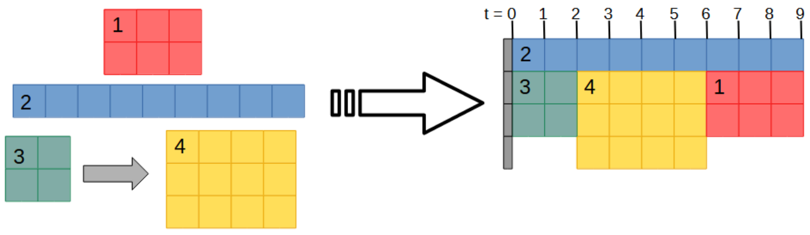

Figure 1 An example of the RCPSP. We have four tasks and within this example all start and end times are in the domain [0...12). The durations and resource consumptions of the tasks are $(3, 2), (9, 1), (2, 2), (4, 3)$ and task three precedes task four. The resulting schedule has $S_1 = 6, S_2 = 0, S_3 = 0, S_4 = 2$ and a makespan of nine.

I can now introduce the Resource Constrained Project Scheduling Problem. It is also defined on a set of tasks, but now we can have multiple constrained resources, rather than just one, each modeled with a separate cumulative constraint. In the RCPSP the tasks can also have precedence constraints, meaning some task needs to be completed before another. The goal is to select the starting times S_i of all $i \in I$ while respecting the cumulative and precedence constraints while minimizing the makespan $\max_{i \in I}(E_i)$. An example of the RCPSP with one resource with capacity of four can be seen in Figure 1.

3 Related Research

Many propagation strategies exist for the cumulative constraint, among which Overload Check [15, 29] which makes sure that no subset of tasks causes an overload in the interval required to complete these tasks (from est to lct), Time-Tabling [20] which makes use of the geometric sweep line idea to build resource consumption profiles, Edge-Finding[21] which infers mandatory precedence relations between tasks based on resource constraints, Time-Table Edge-Finding[28] which combines the techniques of time-tabling and edge-finding, Energetic

Reasoning [3] which reasons about the mandatory energy of tasks and Not-First/Not-Last Pruning [26] which builds up on edge-finding by reasoning about tasks which cannot end first or start last. The propagation done by ER implicitly includes the propagation done by all other techniques except for Not-First/Not-Last [24].

Energetic reasoning as a tool for solving the RCPSP was first proposed by Erschler, Lopez, and Thuriot (1991) [14]. The energy of a task is equal to the duration times resource consumption. ER reasons about the energy of the mandatory part of a task within an interval and the available energy of the interval. Often the geometric representation is helpful where length is duration and height is resource consumption. The propagator was first used in solver when made efficient ($\mathcal{O}(n^3)$) by Baptiste, Le Pape, and Nuijten (1999) [3]. In 2014, Derrien and Petit [12] managed to reduce the number of relevant intervals by a factor of seven. Efforts were then made by Bonifas et al. [7], Ouellet and Quimper [24], and Tesch [27] that have led to a time complexity of $\mathcal{O}(n^2 log(n))$. Finally, Carlier et al.[8] managed to reduce the complexity to $\mathcal{O}(n^2)$.

There has been some work done on explanations for the cumulative constraint and specifically for energetic reasoning by Heinz and Schulz [17] in 2011. They have noted an improvement of the runtime of the propagator with one of their three explanations strategies tested. However since the paper is about the cumulative constraint in general, no specific attention was paid to ER and the tested strategies were simple and greedy. Furthermore, Schutt [25] has proposed detailed propagation strategies for Time-Table Edge-Finding which can almost directly be applied to ER, but it has not been done yet.

Notably no extensive work has been done to analyze the potential benefits of explanations for the ER constraint. Heinz and Schulz's tests only three greedy implementations, one of which yields potential benefit. Furthermore, their main metric is runtime, which shows whether implementations are good enough, but does not give us insight on whether they have potential and can be improved. An analysis of the number of conflicts found can give us this insight. Also the effect of Schutt's adjustments have not yet been explored on the ER propagator.

4 Preliminaries

4.1 Constraint Satisfaction Problems

We can define a constraint satisfaction problem (CSP) as $\mathcal{P} = (\mathcal{X}, \mathcal{C}, \mathcal{D})$ where:

- $\mathcal{X} = x_1, x_2, ..., x_n$ is the set of variables.
- $\mathcal{D}$ is the domain for the variables. For each variable $x_i \in \mathcal{X}, \mathcal{D}(x_i)$ is the set of possible values for the variable x_i. All variables are integer variables with boolean variables being modeled with domain $0, 1$ with 1 signifying true and 0 - false.
- $\mathcal{C}$ is the set of constraints. A constraint $C(X) \in \mathcal{C}$ is a relation between some variables $X \subseteq \mathcal{X}$. The constraint I am focusing on in this paper is the cumulative constraint. An important type of constraint for conflict analysis is the atomic constraint - a predicate $a \otimes b$, where $\otimes \in \{\leq, \geq, =, \neq\}$.

When we have mapped each variable $x_i \in \mathcal{X}$ to a single value in its domain we have an assignment A. This assignment is a solution if it also satisfies all constraints. The RCPSP is a constraint optimization problem (COP). This means that on top of that we have an optimization cost function $f : \mathbb{Z}^n \mapsto \mathbb{Z}$. Now we look for an optimal solution, the one with the lowest cost (in the case of RCPSP the cost is the makespan).

4.2 Constraint Programming

Constraint Programming (CP) is a method for solving the CSPs and COPs. It utilizes solvers, where the constraints are tackled with the use of propagators - functions which prune the domains of the variables based on the current domains, removing values which cannot be a part of a solution. This repeats until no more propagation can be inferred (fixpoint), when a decision needs to be made. Solvers will recursively make these decisions (by partitioning at least one domain, giving us at least two subproblems) and the apply propagation until fixpoint occurs. This repeats either until it is determined that within the current partition no solution exists, or all variables have domains with size one (they have been assigned a value).

4.3 The Energetic Reasoning Propagator

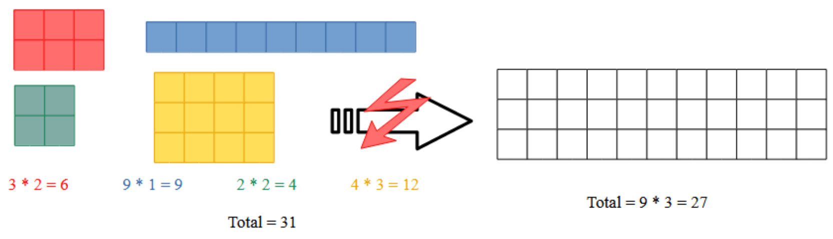

Figure 2 An example of a conflict that can be detected by the ER propagator. The total energy required to complete all the tasks is 31, while the available energy is only 27, therefore if we are ever in this state, scheduling the tasks will be infeasible.

Propagators have two main tasks - to detect conflicts, aborting the currently explored branch and to propagate (make inferences about the domains of the variables). The ER propagator compares in an interval $[a, b)$ the available energy $(b - a) \times capacity$ and the energy required for the execution of mandatory parts of a task within that interval (see Fig 2). In order to define the ER propagator, I introduce the following quantities, which define the relation of a task $i \in I$ to an interval $[t_1, t_2)$:

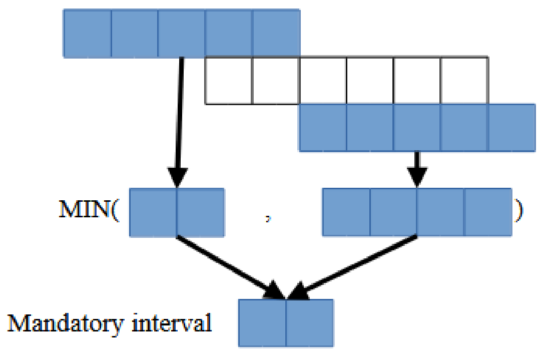

Figure 3 An example application of the left shift, right shift and mandatory interval. The given task has a LS value of 2, RS value of 4 and therefore an MI of 2.

- The left shift placement of a task, tells us how many time units we have to execute the task for within that interval, if we schedule it as early as possible.

$$(LS(i, t_1, t_2) = max(0, min(ect_i, t_2) - max(est_i, t1)) \tag{3}$$

- The right shift placement is similar to the left shift placement, but when the task is scheduled as late as possible.

$$(RS(i, t_1, t_2) = max(0, min(lct_i, t_2) - max(lst_i, t1)) \tag{4}$$

- The mandatory interval for a task within an interval is how many units of time the task needs to execute for within that interval no matter where it is placed(see Fig 3).

$$(MI(i, t_1, t_2) = min(LS(i, t_1, t_2), RS(i, t_1, t_2)) \tag{5}$$

- The available energy for execution of task i in the time interval.

$$Avail(i, t_1, t_2) = C \cdot (t_2 - t_1) - \sum_{j \in I \setminus \{i\}} R_j \cdot MI(j, t_1, t_2) \tag{6}$$

(see Fig 3) With all of these defined, we can define the consistency check and propagation of the ER propagator, similarly to Derrien and Petit [12].

▶ **Proposition 1** (as in Derrien and Petit [12] ER checker[13]). *If the condition*

$$\forall t_1, t_2 \in \mathbb{N}^2,\ t_1 < t_2 \quad C \times (t_2 - t_1) \geq \sum_{i \in \mathcal{I}} R_i \times MI(i, t_1, t_2) \tag{7}$$

does not hold, then the cumulative constraint is infeasible.

▶ **Proposition 2** (as in Derrien and Petit [12]). *For any activity i if there exists an interval $[t_1, t_2)$ such that $Avail(i, t_1, t_2) < R_i * LS(i, t_1, t_2)$, then left shift placement is not feasible and the activity can not start before $t_2 - \frac{1}{R_i} \times Avail(i, t_1, t_2)$.*

▶ **Proposition 3** (as in Derrien and Petit [12]). *For any activity i if there exists an interval $[t_1, t_2)$ such that $Avail(i, t_1, t_2) < R_i * RS(i, t_1, t_2)$, then right shift placement is not feasible and the activity can not end after $t_1 + \frac{1}{R_i} \times Avail(i, t_1, t_2)$.*

The application of proposition 1 gives us the ER checker. Proposition 2 is the rule for adjusting the lower bound of a variable's starting time and proposition 3 is the rule for adjusting the upper bound. The full (fixpoint) ER propagation is only obtained when no bounds can be further adjusted by using propositions 2 or 3 [12].

Furthermore Derrien and Petit [12] offer a classification of relevant intervals $[t_1, t2)$ that are sufficient to be checked to ensure those propositions are fully propagated.

4.4 Conflict Analysis

Conflict analysis [23, 16] is a technique which can offer substantial speedups to solvers for COPs. When a conflict is detected, it allows us to backtrack more than one level, potentially to the first decision which caused the conflict rather than just the previous decision level. At its core it works with nogoods - a conjunction of atomic predicates which leads to a conflict. While solving the problem, propagators can report the reasons for conflicts or propagation also known as an explanation. These explanations are also conjunctions of atomic predicates.

The solver can then remember and manage these explanations (nogood learning) in order to allow smarter backtracking.

In order to ensure that our explanations provide the most benefit we want to ensure that they are as general as possible, allowing for further backtracking. Transforming the initial naive explanations into more general and useful ones is known as explanation lifting. This paper will be focusing on explanation lifting for the ER propagator.

5 Main Contribution

Due to the relevance of conflict analysis in solvers [23], it is also important to generate good explanations for the propagations made. Good explanations are more general, ensuring that they can prune larger parts of the search space. Work on TTEF [25] and ER [17] shows that having these more general explanations does improve solvers.

In the following sections I will expand on the previous work done for the explanations of ER. Firstly I will show how we can relax the default naive explanations by relaxing the bounds for the tasks without affecting the energy at all. Then I will outline how we can reduce the energy overload to obtain an even more general explanation. I will then experiment with different strategies for reducing this overload, to see whether different strategies give some relevant benefit.

5.1 Bounds Relaxations

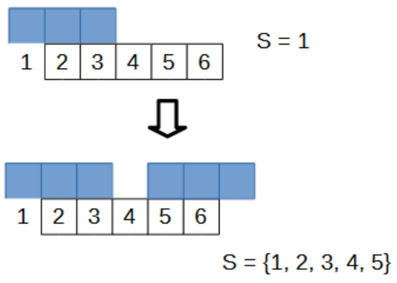

Figure 4 An example application of the bounds relaxations. A task is part of a conflict and currently has the domain {1} for its starting variable. This can be relaxed to the domain {1, 2, 3, 4, 5}, while still maintaining the same mandatory interval and therefore having the same contribution to the conflict. The picture of the state after the relaxation shows the left and right shift placements of the task with the new domain, still having MI of two.

The initial naive explanation for conflicts will be:

$$\bigwedge_{\substack{i \in I \\ MI(i,t_1,t_2)>0}} (\llbracket est_i \leq S_i \rrbracket \wedge \llbracket S_i \leq lst_i \rrbracket) \wedge \rightarrow \perp \tag{8}$$

However these clauses can be relaxed. We can adjust the bounds to some values est'_i and lst'_i, such that the minimum interval of the task for the given interval remains the same (see Fig 4). These values are $est'_i = t_1 + MI(i, t_1, t_2) - D_i$ and $lst'_i = t_2 - MI(i, t_1, t_2)$. We can then construct the explanation as follows:

$$\bigwedge_{\substack{i \in I \\ MI(i,t_1,t_2)>0}} (\llbracket est'_i \leq S_i \rrbracket \wedge \llbracket S_i \leq lst'_i \rrbracket \rightarrow \perp) \tag{9}$$

The mechanism of the explanation for the starting time propagation is similar to the one for the consistency check. For the lower bound adjustment of task i it will be:

$$\bigwedge_{\substack{j \in I \setminus \{i\} \\ MI(j,t_1,t_2)>0}} ([\![est'_j \leq S_j]\!] \wedge [\![S_j \leq lst'_j]\!]) \wedge S_i \geq est_i \rightarrow S_i \geq LB \tag{10}$$

And for the upper bound it is symmetric.

5.2 Dealing with the overload

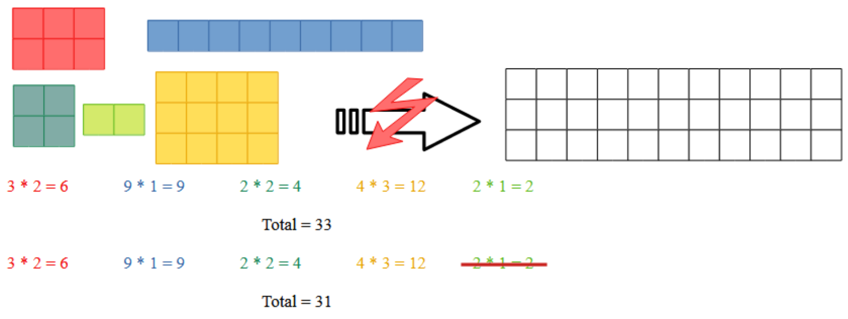

Figure 5 An example application of the adjusting the overload. The conflict is caused by excess energy of 31 and a naive explanation will include all tasks. However we can see that if we do not include the task with energy of 2 in the explanation, the conflict still holds and we have created a more general explanation for the same conflict.

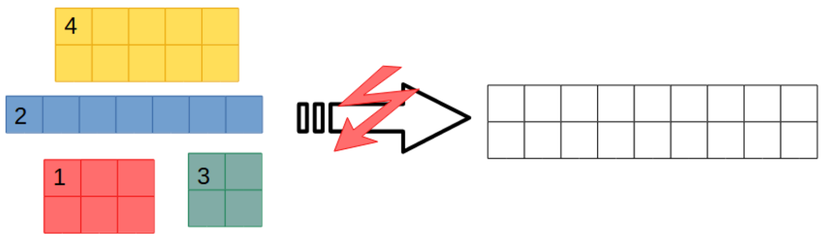

Figure 6 An example of the different overload removal strategies. The interval has available energy of 18, and total energy of 27, giving us overload of 9. Greedy task shift will try to move out task 2 by one time unit, since it has the lowest resource consumption. Greedy task removal will remove task 3, since it uses the lowest amount of energy in total. The probabilistic solution can remove any one of tasks 1, 2 and 3, depending on what the inital domains were.

Let us define the overload of an interval as:

$$OL(t_1,t_2) = \sum_{i \in I} R_i \times MI(i,t_1,t_2) - C \times (t_2 - t_1) \tag{11}$$

As long as $OL(t_1,t_2) > 0$, the interval is still overloaded. Now we can further adjust the bounds of the tasks, obtained from the bounds relaxations in the previous section for overload checking (see Fig 5). Namely, if we decrease est_i by one and increase lst_i by one, we effectively reduce $MI(i,t_1,t_2)$ by one and therefore $OL(t_1,t_2)$ by R_i. However if it is still positive, we have obtained a viable, more general explanation for the conflict. The next problem is how do we select which tasks to adjust further to reduce the overload. In this paper I will be analyzing the following strategies (differences are visualized in figure 6):

- Baseline - no adjustments
- Greedy task shift (proposed in this paper) - following up on the previous strategy, with this one we try to expand the domains of the variables as much as possible by first adjusting the tasks with lowest energy requirement R_i, allowing us to adjust as much as possible.
- Greedy task removal (proposed by Heinz[17]) - this strategy does not shift tasks, but remove them, which can also be considered a case of shifting tasks. With this strategy we remove the tasks with lowest mandatory energy first $MI(i, t_1, t_2) \times R_i$ in order to obtain an explanation with the least number of clauses. Since it only removes tasks and does not move them, if there is still leftover energy, greedy task-shift can also be applied afterward.
- Probabilistic heuristic (proposed in this paper) - from all possible explanations for the conflict, report the one which has the highest probability to occur, if we assume uniform distribution of the variables within their initial domain. That is, the probability of a variable appearing within the new domain is $\frac{|UpdatedDomain|}{|DomainBeforeFirstDecision|}$. We then take the product of these quantities for all tasks relevant to the interval. In order to tackle this approach I will use the 0/1 knapsack problem [10]. Given a set of items (w_i, v_i) where w_i is the weight of an item and v_i is the value of an item and also a maximum capacity W, it tells us which items we need to pick in order to have the maximum sum of values without exceeding the weight capacity. The weight capacity is the energy available for overload (minus one, as we cannot remove the whole overload). The weight of a task is its energy in the interval. We want to remove from the set, the tasks with lowest probability of occurring. Therefore the value of each task is the negative of the logarithm of its probability. The logarithm allows us to still use sums as in the original task, while the minus allows us to work with positive numbers, and also to use maximum as in the original task.

Dealing with the overload follows the same ideas when considering propagations and not conflicts. However the available overload energy is much lower, as we need to make sure we do not weaken the propagation, by changing the propagated bounds. For the lower bound adjustment, the value is adjusted to $t_2 - \frac{1}{R_i} \times Avail(i, t_1, t_2)$ rounded up, since when rounded down, the assignment will still be infeasible. When reducing the overload, we increase the available energy. In order to maintain the same level of propagation we can increase it by up to $R_i - 1 - Avail(i, t_1, t_2) \mod R_i$, ensuring that the result of the division remains the same.

6 Experimental Results

The code used for the experiments, results and analysis scripts can be found on Github.[1] The experimental section of this paper has four goals. First is to compare the approaches based on runtime among each other and to decomposition on a simpler test set. Decomposition outperforms the ER propagator on the simple test sets, this runtime can potentially be improved by implementing a faster version of the propagator. Second is to evaluate the impact of the bounds relaxations on the propagator. Results suggest a decrease of between ten and forty percent on the number of conflicts. Thirdly is to compare the different strategies

[1] `https://github.com/KonstantinKamenov/Pumpkin`

for reducing the overload. On the datasets, these strategies provide an advantage but it is less than five percent reduction compared to the solution using only bounds relaxations, a result which was suggested in TTEF [25]. Lastly, to provide some insight into why overload reduction provides such little improvement. As it turns out, there is not much energy to be reduced for these explanations.

The implemented solution is the improved $O(n^3)$ ER propagator by Derrien and Petit [12]. The benchmarks were run on a 12th Gen Intel(R) Core(TM) i7-12700H, 2300MHz, 14 Cores, 20 Logical Processors. Each run was isolated to run on a single performance logical processor, in high performance mode with a ten minute timeout on the complex test sets (J120 and pack) and with a two minute timeout on the simpler test sets (J60 and BL). The benchmarks I am using are PSPLIB's J60 and J120 [19], pack [9] and BL [4]. PSPLIB's instances are widely used for the RCPSP. However it is considered a more disjunctive benchmark set, therefore I am also using the pack and BL test set [5]. The set being more dsijnuctive means that less pairs of tasks can run at the same time (either due to precedence constraints or because they use too much resources together), where more cumulative test sets have more pairs of tasks which can be simultaneously executed. The paper comparing disjunctive and cumulative datasets also suggests ER techniques perform better on the cumulative sets, and disjunctive sets my be better tackled by other techniques. Therefore the results on pack and BL benchmarks are slightly more relevant. In total fifty J60, sixty-five J120, all forty BL and all fifty-five pack instances were run (for reproducibility the specific instances are mentioned in Appendix B). The model used was the RCPSP model also from the MiniZinc benchmark repository, which uses the smallest indomain search strategy. The approaches tested are:

- decomposition - running the cumulative constraint using decomposition
- naive - just using the most basic explanations without any optimizations
- relaxations - the naive approach, with bounds relaxations applied
- shift - the approach of greedily shifting out the task with lowest resource consumption, alongside the bounds relaxations strategy
- greedy - the greedy task removal approach. Since it only removes task, if there is still leftover energy, shift was also applied. This approach also utilized the bounds relaxations.
- knapsack - the probabilistic heuristic implemented using knapsack. The knapsack was implemented using the bottom-up tabulation approach, in $O(n * W)$ time and space complexity. Since it only removes task, if there is still leftover energy, shift was also applied. This approach also utilized the bounds relaxations.

The primary metric we care about is the number of conflicts since a version of the propagator with higher time complexity was implemented due to its simplicity. A lower number of conflicts signifies a better solution as it was reached in a lower number of steps. Alongside it, runtime, backtrack amount and lbd [2] will be explored. Since averages tend to be dominated by the larger values, a lot of the metrics are presented in terms of gain. That is for each instance, the result of the current solution was divided by the result of the baseline (on the best achieved common solution). Formally, the metrics are as follows:

- conflicts - The number of conflicts of the solution. Lower values (and gain less than 1.0) are better.
- runtime - The time it took the solver to reach the solution. When specified, runtime optimal means the time it took to reach the optimal solution, while runtime proven is the time it took to prove optimality. Lower values (and gain less than 1.0) are better.
- backtrack - The average backtrack amount in number of levels, when a conflict is encountered. Higher values (and gain more than 1.0) are better.

	Runtime optimal (ms)		Runtime optimal gain		Runtime proven (ms)		Runtime proven gain	
Solution	J60	BL	J60	BL	J60	BL	J60	BL
decomposition	275.59	402.43	1.000	1.000	424.32	1273.18	1.000	1.000
naive	916.22	351.73	4.295	3.195	1008.95	675.10	4.292	2.072
relaxations	925.61	199.30	4.227	2.060	1014.34	334.23	4.220	0.877
shift	895.50	193.10	4.135	2.115	974.30	320.23	4.129	0.867
greedy	912.39	195.70	4.195	2.072	997.86	324.08	4.189	0.868
knapsack	916.32	197.78	4.211	2.077	995.34	329.03	4.204	0.887

Table 1 The runtime results of the solutions on the J60 and BL benchmark sets, with decomposition used as baseline for calculating the gains

	Conflicts gain		Runtime gain		Backtrack gain		LBD gain	
Solution	J120	pack	J120	pack	J120	pack	J120	pack
naive	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
relaxations	0.882	0.668	0.940	0.663	1.382	1.057	1.171	0.956

Table 2 Conflicts, runtime, backtrack and lbd gain on the completed instances from J120 and pack datasets. Naive was used as baseline to calculate the gains.

- lbd - The lbd of the solution. The lbd [2] keeps track of the smallest number of decision levels ever seen for the nogood that cause it to propagate. Lower values (and gain less than 1.0) are better.

For experiment four, different metrics were used.

- no energy - The percentage of intervals where no energy could be removed using overload reduction.
- no change - The percentage of intervals where no energy ended up being removed (this includes no energy intervals).
- no removal - The percentage of intervals where no task could be removed (this also includes no energy and no removal intervals).
- normalized overload - To get an idea of how much the tasks could be moved, for an interval the available overload to be removed was divided by the lowest resource consumption of a task in this interval. For example a score of one means the lowest consumption task could be moved out by only one time unit.

As we can see in the results from table 1 on the J60 test set decomposition heavily outperforms all ER solutions, performing four times faster (only forty-four instances of J60 were considered as the others did not terminate in the two minutes for at least one solution). On the other hand, on the more cumulative test set decomposition arrives at the optimal solution more than two times faster, yet solutions using bounds relaxation manage to prove optimality quicker. This performance could be improved by implementing faster versions of the ER propagator.

Based on tables 2 and 3 we can conclude that the bounds relaxation strategy is indeed very effective. Table 2 includes only completed (proven) instances by all compared solutions, which were twenty-six for the J120 set and only eight for the pack set, so these results are not entirely reliable and more testing should be done to explore the effects on proving optimality. For finding the optimal solution bounds relaxations provide more than twenty-five percent

	Conflicts gain		Runtime gain		Backtrack gain		LBD gain	
Solution	J120	pack	J120	pack	J120	pack	J120	pack
naive	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
relaxations	0.688	0.745	0.805	0.773	1.254	1.036	1.295	1.109

Table 3 Conflicts, runtime, backtrack and lbd gain on all selected instances from J120 and pack datasets. The gains were obtained by comparing the values for the lowest common makespan found by the solutions. Naive was used as baseline to calculate the gains.

	Conflicts gain		Runtime gain		Backtrack gain		LBD gain	
Solution	J120	pack	J120	pack	J120	pack	J120	pack
relaxations	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
shift	0.986	0.980	1.000	0.996	1.048	1.002	0.964	0.990
greedy	0.982	0.981	1.000	0.986	1.054	1.002	0.956	0.992
knapsack	0.969	0.977	0.992	0.989	1.047	1.002	0.960	0.992

Table 4 Conflicts, runtime, backtrack and lbd gain on the completed instances from J120 and pack datasets. Relaxations was used as baseline to calculate the gains.

reduction in the number of conflicts. An interesting result is that the relaxed solutions tend to have higher lbd (except for the completed pack instances), however they also backtrack more. We can conclude that despite the naive solution having many clauses on the same level, they are still deep in the search tree therefore they still do not provide a big backtracking advantage.

The results from tables 4 and 5 suggest overload reduction strategies can provide an advantage, but it is a small advantage. Table 4 includes only completed (proven) instances by all compared solutions, which were twenty-seven for the J120 set and thirteen for the pack set, so these results are not entirely reliable and more testing should be done to explore the effects on proving optimality. All overload reduction strategies provide a similar advantage - about four percent reduction on J120 and about one-and-a-half percent on pack. The knapsack solution performed best, followed by greedy and shift solutions. The LBD and backtrack results are also similar, but they follow the expected trend of better solutions having higher backtrack amount and lower lbd.

The results from table 6 are not meant to serve as a comparison between the methods, but rather to show that the metrics are pretty similar across the strategies. Twenty-four

	Conflicts gain		Runtime gain		Backtrack gain		LBD gain	
Solution	J120	pack	J120	pack	J120	pack	J120	pack
relaxations	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
shift	0.965	0.988	0.988	0.999	1.026	1.001	0.981	0.993
greedy	0.964	0.984	0.989	0.988	1.029	1.001	0.975	0.992
knapsack	0.955	0.984	0.974	0.985	1.026	1.001	0.976	0.991

Table 5 Conflicts, runtime, backtrack and lbd gain on all selected instances from J120 and pack datasets. The gains were obtained by comparing the values for the lowest common makespan found by all solutions. Relaxations was used as baseline to calculate the gains.

	No energy		No removal		No change		Normalized overload	
Solution	J120	pack	J120	pack	J120	pack	J120	pack
shift	24.1%	40.6%	100%	100%	70.2%	84.8%	0.981	0.817
greedy	24.1%	40.6%	84.2%	96.9%	70.3%	84.8%	0.975	0.815
knapsack	24.1%	40.6%	84.6%	96.9%	70.1%	84.8%	0.975	0.819

Table 6 Overload metrics for all overload reduction strategies

percent of the J120 and 40 percent of the pack benchmarks did not have any overload to be reduced at all. Eighty-four for J120 and ninety-seven for pack did not allow for any task to be removed at all, meaning these strategies just performed the shift in these explanations. Shift has one hundred percent because it does not remove tasks in the first place. Seventy percent for J120 and eighty-five for pack did not allow for any change to happen even when using the shift strategy (meaning no task at all could be moved or removed), meaning no overload reduction was done for these explanations. And the normalized overload for the explanations is less than one, showing once again that on average the explanations allowed for very little adjustment to be done on the tasks. We can also see that the more cumulative set had less overload amounts in general. This is understandable since it has fewer precedence constraints, meaning tasks are less limited and therefore mandatory intervals are smaller. Also, since more pairs of tasks can be executed simultaneously, having a conflict will result in a smaller overload since tasks use a smaller part of the available energy.

7 Conclusion

In this paper I have delved deeper into explanations for the energetic reasoning cumulative propagator. I took what was done before for time-table edge-finding and adjusted it to the ER propagator. I also explored different possible strategies to deal with the overload in the explanations to generate more general explanations. The results suggest that the bounds relaxation strategy should be used and provides a big advantage, having twenty-five percent lower number of conflicts on average. On these datasets overload reduction did not have a big impact (less than five percent improvement). The best performing solution was the task removal, but the performance of all overload reduction strategies was fairly close.

Future work can focus on exploring more strategies for dealing with the overload. One common strategy that was not explored in this paper is shifting tasks based on the number of conflicts they have been a part of recently, to ensure the most general clauses. Another aspect that can be improved is the algorithms for the strategies. The best strategies make use of sorting, and the even slower knapsack, so implementing these algorithms efficiently is required to provide a boost in runtime and not only the number of conflicts. Finally, the runtime comparison can be analyzed using an ER propagator with lower complexity, to give better impression of the actual improvement provided by the explanations.

References

1 Christian Artigues, Sophie Demassey, and Emmanuel Néron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications.* 01 2008.

2 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. pages 399–404, 07 2009.

3 Ph Baptiste, Claude Le Pape, and Wim Nuijten. Satisfiability tests and time-bound adjustmentsfor cumulative scheduling problems. *Annals of Operations research*, 92(0):305–333, 1999.

4 Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1-2):119–139, 2000.

5 Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1):119–139, Jan 2000. doi:10.1023/A:1009822502231.

6 Nicolas Beldiceanu, Pierre Flener, Jean-Noël Monette, Justin Pearson, and Helmut Simonis. Toward sustainable development in constraint programming. *Constraints*, 19(2):139–149, Apr 2014. doi:10.1007/s10601-013-9152-4.

7 Nicolas Bonifas. Ao (n2log (n)) propagation for the energy reasoning. In *Conference paper, ROADEF*, volume 2016, 2016.

8 J. Carlier, E. Pinson, A. Sahli, and A. Jouglet. An o(n2) algorithm for time-bound adjustments for the cumulative scheduling problem. *European Journal of Operational Research*, 286(2):468–476, 2020. URL: https://www.sciencedirect.com/science/article/pii/S037722172030312X, doi:10.1016/j.ejor.2020.03.079.

9 Jacques Carlier and Emmanuel Néron. On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, 149(2):314–324, 2003. doi:10.1016/S0377-2217(02)00763-4.

10 George B Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

11 Erik Demeulemeester and Willy Herroelen. *Project Scheduling: A Research Handbook.* 01 2002.

12 Alban Derrien and Thierry Petit. A new characterization of relevant intervals for energetic reasoning. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming*, pages 289–297, Cham, 2014. Springer International Publishing.

13 Jacques Erschler, Pierre Lopez, and Catherine Thuriot. Scheduling under time and resource constraints. In *Proc. of Workshop on Manufacturing Scheduling, 11th IJCAI, Detroit, USA*, 1989.

14 Jacques Erschler, Pierre Lopez, and Catherine Thuriot. Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnancement. *Revue d'intelligence artificielle*, 5(3):7–32, 1991.

15 Hamed Fahimi and Claude-Guy Quimper. Linear-time filtering algorithms for the disjunctive constraint. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

16 Thibaut Feydy and Peter J. Stuckey. Lazy clause generation reengineered. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, pages 352–366, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

17 Stefan Heinz and Jens Schulz. Explanations for the cumulative constraint: An experimental study. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms*, pages 400–409, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

18 Rainer Kolisch. *Make-to-Order Assembly Management.* 01 2001. doi:10.1007/978-3-662-04514-5.

19 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(1):205–216, 1997. URL: https://www.sciencedirect.com/science/article/pii/S0377221796001701, doi:10.1016/S0377-2217(96)00170-1.

20 Arnaud Letort, Nicolas Beldiceanu, and Mats Carlsson. A scalable sweep algorithm for the cumulative constraint. In Michela Milano, editor, *Principles and Practice of Constraint Programming*, pages 439–454, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

21 Luc Mercier and Pascal Van Hentenryck. Edge finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1):143–153, 2008.

22 Klaus Neumann and Christoph Schwindt. Activity-on-node networks with minimal and maximal time lags and their application to make-to-order production. *Operations-Research-Spektrum*, 19(3):205–217, Sep 1997. `doi:10.1007/BF01545589`.

23 Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, Sep 2009. `doi:10.1007/s10601-008-9064-x`.

24 Yanick Ouellet and Claude-Guy Quimper. A checker and filtering algorithm for the energetic reasoning. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 477–494. Springer, 2018.

25 Andreas Schutt, Thibaut Feydy, and Peter J. Stuckey. Explaining time-table-edge-finding propagation for the cumulative resource constraint. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 234–250, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

26 Andreas Schutt and Armin Wolf. A new ${\mathcal o}(n^2\log n)$*not − first/not − lastpruningalgorithmforcumulativeresourceconstraints.InDavidCohen, editor, PrinciplesandPracticeofConstraint*
−459, *Berlin, Heidelberg*, 2010.*SpringerBerlinHeidelberg*.

27 Alexander Tesch. Improving energetic propagations for cumulative scheduling. In *International conference on principles and practice of constraint programming*, pages 629–645. Springer, 2018.

28 Petr Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 230–245. Springer, 2011.

29 Armin Wolf and Gunnar Schrader. overload checking for the cumulative constraint and its application. In *International Conference on Applications of Declarative Programming and Knowledge Management*, pages 88–101. Springer, 2005.

A Responsible Research

In this section I will tackle the the ethical implications of this project. Overall there are not any significant ethical issues, but here are some of the main points:

- Data and privacy - This project does not deal with processing or storing user data in any way. The datasets used for analysis are public and commonly used for benchmarking purposes. This project did not use any data for training.
- Reproducibility - The project contains no (unseeded) randomness. The benchmarks to be run were randomly selected, but they will be provided in appendix A.
- Bias - Naturally I can be biased towards a positive result regarding the introduced novelties. Although this cannot be mitigated, The results are presented as is, without adjusting or cherry-picking data.
- Use of AI - throughout this research project AI tools were used for assist with coding. ChatGPT was used to help fix issues with coding in Rust, and Google Gemini was used to generate the scripts for parsing and analyzing the results. Those scripts were later checked by the author. For writing the report, AI was only used for the sake of spell and grammar checking.
- Sustainability - A great analysis of the open challenges associated with the sustainability of CP are presented by Beldiceanu [6].

B Benchmark Sets

In this section, I will provide a list of the tests of each dataset which was used to obtain the experimental results using the MiniZinc benchmarks repository.

- J120: J120_10_5, J120_11_2, J120_13_6, J120_15_4, J120_16_1, J120_16_2, J120_16_3, J120_18_4, J120_1_2, J120_20_8, J120_22_3, J120_22_4, J120_25_10, J120_25_5, J120_26_10, J120_26_6, J120_27_1, J120_27_6, J120_27_8, J120_28_3, J120_29_10, J120_29_9, J120_31_10, J120_31_7, J120_32_2, J120_32_3, J120_32_4, J120_32_5, J120_32_8, J120_32_9, J120_33_1, J120_33_3, J120_35_4, J120_38_4, J120_39_10, J120_3_1, J120_3_10, J120_3_4, J120_41_6, J120_42_2, J120_42_6, J120_43_4, J120_44_6, J120_44_7, J120_45_10, J120_45_2, J120_47_6, J120_48_8, J120_49_8, J120_4_3, J120_4_8, J120_50_9, J120_52_5, J120_53_7, J120_53_8, J120_54_7, J120_56_9, J120_57_8, J120_58_5, J120_58_8, J120_5_4, J120_6_1, J120_6_6, J120_9_2, J120_9_3
- J60: J60_10_8, J60_11_10, J60_11_9, J60_12_4, J60_12_7, J60_12_8, J60_14_4, J60_15_6, J60_18_3, J60_19_2, J60_19_9, J60_20_4, J60_20_7, J60_21_10, J60_23_7, J60_24_6, J60_24_9, J60_26_1, J60_26_10, J60_2_10, J60_2_6, J60_31_8, J60_32_10, J60_32_9, J60_33_1, J60_34_3, J60_36_2, J60_36_7, J60_38_6, J60_39_7, J60_3_10, J60_3_6, J60_41_8, J60_43_3, J60_44_1, J60_44_2, J60_44_9, J60_45_4, J60_46_2, J60_48_1, J60_48_6, J60_48_7, J60_5_10, J60_5_8, J60_6_6, J60_6_8, J60_7_9, J60_8_1, J60_8_4, J60_8_9
- pack: all 55
- BL: all 40 instances