5.352 sliding_time_window

Origin   N. Beldiceanu

Constraint   sliding_time_window(WINDOW_SIZE, LIMIT, TASKS)

Arguments
```
WINDOW_SIZE  :  int
LIMIT        :  int
TASKS        :  collection(origin−dvar, duration−dvar)
```

Restrictions
WINDOW_SIZE > 0
LIMIT ≥ 0
required(TASKS, [origin, duration])
TASKS.duration ≥ 0

Purpose   For any time window of size WINDOW_SIZE, the intersection of all the tasks of the collection TASKS with this time window is less than or equal to a given limit LIMIT.

Example
$$\left(9, 6, \left\langle \begin{array}{ll} \text{origin} - 10 & \text{duration} - 3, \\ \text{origin} - 5 & \text{duration} - 1, \\ \text{origin} - 6 & \text{duration} - 2, \\ \text{origin} - 14 & \text{duration} - 2, \\ \text{origin} - 2 & \text{duration} - 2 \end{array} \right\rangle \right)$$

The lower part of Figure 5.688 indicates the different tasks on the time axis. Each task is drawn as a rectangle with its corresponding identifier in the middle. Finally the upper part of Figure 5.688 shows the different time windows and the respective contribution of the tasks in these time windows. Note that we only need to focus on those time windows starting at the start of one of the tasks. A line with two arrows depicts each time window. The two arrows indicate the start and the end of the time window. At the left of each time window we give its occupation. Since this occupation is always less than or equal to the limit 6, the sliding_time_window constraint holds.

Typical
WINDOW_SIZE > 1
LIMIT > 0
LIMIT < sum(TASKS.duration)
|TASKS| > 1
TASKS.duration > 0

Symmetries
- WINDOW_SIZE can be decreased.
- LIMIT can be increased.
- Items of TASKS are permutable.
- One and the same constant can be added to the origin attribute of all items of TASKS.
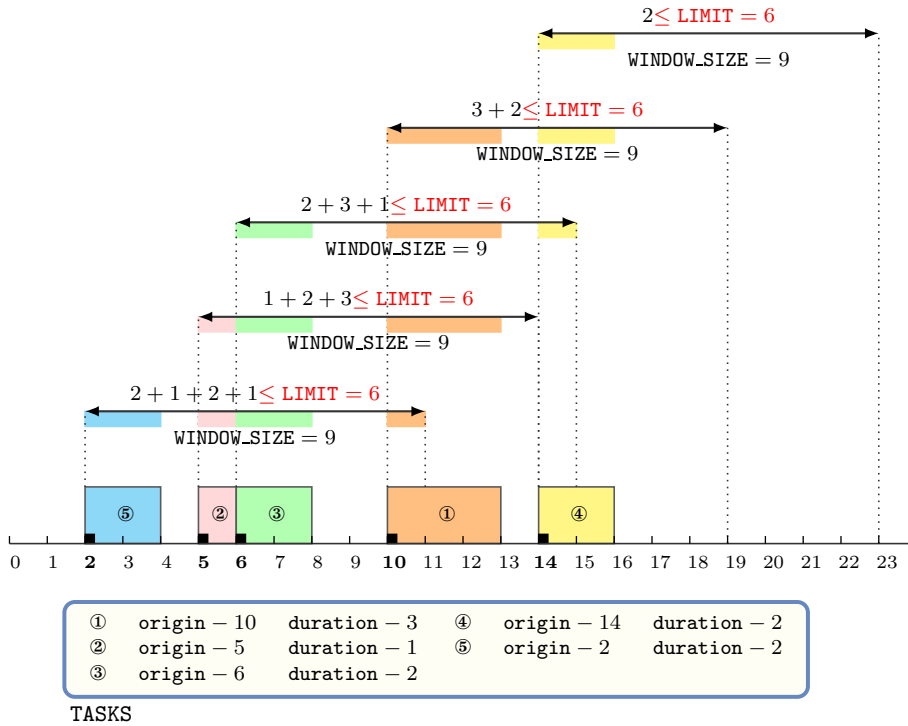- TASKS.duration can be decreased to any value ≥ 0.

Figure 5.688: Time windows and their use for the five tasks of the **Example** slot

Arg. properties

Contractible wrt. TASKS.

Usage

The sliding_time_window constraint is useful for timetabling problems in order to put an upper limit on the total work over sliding time windows.

Reformulation

The sliding_time_window constraint can be expressed in term of a set of $|\text{TASKS}|^2$ reified constraints and of $|\text{TASKS}|$ linear inequalities constraints:

1. For each pair of tasks $\text{TASKS}[i]$, $\text{TASKS}[j]$ $(i, j \in [1, |\text{TASKS}|])$ of the TASKS collection we create a variable $Inter_{ij}$ which is set to the intersection of $\text{TASKS}[j]$ with the time window $\mathcal{W}_i$ of size WINDOW_SIZE that starts at instant $\text{TASKS}[i].\text{origin}$:

   - If $i = j$ (i.e., $\text{TASKS}[i]$ and $\text{TASKS}[j]$ coincide):
     - $Inter_{ij} = \min(\text{TASKS}[i].\text{duration}, \text{WINDOW\_SIZE})$.
   - If $i \neq j$ and $\overline{\text{TASKS}[j].\text{origin}} + \overline{\text{TASKS}[j].\text{duration}} < \underline{\text{TASKS}[i].\text{origin}}$ (i.e., $\text{TASKS}[j]$ for sure ends before the time window $\mathcal{W}_i$):
     - $Inter_{ij} = 0$.
   - If $i \neq j$ and $\underline{\text{TASKS}[j].\text{origin}} > \overline{\text{TASKS}[i].\text{origin}} + \text{WINDOW\_SIZE} - 1$ (i.e., $\text{TASKS}[j]$ for sure starts after the time window $\mathcal{W}_i$):
     - $Inter_{ij} = 0$.
   - Otherwise (i.e., $\text{TASKS}[j]$ can potentially overlap the time window $\mathcal{W}_i$):

– $Inter_{ij}$ $=$ $\max(0, \min(\mathtt{TASKS}[i].\mathtt{origin}$ $+$ $\mathtt{WINDOW\_SIZE}, \mathtt{TASKS}[j].\mathtt{origin}$ $+$ $\mathtt{TASKS}[j].\mathtt{duration})$ $-$ $\max(\mathtt{TASKS}[i].\mathtt{origin}, \mathtt{TASKS}[j].\mathtt{origin}))$.

2. For each task $\mathtt{TASKS}[i]$ ($i \in [1, |\mathtt{TASKS}|]$) we create a linear inequality constraint $Inter_{i1} + Inter_{i2} + \cdots + Inter_{i|\mathtt{TASKS}|} \leq \mathtt{LIMIT}$.

See also   **common keyword:** shift *(temporal constraint)*.

related: sliding_time_window_sum *(sum of intersections of tasks with sliding time window replaced by sum of the points of intersecting tasks with sliding time window)*.

used in graph description: sliding_time_window_from_start.

Keywords   **constraint type:** sliding sequence constraint, temporal constraint.

Arc input(s)	TASKS
Arc generator	$CLIQUE \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• tasks1.origin $\leq$ tasks2.origin • tasks2.origin $-$ tasks1.origin $<$ WINDOW_SIZE
Sets	SUCC $\mapsto$ [source, tasks]
Constraint(s) on sets	sliding_time_window_from_start $\begin{pmatrix} \text{WINDOW\_SIZE,} \\ \text{LIMIT,} \\ \text{tasks,} \\ \text{source.origin} \end{pmatrix}$

Graph model

We generate an arc from a task t_1 to a task t_2 if task t_2 does not start before task t_1 and if task t_2 intersects the time window that starts at the origin of task t_1. Each set generated by SUCC corresponds to all tasks that intersect in time the time window that starts at the origin of a given task.

Parts (A) and (B) of Figure 5.689 respectively show the initial and final graph associated with the **Example** slot. In the final graph, the successors of a given task t correspond to the set of tasks that do not start before task t and intersect the time window that starts at the origin of task t.
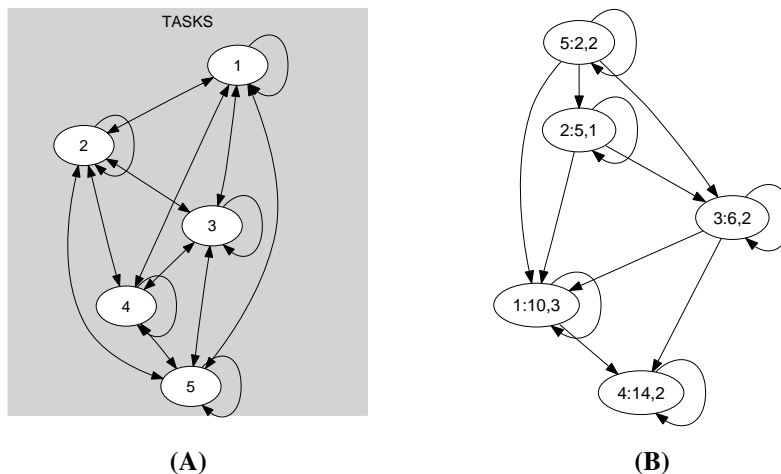

Figure 5.689: Initial and final graph of the sliding_time_window constraint