5.82 cumulative

DESCRIPTION LINKS GRAPH AUTOMATON

Origin [1]

Constraint `cumulative(TASKS, LIMIT)`

Synonym `cumulative_max`.

Arguments

$$\begin{array}{lll} \texttt{TASKS} & : & \texttt{collection}\left(\begin{array}{l}\texttt{origin}-\texttt{dvar},\\ \texttt{duration}-\texttt{dvar},\\ \texttt{end}-\texttt{dvar},\\ \texttt{height}-\texttt{dvar}\end{array}\right) \\ \texttt{LIMIT} & : & \texttt{int} \end{array}$$

Restrictions

$\texttt{require\_at\_least}(2, \texttt{TASKS}, [\texttt{origin}, \texttt{duration}, \texttt{end}])$
$\texttt{required}(\texttt{TASKS}, \texttt{height})$
$\texttt{TASKS.duration} \geq 0$
$\texttt{TASKS.origin} \leq \texttt{TASKS.end}$
$\texttt{TASKS.height} \geq 0$
$\texttt{LIMIT} \geq 0$

Purpose

Cumulative scheduling constraint or scheduling under resource constraints. Consider a set $\mathcal{T}$ of tasks described by the `TASKS` collection. The `cumulative` constraint enforces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is strictly greater than i. It also imposes for each task of $\mathcal{T}$ the constraint $\texttt{origin} + \texttt{duration} = \texttt{end}$.

Example

$$\left(\left\langle\begin{array}{llll} \texttt{origin}-1 & \texttt{duration}-3 & \texttt{end}-4 & \texttt{height}-1,\\ \texttt{origin}-2 & \texttt{duration}-9 & \texttt{end}-11 & \texttt{height}-2,\\ \texttt{origin}-3 & \texttt{duration}-10 & \texttt{end}-13 & \texttt{height}-1,\\ \texttt{origin}-6 & \texttt{duration}-6 & \texttt{end}-12 & \texttt{height}-1,\\ \texttt{origin}-7 & \texttt{duration}-2 & \texttt{end}-9 & \texttt{height}-3 \end{array}\right\rangle, 8\right)$$

Figure 5.165 shows the cumulated profile associated with the example. To each task of the `cumulative` constraint corresponds a set of rectangles coloured with the same colour: the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the resource consumption of the task. The `cumulative` constraint holds since at each point in time we do not have a cumulated resource consumption strictly greater than the upper limit 8 enforced by the last argument of the `cumulative` constraint.

Typical

$|\text{TASKS}| > 1$
range(TASKS.origin) > 1
range(TASKS.duration) > 1
range(TASKS.end) > 1
range(TASKS.height) > 1
TASKS.duration > 0
TASKS.height > 0
LIMIT $<$sum(TASKS.height)

Symmetries

- Items of TASKS are permutable.
- TASKS.height can be decreased to any value ≥ 0.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.
- LIMIT can be increased.

Remark

In the original cumulative constraint of **CHIP** the LIMIT parameter was a domain variable corresponding to the *maximum peak of the resource consumption profile*. Given a fixed time frame, this variable could be used as a cost in order to directly minimise the maximum resource consumption peak.

Some systems like Ilog CP Optimizer also assume that a zero-duration task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is greater than or equal to i. Under this definition, the height of a zero-duration task is also taken into account in the resource consumption profile.

Note that the concept of cumulative is *different* from the concept of rectangles non-overlapping even if, most of the time, each task of a ground solution of a cumulative constraint is simply drawn as a single rectangle. As illustrated by Figure 5.206, this is in fact not always possible (i.e., some rectangles may need to be broken apart). In fact the cumulative constraint is only a necessary condition for rectangles non-overlapping (see Figure 5.205 and the corresponding explanation in the **Algorithm** slot of the diffn constraint).

Algorithm

The first filtering algorithms were related to the notion of compulsory part of a task [223]. They compute a cumulated resource profile of all the compulsory parts of the tasks and

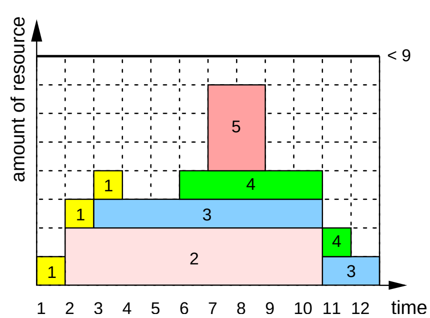


Figure 5.165: Resource consumption profile

prune the origins of the tasks with respect to this profile in order to not exceed the resource capacity. These methods are sometimes called *time tabling*. Even if these methods are quite local, i.e., a task has a non-empty compulsory part only when the difference between its latest start and its earliest start is strictly less than its duration, it scales well and is therefore widely used. Later on, more global algorithms[4] based on the resource consumption of the tasks on specific intervals were introduced [135, 94, 236]. A popular variant, called *edge finding*, considers only specific intervals [254]. An efficient implementation of edge finding in $O(kn \log n)$, where k is the number of distinct task heights and n is the number of tasks, based on a specific data structure, so called a *cumulative Φ-tree* [396], is provided in [395]. A $O(n^2 \log n)$ filtering algorithm based on tasks that can not be the earliest (or not be the latest) is described in [341].

Within the context of linear programming, the reference [191] provides a relaxation of the `cumulative` constraint.

A necessary condition for the `cumulative` constraint is obtained by stating a `disjunctive` constraint on a subset of tasks $\mathcal{T}$ such that, for each pair of tasks of $\mathcal{T}$, the sum of the two corresponding minimum heights is strictly greater than `LIMIT`. This can be done by applying the following procedure:

- Let h be the smallest minimum height strictly greater than $\lfloor \frac{\texttt{LIMIT}}{2} \rfloor$ of the tasks of the `cumulative` constraint. If no such task exists then the procedure is stopped without stating any `disjunctive` constraint.
- Let $\mathcal{T}_h$ denote the set of tasks of the `cumulative` constraint for which the minimum height is greater than or equal to h. By construction, the tasks of $\mathcal{T}_h$ cannot overlap. But we can eventually add one more task as shown by the next step.
- When it exists, we can add one task that does not belong to $\mathcal{T}_h$ and such that its minimum height is strictly greater than $\texttt{LIMIT} - h$. Again, by construction, this task cannot overlap all the tasks of $\mathcal{T}_h$.

When the tasks are involved in several `cumulative` constraints more sophisticated methods are available for extracting `disjunctive` constraints [16, 15].

In the context where, both the duration and height of all the tasks are fixed, [33] provides two kinds of additional filtering algorithms that are specially useful when the slack σ (i.e., the difference between the available space and the sum of the surfaces of the tasks) is very small:

- The first one introduces bounds for the so called *cumulative longest hole problem*. Given an integer ϵ that does not exceed the resource limit, and a subset of tasks $\mathcal{T}'$ for which the resource consumption is a most ϵ, the *cumulative longest hole problem* is to find the largest integer $lmax_\sigma^\epsilon(\mathcal{T}')$ such that there is a cumulative placement of maximum height ϵ involving a subset of tasks of $\mathcal{T}'$ where, on one interval $[i, i + lmax_\sigma^\epsilon(\mathcal{T}') - 1]$ of the cumulative profile, the area of the empty space does not exceed σ.
- The second one used dynamic programming for filtering so called *balancing knapsack constraints*. When the slack is 0, such constraints express the fact that the total height of tasks ending at instant i must equal the total height of tasks starting at instant i. Such constraints can be generalized to non-zero slack.

[4]Even if these more global algorithms usually can prune more early in the search tree, these algorithms do not catch all deductions derived from the cumulated resource profile of compulsory parts.

Systems

cumulativeMax in **Choco**, cumulative in **Gecode**, cumulative in **JaCoP**, cumulative in **SICStus**.

See also

assignment dimension added: coloured_cumulatives *(sum of* task heights *replaced by number of distinct* colours, *assignment dimension added)*, cumulatives *(negative* heights *allowed and assignment dimension added)*.

common keyword: calendar *(scheduling constraint)*, coloured_cumulative *(resource constraint, sum of* task heights *replaced by number of distinct values)*, coloured_cumulatives *(resource constraint)*, cumulative_convex *(resource constraint,* task *defined by a set of* points*)*, cumulative_product *(resource constraint, sum of* task heights *replaced by product of* task heights*)*, cumulative_with_level_of_priority *(resource constraint, a* cumulative *constraint for each set of* tasks *having a priority less than or equal to a given threshold)*.

generalisation: cumulative_two_d *(*task *replaced by* rectangle *with a* height*)*.

implied by: diffn *(*cumulative *is a neccessary condition for each dimension of the* diffn *constraint)*.

related: lex_chain_less, lex_chain_lesseq *(lexicographic ordering on the origins of* tasks, rectangles, *...)*, ordered_global_cardinality *(controlling the shape of the cumulative profile for breaking symmetry)*.

soft variant: soft_cumulative.

specialisation: atmost *(*task *replaced by* variable*)*, bin_packing *(all* tasks *have a* duration *of* 1 *and a fixed* height*)*, disjunctive *(all* tasks *have a* height *of* 1*)*.

used in graph description: sum_ctr.

Keywords

characteristic of a constraint: core, automaton, automaton with array of counters.

complexity: sequencing with release times and deadlines.

constraint type: scheduling constraint, resource constraint, temporal constraint.

filtering: linear programming, dynamic programming, compulsory part, cumulative longest hole problems, Phi-tree.

modelling: zero-duration task.

problems: producer-consumer.

puzzles: squared squares.

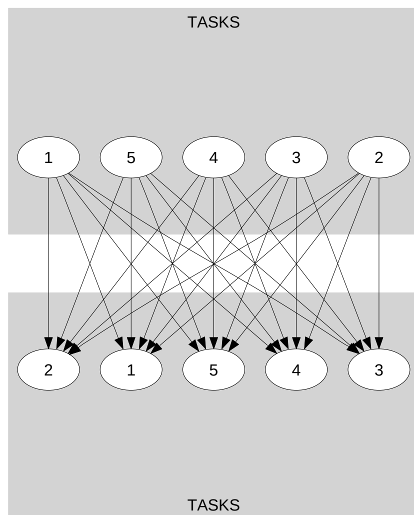
Arc input(s)	`TASKS`
Arc generator	$SELF \mapsto$ `collection(tasks)`
Arc arity	1
Arc constraint(s)	`tasks.origin + tasks.duration = tasks.end`
Graph property(ies)	**NARC** = |`TASKS`|

Arc input(s)	`TASKS TASKS`
Arc generator	$PRODUCT \mapsto$ `collection(tasks1, tasks2)`
Arc arity	2
Arc constraint(s)	• `tasks1.duration` > 0 • `tasks2.origin` $\leq$ `tasks1.origin` • `tasks1.origin` $<$ `tasks2.end`
Graph class	• `ACYCLIC` • `BIPARTITE` • `NO_LOOP`
Sets	$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables}-\texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ {[\texttt{item}(\texttt{var}-\texttt{TASKS.height})]}\end{array}\right)\end{array}\right]$
Constraint(s) on sets	`sum_ctr(variables,` $\leq$`, LIMIT)`

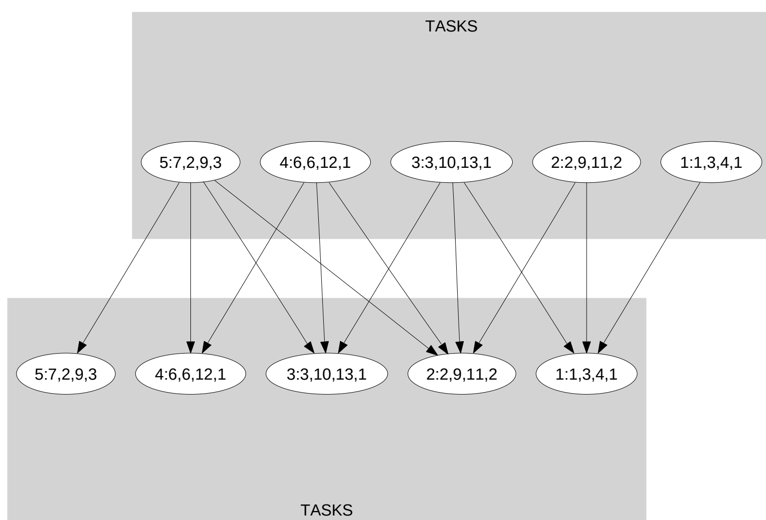
Graph model The first graph constraint enforces for each task the link between its origin, its duration and its end. The second graph constraint makes sure, for each time point t corresponding to the start of a task, that the cumulated heights of the tasks that overlap t does not exceed the limit of the resource.

Parts (A) and (B) of Figure 5.166 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks that overlap that time point. The `cumulative` constraint holds since for each successor set $\mathcal{S}$ of the final graph the sum of the heights of the tasks in $\mathcal{S}$ does not exceed the limit `LIMIT` $= 8$.

Signature Since `TASKS` is the maximum number of vertices of the final graph of the first graph constraint we can rewrite **NARC** $=$ |`TASKS`| to **NARC** $\geq$ |`TASKS`|. This leads to simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.



(A)



(B)

Figure 5.166: Initial and final graph of the `cumulative` constraint

Automaton Figure 5.167 depicts the automaton associated with the `cumulative` constraint. To each item of the collection `TASKS` corresponds a signature variable $\mathtt{S}_i$ that is equal to 1.

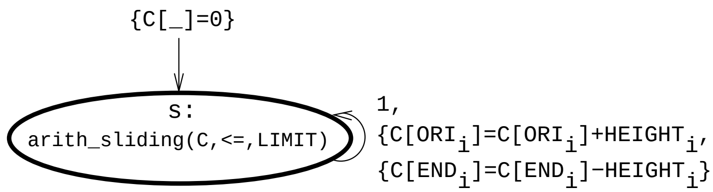


Figure 5.167: Automaton of the `cumulative` constraint