Time Table Edge Finding with Energy Variables

Moli Yang[1], Andreas Schutt[1,2], and Peter J. Stuckey[2,3]

[1] University of Melbourne, Australia
[2] Data61 CSIRO, Australia
[3] Monash University, Australia

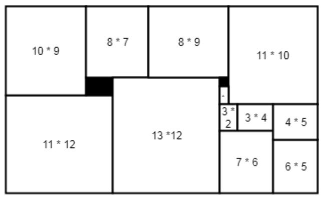
Abstract. Cumulative resource constraints can model scarce resources in scheduling problems or a dimension in packing and cutting problems. In order to efficiently solve such problems with a constraint programming solver, it is important to have strong and fast propagators for cumulative resource constraints. In this paper, we develop a time-table edge-finding energy propagator for cumulative constraint which can reason more strongly based on energy. We give results using this propagator in a lazy clause generation system on rectangle packing and evacuation scheduling problems. We are able to prune the search space and reduce solve time compared with a time-table or time-table edge-finding propagator.

1 Introduction

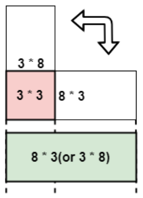
The `cumulative` constraint models the use of a limited resource over time in executing a series of tasks requiring the resource. The resource may be a set of machines, a group of workers, entities like power supply or even a dimension in a packing or cutting problem. Because of its broad modelling capability the `cumulative` constraint has been widely used in many industrial scheduling. Hence it is important to have strong and fast propagation techniques for the `cumulative` constraint so that constraint programming (CP) solvers can detect inconsistency and remove invalid values for the domains of the variables involved more efficiently. Moreover, for CP solvers that incorporate nogood learning [7], it is also important to generate strong reusable explanations for the reasoning of the `cumulative` constraint.

Vilim [10] developed TTEF propagation combining time-table propagation [1], which is usually superior for *highly disjunctive* problems, and edge-finding propagation [2], which is more appropriate for *highly cumulative* problems, in order to perform stronger propagation while having a low runtime overhead. Vilim shows that on a range of highly disjunctive project scheduling problems, TTEF propagation can generate lower bounds on the project deadline that are superior to previous methods. He used a CP solver without nogood learning.

Schutt *et al* [9] extended TTEF propagation for use in a lazy clause generation (LCG) CP solver [7] by showing how toe explain is propagation. LCG solvers are state of the art for solving many problems involving cumulative constraints.



(a) An optimal packing of the rectangle of size 1×2, 2×3, $\ldots 12 \times 13$ where rotation is allowed.



(b) The two rotations of a rectangle and its energy usage

Fig. 1: (a) rectangle packing and (b) the loss of information when only using duration and resource usage variables

Their results shows that TTEF performs well in both lowering runtime and reducing search space or highly cumulative scheduling problems. However, the stronger propagation does not generally pay off for highly disjunctive problems.

An example of the usage of the `cumulative` constraint is in optimal rectangle packing [6], which is, given a set of rectangles, find the minimum area of a rectangle containing all rectangles without overlap. The cumulative constraint is used as a redundant constraint to constrain the maximal usage of height, when considering each rectangle as a task of duration length, and resource usage height; and similarly to constraint the maximum usage of width, when considering each rectangle as a task of duration height, and resource usage length. Note that the `cumulative` constraint provides very strong propagation in the case that the orientation of the rectangle is fixed, so the length and the height are known. But if we allow rectangles to be rotated, then we do not know the length and height of the rectangle, since each has two possibilities (unless the rectangle is a square).

Example 1. Consider a set of rectangles of sizes 1×2, 2×3, ... , up to 12×13, where rectangles may be rotated by 90-degree. Figure 1a shows the optimal solution in a 21×35 bounding box. □

If we consider the rectangle packing problem we can immediately see a weakness of a cumulative constraint that reasons using only start times, durations and resources usages. When we consider packing a rectangle of dimensions $w \times h$ whose orientation is not fixed then the minimum duration is $\min(w, h)$ and the minimum resource usage is $\min(w, h)$ and hence the overall minimum energy utilization is $\min(w, h)^2$, whereas we know the energy utilization is always exactly $w \times h$. With explicit energy usage variables, we can make use of the much larger lower bound on energy usage, and hence hope to propagate more.

Example 2. Figure 1b illustrates this phenomena explicitly by packing a 3×8 rectangle into an interval which is at least 8 long. Without knowing the orientation of the rectangle the lower bound on duration and resource usage are both

3, for a minimum resource usage of 9. But since the entire rectangle fits in the interval whatever rotation we know the energy usage is exactly 24. □

In this paper we define a `cumulative` propagator that uses energy variables in a time-table edge-finding propagation algorithm; we show how to explain its propagation; and we compare it against time-table and time-table edge-finding propagators.

2 Cumulative resource constraint with energy variables

In cumulative resource scheduling, a set of (non-preemptive) tasks $\mathcal{V}$ and one cumulative resource with a (constant) resource limit L is given where a *task* i is specified by its *start time* S_i, its *duration* D_i, its *resource usage* R_i, its *energy* $E_i = D_i \cdot R_i$. In this paper we assume each of S_i, D_i, R_i and E_i may be an integer variable and L is assumed to be an integer constant.

We assume a set of integer times τ and use notation $[t_1, t_2)$ to indicate the period starting at time t_1 and finishing (non-inclusive) at time t_2. We define est_i (d_i^{min}, r_i^{min}, e_i^{min}) and lst_i (d_i^{max}, r_i^{max}, e_i^{max}) as the current *lower* and *upper* bounds of start time (duration, resource usage, energy respectively) of i. Further, we define the *earliest completion time* $ect_i \leftarrow est_i + d_i^{min}$, and the *latest completion time* $lct_i \leftarrow lst_i + d_i^{max}$.

The cumulative resource constraint with energy $\texttt{cumulative}(S, D, R, E, L)$ is characterized by the set of tasks $\mathcal{V}$ and a cumulative resource with resource capacity L. The constraint is satisfied by finding a solution that assigns values to each of the start time variables S_i, duration variables D_i, resource usage variables R_i and energy usage variables E_i ($i \in \mathcal{V}$), so that the following conditions hold.

$$\begin{aligned}
&est_i \leq S_i \leq lst_i, && \forall i \in \mathcal{V} \\
&d_i^{min} \leq D_i \leq d_i^{max}, && \forall i \in \mathcal{V} \\
&r_i^{min} \leq R_i \leq r_i^{max}, && \forall i \in \mathcal{V} \\
&e_i^{min} \leq E_i \leq e_i^{max}, && \forall i \in \mathcal{V} \\
&R_i \times D_i = E_i, && \forall i \in \mathcal{V} \\
&\sum_{i \in \mathcal{V} : \tau \in [S_i, S_i + D_i)} R_i \leq L && \forall \tau
\end{aligned}$$

where τ ranges over the time periods considered. Note that this problem is NP-hard [5].

3 Time-table edge-finding propagation with energy variables

The basic idea of TTEF propagation is to treat a task as a fixed part (used in time-table propagation) and a free part and to determine the range of start times based on the energy available from the resource and the energy required for the tasks in specific time windows.

Time-table edge-finding [10,9] calculates the amount of energy $e_i(a,b)$ that must be used by a task i in the time window between two time points a and b. The TTEF calculation for $e_i(a,b)$ without energy variables is given by

$$e_i(a,b) := \begin{cases} d_i^{min} \times r_i^{min}, & a \leq est_i \wedge lct_i \leq b \\ \max(0, b - lst_i) \cdot r_i^{min} & a \leq est_i \wedge lct_i > b \\ \max(0, \min(b, ect_i) - \max(a, lst_i)) \cdot r_i^{min} & otherwise \end{cases}$$

The first case is when the entire task must occur in the time window, here we can use the lower bound on the total energy of the task given by $d_i^{min} \times r_i^{min}$. The second case is when the task partially overlaps and some parts might run after the time window, here we use the minimum duration of the overlap times the minimum resource usage. The third case is for all others for which we only consider the minimum energy from the overlapping compulsory part of the task.

The weakness of the usual TTEF formulation without energy variables is that the lower bound of energy of a task $d_i^{min} \times r_i^{min}$ can be very weak, as shown in Example 2. When we have energy variables we can calculate minimum energy usage within a time window more accurately.

$$e_i(a,b) := \begin{cases} e_i^{min} & a \leq est_i \wedge lct_i \leq b \\ \max(0, b - lst_i) \cdot r_i^{min} & a \leq est_i \wedge lct_i > b \\ \max(0, \min(b, ect_i) - \max(a, lst_i)) \cdot r_i^{min} & otherwise \end{cases}$$

Note that only the first case for the TTEF calculation changes. We assume that the product constraint $E_i = D_i \times R_i$ is separately propagated so $e_i^{min} \geq d_i^{min} \times r_i^{min}$.

3.1 Consistency check with energy variables

The consistency check is the part of TTEF energy propagation that checks if there is a resource overload in any task interval. Time-table edge finding splits the total energy of a task into a *fixed part* $e_i^{fix} \leftarrow r_i^{min} \cdot (lst_i - ect_i)$ and a *free part* $e_i^{free} \leftarrow e_i^{min} - e_i^{fix}$. Let $\mathcal{V}^{Free}$ be the set of tasks with a non-empty free part $\{i \in \mathcal{V} \mid e_i^{free} > 0\}$. The use of energy variables simply allows us to have a better estimation of the least energy used by a task within a time window $[a,b)$.

Proposition 1 (Consistency Check). *The cumulative resource scheduling problem is inconsistent if* $R \cdot (b-a) - energy(a,b) < 0$ *where* $energy(a,b) := \sum_{i \in \mathcal{V}} e_i(a,b)$

This check can be done in $\mathcal{O}(l^2 + n)$ runtime [10], where $l = |\mathcal{V}^{Free}|$, if the resource profile is given.

The algorithm for the consistency check is shown in Algorithm 1. The difference from TTEF is that for a task i, if lct_i is later then the end time, we can

Algorithm 1 TTEF_En consistency check algorithm

```
1: procedure TTEF_En consistency check
2:     $end \leftarrow \infty; minAvail \leftarrow \infty$
3:     for $y \leftarrow n$ down to 1 do
4:         $b \leftarrow Y[y]$
5:         if $lct_b = end$ then $continue$
6:         if $end \neq \infty$ and $minAvail \neq \infty$ and $minAvail \geq R \cdot (end - lct_b) - ttEn(lct_b, end)$ then $continue$
7:         $end \leftarrow lct_b$; $En^{free} \leftarrow 0; minAvail \leftarrow \infty$
8:         for $x \leftarrow n$ down to 1 do
9:             $a \leftarrow X[x]$
10:            if $end \leq est_a$ then $continue$
11:            $begin \leftarrow est_a$
12:            $eMin = max(e_a^{min}, d_a^{min} \times r_a^{min})$
13:            if $lct_a \leq end$ then
14:                $En^{free} \leftarrow En^{free} + eMin - e_a^{fix}$
15:            else
16:                $enIn^{fix} \leftarrow max(0, min(end, ect_a) - lst_a) \times r_a^{min}$
17:                $enIn^{free} \leftarrow min(e_a^{free}, max(0, r_a^{min} \times (end - lst_a) - enIn^{fix}))$
18:                $En^{free} \leftarrow En^{free} + enIn^{free}$
19:            $En^{avail} \leftarrow R \cdot (end - begin) - En^{free} - ttEn(a, b)$
20:            if $En^{avail} < 0$ then
21:                $explainOverload(begin, end)$
22:                return $false$
23:            if $En^{avail} < minAvail$ then $minAvail \leftarrow En^{avail}$
```

take all its free energy into account; if not, we use the part of free energy lying between the time interval. The differences from the algorithm of Schutt *et al* [9] are shown in blue.

In order to use the cumulative propagator within a CP solver with nogood learning [7] the propagator needs to be able to explain the *reason* for failures of the consistency check. That is it needs to determine a set of facts true about the current domain D which ensure that the consistency check leads to failure.

We need to explain for each task i where $e_i(a,b) > 0$ why its energy usage was at least $e_i(a,b)$. Hence, for task i, the start time should be larger than $a - \left\lfloor \frac{e_i^{min} - e_i(a,b)}{r_i^{max}} \right\rfloor$ and less than $b - \left\lceil \frac{e_i(a,b)}{r_i^{min}} \right\rceil$. And also, the resource usage should be less than r_i^{max} and larger than r_i^{min} because if not, the energy of task i which lie in the time window could be less $e_i(a,b)$. In summary, the explanation should be of the form.

$$\bigwedge_{i \in \mathcal{V}: e_i(a,b) > 0} [\![a - \left\lfloor \frac{e_i^{min} - e_i(a,b)}{r_i^{max}} \right\rfloor \leq S_i]\!] \wedge [\![S_i \leq b - \left\lceil \frac{e_i(a,b)}{r_i^{min}} \right\rceil]\!]$$
$$\wedge [\![r_i^{min} \leq R_i]\!] \wedge [\![R_i \leq r_i^{max}]\!] \wedge [\![e_i^{min} \leq E_i]\!] \rightarrow \bot$$

To further generalize the explanation, we can make use of the overload energy $\varDelta := energy(a, b) - R \cdot (b - a) - 1$, if $\varDelta > 0$. Since the overload occurs even if some tasks use less energy we can give some allowable reduction δ_i in the energy used in the time window for each task i and still use up too much energy in the time window. We choose δ_i such that $\sum_{i \in \mathcal{V}: e_i(a,b)>0} \delta_i = \varDelta$. For task i, if $\delta_i \geq e_i(a, b)$ then we can remove the task i completely from the explanation. Otherwise the start time lower bound and upper bound for the explanation can be relaxed to $a - \frac{e_i^{min} - e_i(a,b) + \delta_i}{r_i^{max}}$ and $b - \frac{e_i(a,b) - \delta_i}{r_i^{min}}$, respectively.

3.2 Start time lower bound propagation with energy variables

Propagation on the lower and upper bounds of the start time variables S_i are basically symmetric; consequently we only discuss the case for the lower bounds' propagation.

To prune the start time lower bound of an task u, TTEF_EN checks if there is an overload when task u starts at its earliest start time in a time window $[a, b)$. Vilim [10] considers four positions of u relative to the time window. In our case, the four positions should be defined as *right* ($a \leq est_u < b < ect_u$), *inside* ($a \leq est_u < ect_u \leq b$), *through* ($est_u < a \wedge b < ect_u$), and *left* ($est_u < a < ect_u \leq b$).

For *right* and *inside* task u, we define $e_u^{est}(a, b) := min(e_u^{min}, r_u^{min} \times (min(est_u + d_u^{max}, b) - est_u))$ as the minimum energy used when u starts at its earliest start time and $e_u^{max}(a, b)$ as the maximum available energy remaining in the time window when u is left out. The update rule for the *right* and *inside* task u, illustrated in Figure 1, is

$$R \cdot (b - a) - (energy(a, b, u) + e_u^{est}(a, b)) < 0 \rightarrow b - \left\lfloor \frac{e_u^{max}(a, b)}{r_u^{min}} \right\rfloor \leq S_u \quad (1)$$

where $energy(a, b, u) := energy(a, b) - e_u(a, b)$ and $e_u^{max}(a, b) := R \times (b - a) - energy(a, b, u)$. We omit the propagation algorithm for space reasons, its similar to that shown in [9].

To explain the propagation of new bound nLB for task u, the principle is that we decrease the lower bound on the left hand side as much as possible so that the same propagation holds. For the task u, we can push the explanation of lower bound to left until the minimum energy lying in the time window just equals to $e_u^{max}(a, b)$. And also, for all other tasks, we can perform the similar generation discussed in the case of resource overload.

$$\begin{aligned}
\bigwedge_{i \in \mathcal{V} \setminus \{u\}: e_i(a,b)>0} & (\llbracket a - \left\lfloor \frac{e_i^{min} - e_i(a,b)}{r_i^{max}} \right\rfloor \leq S_i \rrbracket \wedge \llbracket S_i \leq b - \left\lceil \frac{e_i(a,b)}{r_i^{min}} \right\rceil \rrbracket \\
& \wedge \llbracket r_i^{min} \leq R_i \rrbracket \wedge \llbracket R_i \leq r_i^{max} \rrbracket \wedge \llbracket e_i^{min} \leq E_i \rrbracket) \wedge \llbracket e_u^{min} \leq E_u \rrbracket \\
& \wedge \llbracket a - \left\lfloor \frac{e_u^{min} - e_u^{max}(a,b)}{r_u^{max}} + 1 \right\rfloor \leq S_u \rrbracket \wedge \llbracket r_u^{min} \leq R_u \rrbracket \wedge \llbracket R_u \leq r_u^{max} \rrbracket \\
& \rightarrow \llbracket nLB \leq S_u \rrbracket
\end{aligned}$$

consecutive	12		13		14		15		16	
	conf	time	conf	time	conf	time	conf	time	conf	time
tt	11250	9.21	33556	23.91	111749	71.85	306233	191.01	∞	∞
TTEF	13912	9.76	30705	20.91	72670	45.64	216442	144.92	885743	581.41
TTEF_EN	8367	7.23	23836	15.36	59998	37.57	157539	113.56	806538	503.17
double	7		8		9		10		11	
perimeter	conf	time	conf	time	conf	time	conf	time	conf	time
tt	2251	0.90	12195	3.55	41203	16.88	106326	62.34	∞	∞
TTEF	2200	1.18	10550	5.08	39329	17.06	93773	61.69	817887	815.26
TTEF_EN	1979	0.91	8035	3.40	25127	9.57	55212	34.56	454654	361.18
free	6		7		8		9		10	
	conf	time	conf	time	conf	time	conf	time	conf	time
tt	1896	1.71	6965	6.88	155992	119.72	594878	488.63	∞	∞
TTEF	1700	1.65	8001	7.74	153885	118.31	546442	457.52	∞	∞
TTEF_EN	1712	1.68	6237	5.76	96074	80.68	446776	311.91	∞	∞

Table 1: Results for rectangle packing

4 Experimental Evaluation

We now compare our solution approach TTEF_EN to both time-table (`tt`), time-table edge-finding TTEF propagation. We compare conflicts and average time for 10 runs, ∞ indicates all runs fail to prove optimality in time. The experiments were run on a X86-64 architecture running MacOS 10.13 and a Intel Core m3 CPU processor at 1.2 GHz. We set the timeout for each run as 1800 sec. All models and data are available at `people.eng.unimelb.edu.au/pstuckey/ttefen`.

Rectangle Packing problems [6] are highly cumulative and hence good examples for TTEF propagation. We compare three different versions: (a) consecutive rectangle packing [6], where instance N is the set of rectangles of size $1\times2, 2\times3, ...,$ up to $N \times (N+1)$ that may be rotated. (b) double-perimeter rectangle packing [6], where instance N is the set of rectangles of size $1 \times (2N-1), 2 \times (2N-2), ...,$ up to $N \times (N+1)$ that may be rotated. (c) free rectangle packing, where instance N is a set of rectangles constrained to take areas to be $1 \times (2N-1), 2 \times (2N-2), ...,$ up to $N \times (N+1)$ with any height and width giving the correct area. The results using default activity based search are shown in Table 1. Clearly TTEF_EN propagation is superior to the alternatives, and its advantage grows with problem size. We also compared using fixed search (not shown) where TTEF_EN was also superior, but not by as much.

Evacuation Planning problems [4] try to schedule evacuation tasks so everyone is evacuated as quickly as possible. Cumulative constraints constrain the flow rates r_i of evacuation tasks on road segments. The total energy of a task i is the number n_i of evacuees constrained so that the $d_i \times r_i \geq n_i$. The results using default search are shown in Table 2, where N is the number of evacuation zones, we use 10 randomly generated instances for each N and show average

	9		10		11		20	30	40	50
	conf	time	conf	time	conf	time	evac	evac	evac	evac
tt	2069.8	218.81	6962.7	684.59	1830.5	490.84	9326.4	17348.3	21941.5	27926.0
TTEF	2306.2	228.42	7108.7	675.45	2437.4	551.64	8995.8	16968.1	21909.9	27925.3
TTEF_EN	1993.9	219.21	5817.8	578.43	1646.1	448.05	8959.7	16851.6	21884.9	27879.7

Table 2: Evacuation problem.

results. For small examples where all methods can prove optimality, we compare conflicts and time. For larger examples we simple compare minimal evacuation time (evac) at time out. The smaller results show that energy variables improve the number of conflicts and time (except the smallest example). Interestingly here TTEF does not beat `tt` in terms of conflicts or time. For larger results we see TTEF is superior to `tt` and bettered by TTEF_EN.

5 Conclusion and Related Work

The addition of energy variables to the `cumulative` constraint allows us to improve any energy based reasoning approach for `cumulative`. The experiments show that in problem classes where TTEF propagation is effective, the version using energy variables TTEF_EN is even more effective.

Note that a number of versions of the `cumulative` constraint appearing in the CHIP system [8] included energy variables (there called “surface” variables). How these variables are used in propagation is not described in any detail; we do not believe they are combined with TTEF propagation. Interestingly, no version of `cumulative` with energy variables appears in the Global Constraint Catalog, even though the CHIP developers are key contributors. However, Beldiceanu [3], one of the key contributors, describes a function called *ask_what_if* that can be passed to global constraint propagators and the propagator can query about bounds on, *e.g.*, a product of two variables. This could be used to imitate energy variables, but is not implemented in any system we are aware of. The most common used CP solvers (Gecode, Choco, JaCoP, SICStus Prolog, CP Optimizer, OR Tools) do not have an implementation, which considers the product/ energy variable. To best of our knowledge, there is no publication of filtering algorithms on the energy variable. Note that one of the authors implemented one in Objective CP.

References

1. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling **17**(7), 57–73 (1993)
2. Baptiste, P., Le Pape, C.: Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. Constraints **5**(1-2), 119–139 (2000)

3. Beldiceanu, N.: Global constraints as graph properties on structured network of elementary constraints of the same type. SICS Technical Report T2000/01, SICS, Uppsala, Sweden (2000)
4. Even, C., Schutt, A., Hentenryck, P.V.: A constraint programming approach for non-preemptive evacuation scheduling. In: Lecture Notes in Computer Science, pp. 574–591. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-23219-5_40
5. Garey, M.R., Johnson, D.R.: Computers and Intractability. W.H. Freeman and Co. (1979)
6. Korf, R.: Optimal rectangle packing: Initial results. In: Giunchiglia, E., Muscettola, N., Nau, D. (eds.) Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003). pp. 287–295. AAAI PRess (2003)
7. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
8. SA, C.: CHIP v5.12.2.0 finite domain constraints reference manual (2017)
9. Schutt, A., Feydy, T., Stuckey, P.J.: Explaining time-table-edge-finding propagation for the cumulative resource constraint. In: Proceedings of the 10th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming. pp. 234–250. No. 7874 in LNCS, Springer (2013)
10. Vilím, P.: Timetable edge finding filtering algorithm for discrete cumulative resources. In: Achterberg, T., Beck, J. (eds.) Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – CPAIOR 2011. Lecture Notes in Computer Science, vol. 6697, pp. 230–245. Springer Berlin / Heidelberg (2011). https://doi.org/10.1007/978-3-642-21311-3$_2$2