

5.168 interval_and_count

	DESCRIPTION	LINKS	GRAPH	AUTOMATON
Origin	[114]			
Constraint	interval_and_count(ATMOST, COLOURS, TASKS, SIZE_INTERVAL)			
Arguments	ATMOST : int COLOURS : collection(val-int) TASKS : collection(origin-dvar, colour-dvar) SIZE_INTERVAL : int			
Restrictions	ATMOST ≥ 0 required(COLOURS, val) distinct(COLOURS, val) required(TASKS, [origin, colour]) SIZE_INTERVAL > 0			

Purpose First consider the set of tasks of the TASKS collection, where each task has a specific colour that may not be initially fixed. Then consider the intervals of the form $[k \cdot \text{SIZE_INTERVAL}, k \cdot \text{SIZE_INTERVAL} + \text{SIZE_INTERVAL} - 1]$, where k is an integer. The interval_and_count constraint enforces that, for each interval I_k previously defined, the total number of tasks, which both are assigned to I_k and take their colour in COLOURS, does not exceed the limit ATMOST.

Example
$$\left(\begin{array}{l} 2, \langle 4 \rangle, \\ \left\langle \begin{array}{ll} \text{origin} - 1 & \text{colour} - 4, \\ \text{origin} - 0 & \text{colour} - 9, \\ \text{origin} - 10 & \text{colour} - 4, \\ \text{origin} - 4 & \text{colour} - 4 \end{array} \right\rangle, 5 \end{array} \right)$$

Figure 5.339 shows the solution associated with the example. The constraint interval_and_count holds since, for each interval, the number of tasks taking colour 4 does not exceed the limit 2.

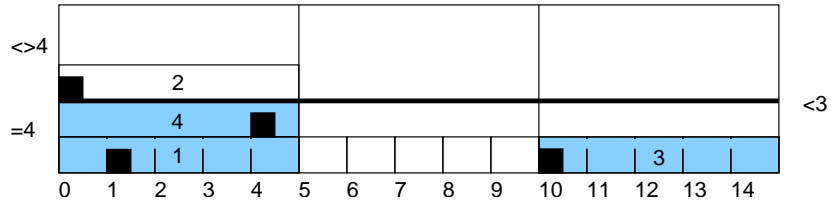


Figure 5.339: Solution with the use of each interval

Typical

```

ATMOST > 0
|COLOURS| > 0
|TASKS| > 1
range(TASKS.origin) > 1
range(TASKS.colour) > 1
SIZE_INTERVAL > 1

```

Symmetries

- ATMOST can be **increased**.
- Items of COLOURS are **permutable**.
- Items of TASKS are **permutable**.
- One and the same constant can be **added** to the `origin` attribute of all items of TASKS.
- An occurrence of a value of `TASKS.origin` that belongs to the k -th interval, of size `SIZE_INTERVAL`, can be **replaced** by any other value of the same interval.
- An occurrence of a value of `TASKS.colour` that belongs to `COLOURS.val` (resp. does not belong to `COLOURS.val`) can be **replaced** by any other value in `COLOURS.val` (resp. not in `COLOURS.val`).

Usage

This constraint was originally proposed for dealing with timetabling problems. In this context the different intervals are interpreted as morning and afternoon periods of different consecutive days. Each colour corresponds to a type of course (i.e., French, mathematics). There is a restriction on the maximum number of courses of a given type each morning as well as each afternoon.

Remark

If we want to only consider intervals that correspond to the morning or to the afternoon we could extend the `interval_and_count` constraint in the following way:

- We introduce two extra parameters `REST` and `QUOTIENT` that correspond to non-negative integers such that `REST` is strictly less than `QUOTIENT`,
- We add the following condition to the arc constraint:
 $(\text{tasks1.origin}/\text{SIZE_INTERVAL}) \equiv \text{REST} \pmod{\text{QUOTIENT}}$

Now, if we want to express a constraint on the morning intervals, we set `REST` to 0 and `QUOTIENT` to 2.

Reformulation

Let K denote the index of the last possible interval where the tasks can be assigned: $K = \lfloor \frac{\max_{i \in [1, |\text{TASKS}|]} (\text{TASKS}[i].\text{origin}) + \text{SIZE_INTERVAL} - 1}{\text{SIZE_INTERVAL}} \rfloor$. The `interval_and_count(ATMOST, COLOURS, TASKS, SIZE_INTERVAL)` constraint can be expressed in term of a set of reified constraints and of K arithmetic constraints (i.e., `sum_ctr` constraints).

1. For each task `TASKS[i]` ($i \in [1, |\text{TASKS}|]$) of the `TASKS` collection we create a 0-1 variable B_i that will be set to 1 if and only if task `TASKS[i]` takes a colour within the set of colours `COLOURS`:

$$B_i \Leftrightarrow \text{TASKS}[i].\text{colour} = \text{COLOURS}[1].\text{val} \vee$$

$$\text{TASKS}[i].\text{colour} = \text{COLOURS}[2].\text{val} \vee$$

$$\dots$$

$$\text{TASKS}[i].\text{colour} = \text{COLOURS}[|\text{COLOURS}|].\text{val}.$$

2. For each task $\text{TASKS}[i]$ ($i \in [1, |\text{TASKS}|]$) and for each interval $[k \cdot \text{SIZE_INTERVAL}, k \cdot \text{SIZE_INTERVAL} + \text{SIZE_INTERVAL} - 1]$ ($k \in [0, K]$) we create a 0-1 variable B_{ik} that will be set to 1 if and only if, both task $\text{TASKS}[i]$ takes a colour within the set of colours COLOURS, and the origin of task $\text{TASKS}[i]$ is assigned within interval $[k \cdot \text{SIZE_INTERVAL}, k \cdot \text{SIZE_INTERVAL} + \text{SIZE_INTERVAL} - 1]$:

$$B_{ik} \Leftrightarrow B_i \wedge$$

$$\text{TASKS}[i].\text{origin} \geq k \cdot \text{SIZE_INTERVAL} \wedge$$

$$\text{TASKS}[i].\text{origin} \leq k \cdot \text{SIZE_INTERVAL} + \text{SIZE_INTERVAL} - 1$$
3. Finally, for each interval $[k \cdot \text{SIZE_INTERVAL}, k \cdot \text{SIZE_INTERVAL} + \text{SIZE_INTERVAL} - 1]$ ($k \in [0, K]$), we impose the sum $B_{1k} + B_{2k} + \dots + B_{|\text{TASKS}|k}$ to not exceed the maximum allowed capacity ATMOST.

See also [assignment dimension removed](#): `among_low_up` (*assignment dimension corresponding to intervals is removed*).

related: `interval_and_sum` (`among_low_up` constraint replaced by `sum_ctr`).

used in graph description: `among_low_up`.

Keywords [application area](#): assignment.

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

constraint type: timetabling constraint, resource constraint, temporal constraint.

modelling: assignment dimension, interval.

Arc input(s)	TASKS TASKS
Arc generator	<code>PRODUCT</code> \mapsto <code>collection</code> (tasks1, tasks2)
Arc arity	2
Arc constraint(s)	tasks1.origin/SIZE_INTERVAL = tasks2.origin/SIZE_INTERVAL
Sets	$SUCC \mapsto$ $\left[\begin{array}{l} \text{source,} \\ \text{variables} - \text{col} \left(\begin{array}{l} \text{VARIABLES} - \text{collection}(\text{var} - \text{dvar}), \\ [\text{item}(\text{var} - \text{TASKS.colour})] \end{array} \right) \end{array} \right]$
Constraint(s) on sets	<code>among_low_up</code> (0, ATMOST, variables, COLOURS)

Graph model

We use a bipartite graph where each class of vertices corresponds to the different tasks of the TASKS collection. There is an arc between two tasks if their origins belong to the same interval. Finally we enforce an `among_low_up` constraint on each set S of successors of the different vertices of the final graph. This put a restriction on the maximum number of tasks of S for which the colour attribute takes its value in COLOURS.

Parts (A) and (B) of Figure 5.340 respectively show the initial and final graph associated with the **Example** slot. Each connected component of the final graph corresponds to items that are all assigned to the same interval.

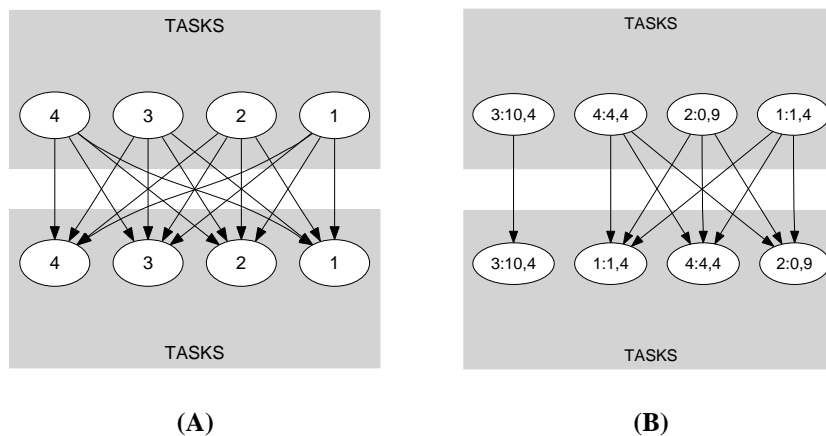


Figure 5.340: Initial and final graph of the interval_and_count constraint

Automaton

Figure 5.341 depicts the automaton associated with the `interval_and_count` constraint. Let COLOUR_i be the `colour` attribute of the i^{th} item of the `TASKS` collection. To each pair $(\text{COLOURS}, \text{COLOUR}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $\text{COLOUR}_i \in \text{COLOURS} \Leftrightarrow S_i$.

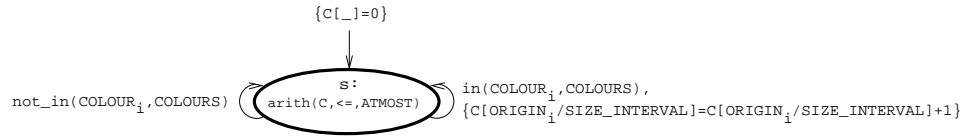


Figure 5.341: Automaton of the `interval_and_count` constraint

20000128

1083