5.351 sliding_sum

DESCRIPTION LINKS GRAPH

Origin CHIP

Constraint sliding_sum(LOW, UP, SEQ, VARIABLES)

Synonym sequence.

Arguments

LOW	:	int
UP	:	int
SEQ	:	int
VARIABLES	:	collection(var−dvar)

Restrictions

$\texttt{UP} \geq \texttt{LOW}$
$\texttt{SEQ} > 0$
$\texttt{SEQ} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)

Purpose

Constrains all sequences of SEQ consecutive variables of the collection VARIABLES so that the sum of the variables belongs to interval [LOW, UP].

Example

$(3, 7, 4, \langle 1, 4, 2, 0, 0, 3, 4\rangle)$

The example considers all sliding sequences of SEQ $=$ 4 consecutive values of $\langle 1, 4, 2, 0, 0, 3, 4\rangle$ collection and constraints the sum to be in $[\texttt{LOW}, \texttt{UP}] = [3, 7]$. The sliding_sum constraint holds since the sum associated with the corresponding subsequences 1 4 2 0, 4 2 0 0, 2 0 0 3, and 0 0 3 4 are respectively 7, 6, 5 and 7.

Typical

$\texttt{LOW} \geq 0$
$\texttt{UP} > 0$
$\texttt{SEQ} > 1$
$\texttt{SEQ} < |\texttt{VARIABLES}|$
$\texttt{VARIABLES.var} \geq 0$
$\texttt{UP} <$ sum(VARIABLES.var)

Symmetry Items of VARIABLES can be reversed.

Arg. properties

- Contractible wrt. VARIABLES when $\texttt{SEQ} = 1$.
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Algorithm Beldiceanu and Carlsson [30] have proposed a first incomplete filtering algorithm for the sliding_sum constraint. In 2008, Maher *et al.* showed in [273] that the sliding_sum constraint has a solution "*if and only there are no negative cycles in the flow graph associated with the dual linear program*" that encodes the conjunction of inequalities. They derive a bound-consistency filtering algorithm from this fact.

Systems sliding_sum in **MiniZinc**.

See also **common keyword:** `sliding_distribution` *(sliding sequence constraint)*.

part of system of constraints: `sum_ctr`.

soft variant: `relaxed_sliding_sum`.

used in graph description: `sum_ctr`.

Keywords **characteristic of a constraint:** hypergraph, sum.

combinatorial object: sequence.

constraint type: decomposition, sliding sequence constraint, system of constraints.

filtering: linear programming, flow, bound-consistency.

Arc input(s)	`VARIABLES`
Arc generator	$PATH \mapsto$ `collection`
Arc arity	`SEQ`
Arc constraint(s)	• `sum_ctr`(`collection`, $\geq$, `LOW`) • `sum_ctr`(`collection`, $\leq$, `UP`)
Graph property(ies)	$\mathbf{NARC} = |$`VARIABLES`$| -$ `SEQ` $+ 1$

Graph model We use `sum_ctr` as an arc constraint. `sum_ctr` takes a collection of domain variables as its first argument.

Parts (A) and (B) of Figure 5.687 respectively show the initial and final graph associated with the **Example** slot. Since all arc constraints hold (i.e., because of the graph property $\mathbf{NARC} = |$`VARIABLES`$| -$ `SEQ` $+ 1$) the final graph corresponds to the initial graph.

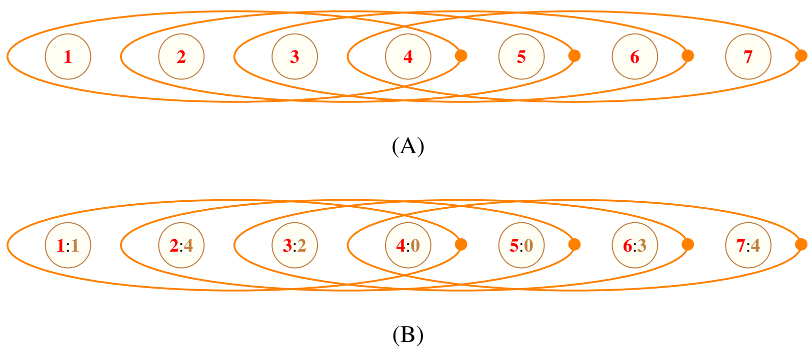

Figure 5.687: (A) Initial and (B) final graph of the `sliding_sum`$(3, 7, 4, \langle 1, 4, 2, 0, 0, 3, 4\rangle)$ constraint of the **Example** slot where each ellipse represents an hyperedge involving `SEQ` $= 4$ vertices (e.g., the first ellipse represents the constraint $1 + 4 + 2 + 0 \in [3, 7]$)

Signature Since we use the *PATH* arc generator with an arity of `SEQ` on the items of the `VARIABLES` collection, the expression $|$`VARIABLES`$| -$ `SEQ` $+ 1$ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\mathbf{NARC} = |$`VARIABLES`$| -$ `SEQ` $+ 1$ to $\mathbf{NARC} \geq |$`VARIABLES`$| -$ `SEQ` $+ 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.