5.73 coloured_cumulative

DESCRIPTION LINKS GRAPH

Origin Derived from `cumulative` and `nvalues`.

Constraint `coloured_cumulative(TASKS, LIMIT)`

Synonym `colored_cumulative`.

Arguments

$$\texttt{TASKS} : \texttt{collection}\begin{pmatrix}\texttt{origin}-\texttt{dvar},\\ \texttt{duration}-\texttt{dvar},\\ \texttt{end}-\texttt{dvar},\\ \texttt{colour}-\texttt{dvar}\end{pmatrix}$$

$$\texttt{LIMIT} : \texttt{int}$$

Restrictions

`require_at_least`(2, `TASKS`, [`origin`, `duration`, `end`])
`required`(`TASKS`, `colour`)
`TASKS.duration` ≥ 0
`TASKS.origin` $\leq$ `TASKS.end`
`LIMIT` ≥ 0

Purpose

Consider the set $\mathcal{T}$ of tasks described by the `TASKS` collection. The `coloured_cumulative` constraint forces that, at each point in time, the number of distinct colours 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. For each task of $\mathcal{T}$ it also imposes the constraint `origin` + `duration` = `end`.

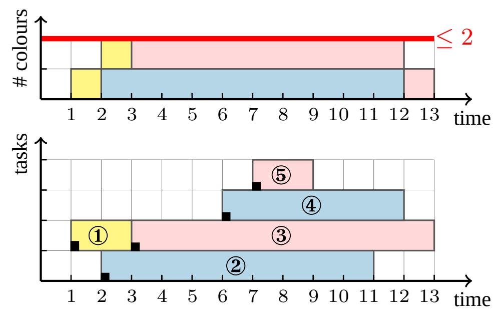
Example

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

Figure 5.193 shows the solution associated with the example. Each rectangle of the figure corresponds to a task of the `coloured_cumulative` constraint. Tasks that have their colour attribute set to 1, 2 and 3 are respectively coloured in yellow, blue and pink. The `coloured_cumulative` constraint holds since at each point in time we do not have more than `LIMIT` = 2 distinct colours.

Typical

|`TASKS`| > 1
`range`(`TASKS.origin`) > 1
`range`(`TASKS.duration`) > 1
`range`(`TASKS.end`) > 1
`range`(`TASKS.colour`) > 1
`LIMIT` < `nval`(`TASKS.colour`)



colour codes:

🟨 1 🟦 2 🟥 3

TASKS

①	o − 1	d − 2	e − 3	c − 1
②	o − 2	d − 9	e − 11	c − 2
③	o − 3	d − 10	e − 13	c − 3
④	o − 6	d − 6	e − 12	c − 2
⑤	o − 7	d − 2	e − 9	c − 3

(o for `origin`, d for `duration`, e for `end`, c for `colour`)

Figure 5.193: The coloured cumulative solution to the **Example** slot with at most two distinct colours in parallel

Symmetries

- Items of `TASKS` are permutable.
- One and the same constant can be added to the `origin` and `end` attributes of all items of `TASKS`.
- All occurrences of two distinct values of `TASKS.colour` can be swapped; all occurrences of a value of `TASKS.colour` can be renamed to any unused value.
- `LIMIT` can be increased.

Arg. properties

Contractible wrt. `TASKS`.

Usage

Useful for scheduling problems where a machine can only proceed in parallel a maximum number of tasks of distinct type. This condition cannot be modelled by the classical `cumulative` constraint. Also useful for coloured bin packing problems (i.e., `duration` $= 1$) where each item has a colour and no bin contains items with more than `LIMIT` distinct colours [132, 186, 206].

Reformulation

The `coloured_cumulative` constraint can be expressed in term of a set of reified constraints and of $|$`TASKS`$|$ `nvalue` constraints:

1. For each pair of tasks `TASKS`$[i]$, `TASKS`$[j]$ $(i, j \in [1, |$`TASKS`$|])$ of the `TASKS` collection we create a variable C_{ij} which is set to the colour of task `TASKS`$[j]$ if task `TASKS`$[j]$ overlaps the origin attribute of task `TASKS`$[i]$, and to the colour of task `TASKS`$[i]$ otherwise:
   - If $i = j$:
     - $C_{ij} = \texttt{TASKS}[i].\texttt{colour}$.
   - If $i \neq j$:
     - $C_{ij} = \texttt{TASKS}[i].\texttt{colour} \vee C_{ij} = \texttt{TASKS}[j].\texttt{colour}$.
     - $((\texttt{TASKS}[j].\texttt{origin} \leq \texttt{TASKS}[i].\texttt{origin} \wedge$
       $\texttt{TASKS}[j].\texttt{end} > \texttt{TASKS}[i].\texttt{origin}) \wedge (C_{ij} = \texttt{TASKS}[j].\texttt{colour})) \vee$
       $((\texttt{TASKS}[j].\texttt{origin} > \texttt{TASKS}[i].\texttt{origin} \vee$
       $\texttt{TASKS}[j].\texttt{end} \leq \texttt{TASKS}[i].\texttt{origin}) \wedge (C_{ij} = \texttt{TASKS}[i].\texttt{colour}))$

2. For each task TASKS$[i]$ $(i \in [1, |$TASKS$|])$ we create a variable N_i which gives the number of distinct colours associated with the tasks that overlap the origin of task TASKS$[i]$ (TASKS$[i]$ overlaps its own origin) and we impose N_i to not exceed the maximum number of distinct colours LIMIT allowed at each instant:
   - $N_i \geq 1 \wedge N_i \leq$ LIMIT.
   - nvalue$(N_i, \langle C_{i1}, C_{i2}, \ldots, C_{i|\texttt{TASKS}|}\rangle)$.

See also

assignment dimension added: coloured_cumulatives.

common keyword: cumulative, track *(resource constraint)*.

related: nvalue.

specialisation: disjoint_tasks *(a colour is assigned to each collection of tasks of constraint* disjoint_tasks *and a limit of one single colour is enforced)*.

used in graph description: nvalues.

Keywords

characteristic of a constraint: coloured.

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

filtering: compulsory part.

modelling: number of distinct values, zero-duration task.

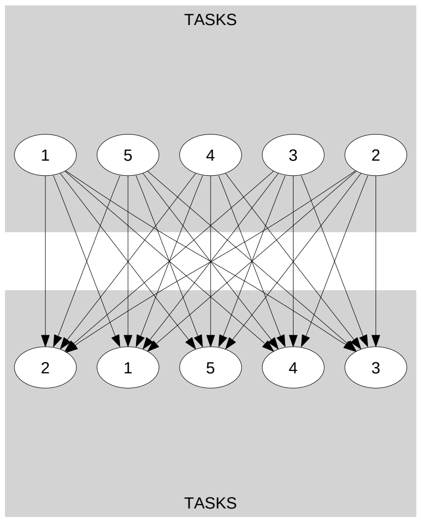
Arc input(s)	`TASKS`
Arc generator	$SELF \mapsto$ `collection(tasks)`
Arc arity	1
Arc constraint(s)	`tasks.origin` + `tasks.duration` = `tasks.end`
Graph property(ies)	**NARC**$=$ |`TASKS`|

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	$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{TASKS.colour})]\end{array}\right)\end{array}\right]$
Constraint(s) on sets	`nvalues(variables,` $\leq$`, LIMIT)`

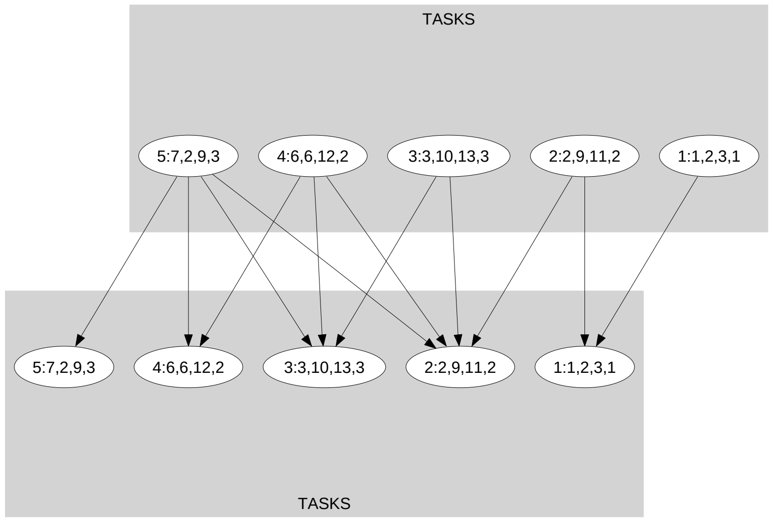
Graph model Same as `cumulative`, except that we use another constraint for computing the resource consumption at each time point.

Parts (A) and (B) of Figure 5.194 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 `coloured_cumulative` constraint holds since for each successor set $\mathcal{S}$ of the final graph the number of distinct colours of the tasks in $\mathcal{S}$ does not exceed the `LIMIT` 2.

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



(A)



(B)

Figure 5.194: Initial and final graph of the `coloured_cumulative` constraint