5.354 sliding_time_window_sum

DESCRIPTION LINKS GRAPH

Origin

Derived from `sliding_time_window`.

Constraint

`sliding_time_window_sum(WINDOW_SIZE, LIMIT, TASKS)`

Arguments

WINDOW_SIZE	:	int
LIMIT	:	int
TASKS	:	collection(origin−dvar, end−dvar, npoint−dvar)

Restrictions

WINDOW_SIZE > 0
LIMIT ≥ 0
required(TASKS, [origin, end, npoint])
TASKS.origin $\leq$ TASKS.end
TASKS.npoint ≥ 0

Purpose

For any time window of size WINDOW_SIZE, the sum of the points of the tasks of the collection TASKS that overlap that time window do not exceed a given limit LIMIT.

Example

$$\left(9, 16, \left\langle\begin{array}{lll} \texttt{origin}-10 & \texttt{end}-13 & \texttt{npoint}-2, \\ \texttt{origin}-5 & \texttt{end}-6 & \texttt{npoint}-3, \\ \texttt{origin}-6 & \texttt{end}-8 & \texttt{npoint}-4, \\ \texttt{origin}-14 & \texttt{end}-16 & \texttt{npoint}-5, \\ \texttt{origin}-2 & \texttt{end}-4 & \texttt{npoint}-6 \end{array}\right\rangle\right)$$

The lower part of Figure 5.691 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 5.691 shows the different time windows and the respective contribution of the tasks in these time windows. A line with two arrows depicts each time window. The two arrows indicate the start and the end of the time window. At the right of each time window we give its occupation. Since this occupation is always less than or equal to the limit 16, the `sliding_time_window_sum` constraint holds.

Typical

WINDOW_SIZE > 1
LIMIT > 0
LIMIT $<$sum(TASKS.npoint)
$|$TASKS$| > 1$
TASKS.origin $<$ TASKS.end
TASKS.npoint > 0

Symmetries

- WINDOW_SIZE can be decreased.
- LIMIT can be increased.
- Items of TASKS are permutable.
- TASKS.npoint 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.

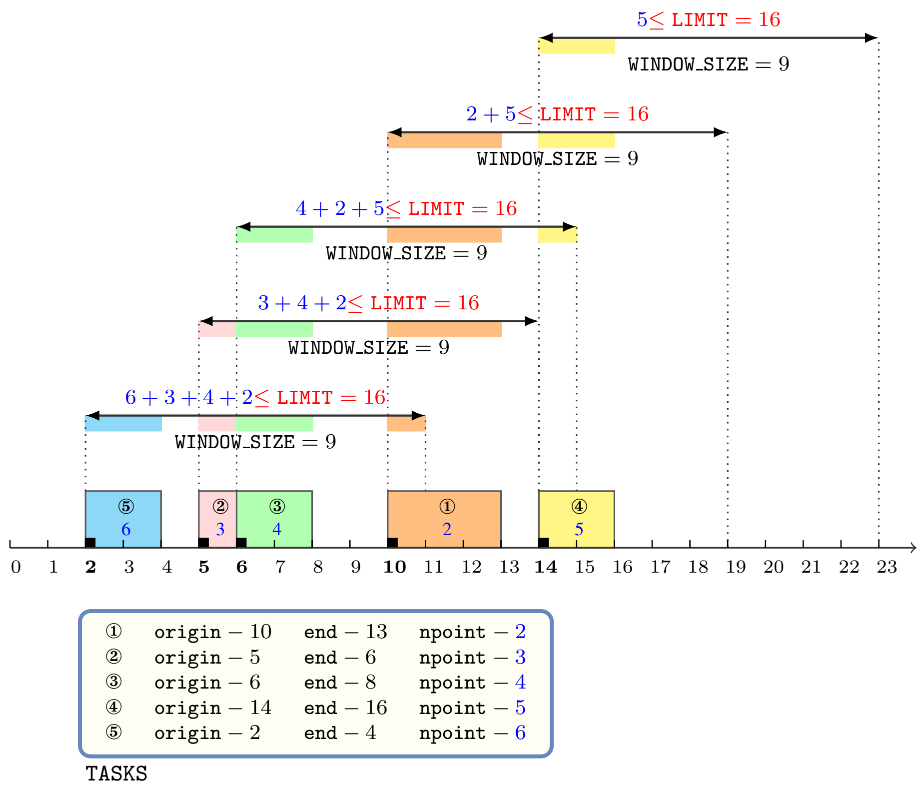


Figure 5.691: Time windows and their use for the five tasks of the **Example** slot

Arg. properties

Contractible wrt. TASKS.

Usage

This constraint may be used for timetabling problems in order to put an upper limit on the cumulated number of points in a shift.

Reformulation

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

1. For each pair of tasks $\mathtt{TASKS}[i], \mathtt{TASKS}[j]$ $(i, j \in [1, |\mathtt{TASKS}|])$ of the TASKS collection we create a variable $Point_{ij}$ which is set to $\mathtt{TASKS}[j].\mathtt{npoint}$ if $\mathtt{TASKS}[j]$ intersects the time window $\mathcal{W}_i$ of size WINDOW_SIZE that starts at instant $\mathtt{TASKS}[i].\mathtt{origin}$, or 0 otherwise:
   - If $i = j$ (i.e., $\mathtt{TASKS}[i]$ and $\mathtt{TASKS}[j]$ coincide):
     - $Point_{ij} = \mathtt{TASKS}[i].\mathtt{npoint}$.
   - If $i \neq j$ and $\overline{\mathtt{TASKS}[j].\mathtt{end}} < \underline{\mathtt{TASKS}[i].\mathtt{origin}}$ (i.e., $\mathtt{TASKS}[j]$ for sure ends before the time window $\mathcal{W}_i$):
     - $Point_{ij} = 0$.
   - If $i \neq j$ and $\underline{\mathtt{TASKS}[j].\mathtt{origin}} > \overline{\mathtt{TASKS}[i].\mathtt{origin}} + \mathtt{WINDOW\_SIZE} - 1$ (i.e., $\mathtt{TASKS}[j]$ for sure starts after the time window $\mathcal{W}_i$):

    - $Point_{ij} = 0$.
  - Otherwise (i.e., $\texttt{TASKS}[j]$ can potentially overlap the time window $\mathcal{W}_i$):
    - $Point_{ij} = \min(1, \max(0, \min(\texttt{TASKS}[i].\texttt{origin} + \texttt{WINDOW\_SIZE}, \texttt{TASKS}[j].\texttt{end}) - \max(\texttt{TASKS}[i].\texttt{origin}, \texttt{TASKS}[j].\texttt{origin}))) \cdot \texttt{TASKS}[j].\texttt{npoint}$.

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

See also

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

used in graph description: `sum_ctr`.

Keywords

characteristic of a constraint: time window, sum.

constraint type: sliding sequence constraint, temporal constraint.

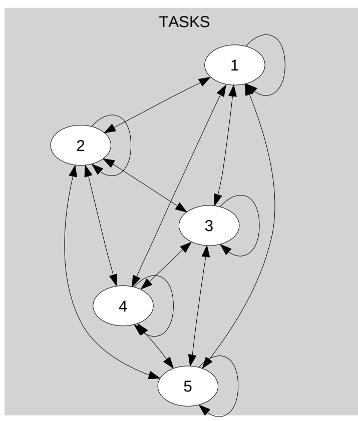
Arc input(s)	TASKS
Arc generator	$SELF \mapsto$ collection(tasks)
Arc arity	1
Arc constraint(s)	tasks.origin $\leq$ tasks.end
Graph property(ies)	**NARC**$=$ |TASKS|

Arc input(s)	TASKS
Arc generator	$CLIQUE \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• tasks1.end $\leq$ tasks2.end • tasks2.origin $-$ tasks1.end $<$ WINDOW_SIZE $-$ 1
Sets	SUCC $\mapsto$ [source, variables $-$ col (VARIABLES$-$collection(var$-$dvar), [item(var $-$ TASKS.npoint)])]
Constraint(s) on sets	sum_ctr(variables, $\leq$, LIMIT)

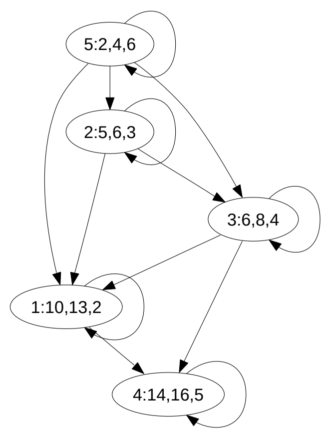
Graph model We generate an arc from a task t_1 to a task t_2 if task t_2 does not end before the end of task t_1 and if task t_2 intersects the time window that starts at the last instant of task t_1. Each set generated by SUCC corresponds to all tasks that intersect in time the time window that starts at instant end $-$ 1, where end is the end of a given task.

Parts (A) and (B) of Figure 5.692 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 both do not end before the end of task t, and intersect the time window that starts at the end $-$ 1 of task t.

Signature Consider the first graph constraint. Since we use the $SELF$ arc generator on the TASKS collection the maximum number of arcs of the final graph is equal to |TASKS|. Therefore we can rewrite $\mathbf{NARC} = |\mathtt{TASKS}|$ to $\mathbf{NARC} \geq |\mathtt{TASKS}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.



(A)



(B)

Figure 5.692: Initial and final graph of the `sliding_time_window_sum` constraint