## 5.354 sliding_time_window_from_start

DESCRIPTION LINKS GRAPH

**Origin** Used for defining sliding_time_window.

**Constraint** sliding_time_window_from_start(WINDOW_SIZE, LIMIT, TASKS, START)

**Arguments**

| | | |
|---|---|---|
| WINDOW_SIZE | : | int |
| LIMIT | : | int |
| TASKS | : | collection(origin−dvar, duration−dvar) |
| START | : | dvar |

**Restrictions**

$\texttt{WINDOW\_SIZE} > 0$
$\texttt{LIMIT} \geq 0$
required(TASKS, [origin, duration])
$\texttt{TASKS.duration} \geq 0$

**Purpose** The sum of the intersections of all the tasks of the TASKS collection with interval $[\texttt{START}, \texttt{START} + \texttt{WINDOW\_SIZE} - 1]$ is less than or equal to LIMIT.

**Example**

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

The intersections of tasks $\langle \texttt{id} - 1\ \texttt{origin} - 10\ \texttt{duration} - 3\rangle$, $\langle \texttt{id} - 2\ \texttt{origin} - 5\ \texttt{duration} - 1\rangle$, and $\langle \texttt{id} - 3\ \texttt{origin} - 6\ \texttt{duration} - 2\rangle$ with interval $[\texttt{START}, \texttt{START} + \texttt{WINDOW\_SIZE} - 1] = [5, 5 + 9 - 1] = [5, 13]$ are respectively equal to 3, 1, and 2 (i.e., the three tasks of the TASKS collection are in fact included within interval $[5, 13]$). Consequently, the sliding_time_window_from_start constraint holds since the sum $3 + 1 + 2$ of these intersections does not exceed the value of its second argument $\texttt{LIMIT} = 6$.

**Typical**

$\texttt{WINDOW\_SIZE} > 1$
$\texttt{LIMIT} > 0$
$\texttt{LIMIT} < \texttt{WINDOW\_SIZE}$
$|\texttt{TASKS}| > 1$
$\texttt{TASKS.duration} > 0$

**Symmetries**

- WINDOW_SIZE can be decreased.
- LIMIT can be increased.
- Items of TASKS are permutable.
- TASKS.duration can be decreased to any value $\geq 0$.
- One and the same constant can be added to START as well as to the origin attribute of all items of TASKS.

**Arg. properties** Contractible wrt. TASKS.

**Reformulation** Similar to the reformulation of `sliding_time_window`.

**Used in** `sliding_time_window`.

**Keywords** **characteristic of a constraint:** derived collection.

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

**Derived Collection** $\texttt{col}(\texttt{S}-\texttt{collection}(\texttt{var}-\texttt{dvar}),[\texttt{item}(\texttt{var}-\texttt{START})])$

**Arc input(s)** S TASKS

**Arc generator** $PRODUCT\mapsto\texttt{collection}(\texttt{s},\texttt{tasks})$

**Arc arity** 2

**Arc constraint(s)** TRUE

**Graph property(ies)**

$$\mathbf{SUM\_WEIGHT\_ARC}\left(\texttt{max}\left(0,\begin{array}{l}\texttt{min}\left(\begin{array}{l}\texttt{s.var}+\texttt{WINDOW\_SIZE},\\ \texttt{tasks.origin}+\texttt{tasks.duration}\end{array}\right)-\\ \texttt{max}(\texttt{s.var},\texttt{tasks.origin})\end{array}\right)\right)\leq\texttt{LIMIT}$$

**Graph model** Since we use the TRUE arc constraint the final and the initial graph are identical. The unique source of the final graph corresponds to the interval $[\texttt{START},\texttt{START}+\texttt{WINDOW\_SIZE}-1]$. Each sink of the final graph represents a given task of the TASKS collection. We associate to each arc the value given by the intersection of the task associated with one of the extremities of the arc with the time window $[\texttt{START},\texttt{START}+\texttt{WINDOW\_SIZE}-1]$. Finally, the graph property **SUM_WEIGHT_ARC** sums up all the valuations of the arcs and check that it does not exceed a given limit.

Parts (A) and (B) of Figure 5.709 respectively show the initial and final graph associated with the **Example** slot. To each arc of the final graph we associate the intersection of the corresponding sink task with interval $[\texttt{START},\texttt{START}+\texttt{WINDOW\_SIZE}-1]$. The constraint sliding_time_window_from_start holds since the sum of the previous intersections does not exceed LIMIT.

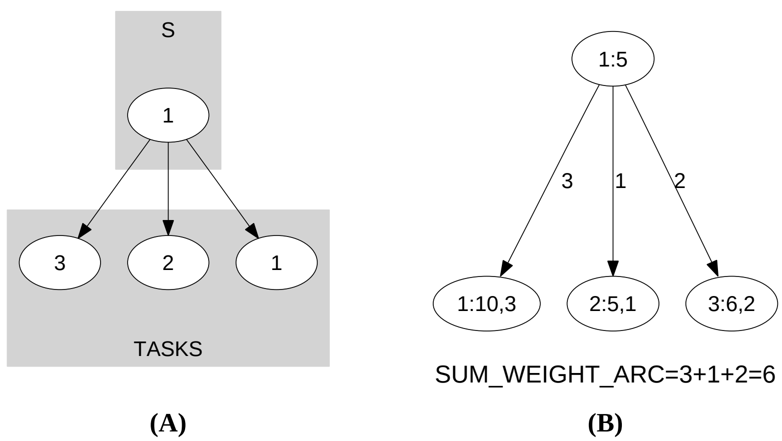



Figure 5.709: Initial and final graph of the sliding_time_window_from_start constraint

## 5.355 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 $\geq 0$
required(TASKS, [origin, end, npoint])
TASKS.origin $\leq$ TASKS.end
TASKS.npoint $\geq 0$

**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 5.710 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 5.710 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 $\geq 0$.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.

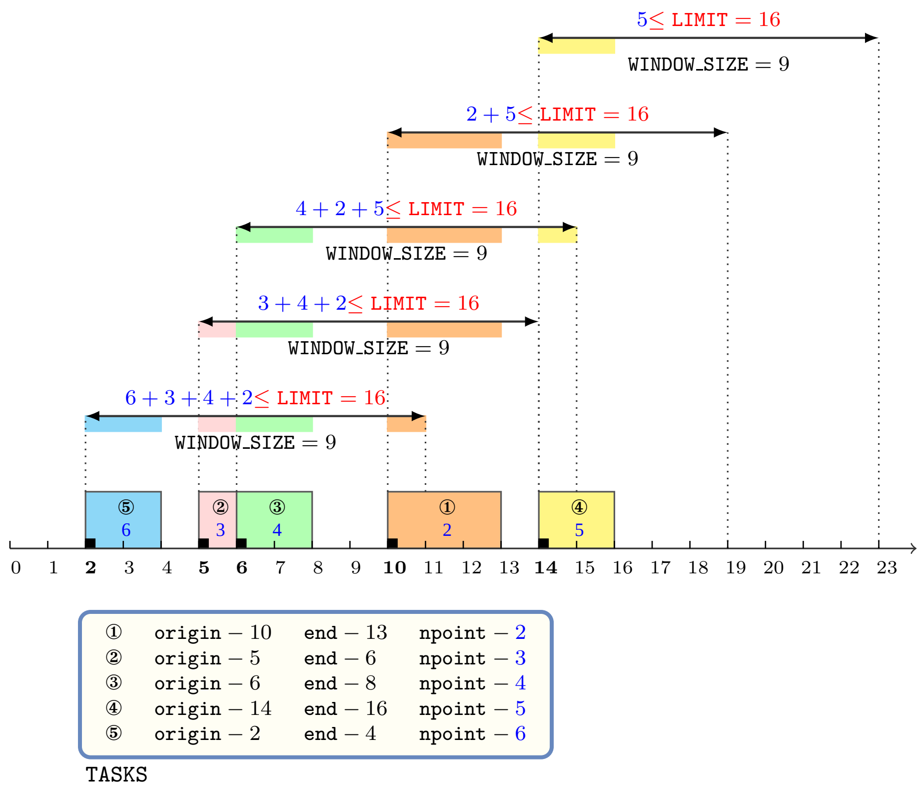



Figure 5.710: Time windows and their use for the five tasks of the **Example** slot

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

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

**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.

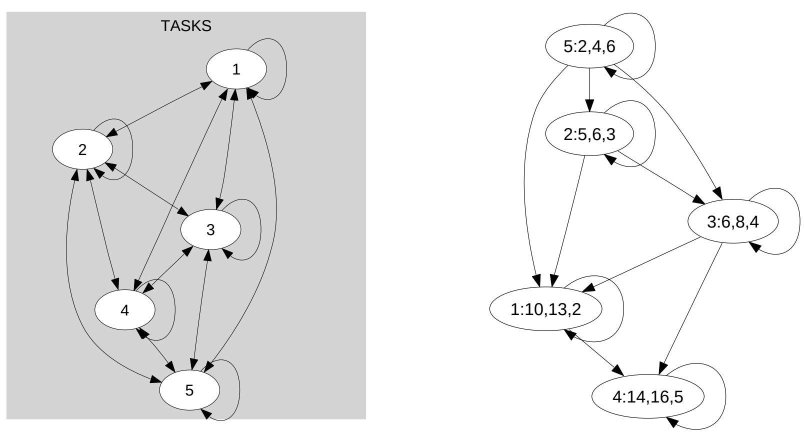
| | |
|---|---|
| **Arc input(s)** | TASKS |
| **Arc generator** | $SELF \mapsto$ collection(tasks) |
| **Arc arity** | 1 |
| **Arc constraint(s)** | tasks.origin $\leq$ tasks.end |
| **Graph property(ies)** | **NARC** $=$ \|TASKS\| |

| | |
|---|---|
| **Arc input(s)** | TASKS |
| **Arc generator** | $CLIQUE \mapsto$ collection(tasks1, tasks2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • tasks1.end $\leq$ tasks2.end<br>• tasks2.origin $-$ tasks1.end $<$ WINDOW_SIZE $- 1$ |
| **Sets** | $\mathsf{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var} - \texttt{TASKS.npoint})]\end{array}\right)\end{array}\right]$ |
| **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 5.711 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 $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.




**(A)** **(B)**

Figure 5.711: Initial and final graph of the `sliding_time_window_sum` constraint

## 5.356 smooth

DESCRIPTION LINKS GRAPH AUTOMATON

**Origin**

Derived from change.

**Constraint**

smooth(NCHANGE, TOLERANCE, VARIABLES)

**Arguments**

| | | |
|---|---|---|
| NCHANGE | : | dvar |
| TOLERANCE | : | int |
| VARIABLES | : | collection(var−dvar) |

**Restrictions**

NCHANGE $\geq 0$
NCHANGE $<$ |VARIABLES|
TOLERANCE $\geq 0$
required(VARIABLES, var)

**Purpose**

NCHANGE is the number of times that $|X - Y| >$ TOLERANCE holds; $X$ and $Y$ correspond to consecutive variables of the collection VARIABLES.

**Example**

$(1, 2, \langle 1, 3, 4, 5, 2 \rangle)$

In the example we have one change between values $5$ and $2$ since the difference in absolute value is greater than the tolerance (i.e., $|5 - 2| > 2$). Consequently the NCHANGE argument is fixed to $1$ and the smooth constraint holds.

**Typical**

TOLERANCE $> 0$
|VARIABLES| $> 3$
range(VARIABLES.var) $> 1$

**Symmetries**

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

**Arg. properties**

- Functional dependency: NCHANGE determined by TOLERANCE and VARIABLES.
- Prefix-contractible wrt. VARIABLES when NCHANGE $= 0$.
- Suffix-contractible wrt. VARIABLES when NCHANGE $= 0$.
- Prefix-contractible wrt. VARIABLES when NCHANGE $=$ |VARIABLES| $- 1$.
- Suffix-contractible wrt. VARIABLES when NCHANGE $=$ |VARIABLES| $- 1$.

**Usage**

This constraint is useful for the following problems:

- Assume that VARIABLES corresponds to the number of people that work on consecutive weeks. One may not normally increase or decrease too drastically the number of people from one week to the next week. With the smooth constraint you can state a limit on the number of drastic changes.

- Assume you have to produce a set of orders, each order having a specific attribute. You want to generate the orders in such a way that there is not a too big difference between the values of the attributes of two consecutive orders. If you cannot achieve this on two given specific orders, this would imply a set-up or a cost. Again, with the `smooth` constraint, you can control this kind of drastic changes.

**Algorithm** A first incomplete algorithm is described in [30]. The sketch of a filtering algorithm for the conjunction of the `smooth` and the `stretch` constraints based on dynamic programming achieving arc-consistency is mentioned by Lars Hellsten in [208, page 60].

**Reformulation** The `smooth` constraint can be reformulated with the `seq_bin` constraint [310] that we now introduce. Given N a domain variable, X a sequence of domain variables, and C and B two binary constraints, $\texttt{seq\_bin}(\texttt{N}, \texttt{X}, \texttt{C}, \texttt{B})$ holds if (1) N is equal to the number of C-stretches in the sequence X, and (2) B holds on any pair of consecutive variables in X. A C-stretch is a generalisation of the notion of stretch introduced by G. Pesant [305], where the equality constraint is made explicit by replacing it by a binary constraint C, i.e., a C-stretch is a maximal length subsequence of X for which the binary constraint C is satisfied on consecutive variables. $\texttt{smooth}(\texttt{NCHANGE}, \texttt{VARIABLES}, \texttt{TOLERANCE})$ can be reformulated as $\texttt{N} = \texttt{N1} - 1 \ \wedge \ \texttt{seq\_bin}(\texttt{N1}, \texttt{X}, |x_i - x_{i+1}| \leq \texttt{TOLERANCE}, \mathrm{true})$, where true is the universal constraint.

**See also** **common keyword:** `change` *(number of changes in a sequence with respect to a binary constraint)*.

**related:** `distance`.

**Keywords** **characteristic of a constraint:** automaton, automaton with counters, non-deterministic automaton, non-deterministic automaton.

**constraint arguments:** pure functional dependency.

**constraint network structure:** sliding cyclic(1) constraint network(2), Berge-acyclic constraint network.

**constraint type:** timetabling constraint.

**filtering:** glue matrix, dynamic programming.

**modelling:** number of changes, functional dependency.

**modelling exercises:** n-Amazons.

**puzzles:** n-Amazons.

| | |
|---|---|
| **Arc input(s)** | VARIABLES |
| **Arc generator** | $PATH \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | abs(variables1.var − variables2.var) > TOLERANCE |
| **Graph property(ies)** | $\mathbf{NARC}$= NCHANGE |

**Graph model** Parts (A) and (B) of Figure 5.712 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the unique arc of the final graph is stressed in bold.

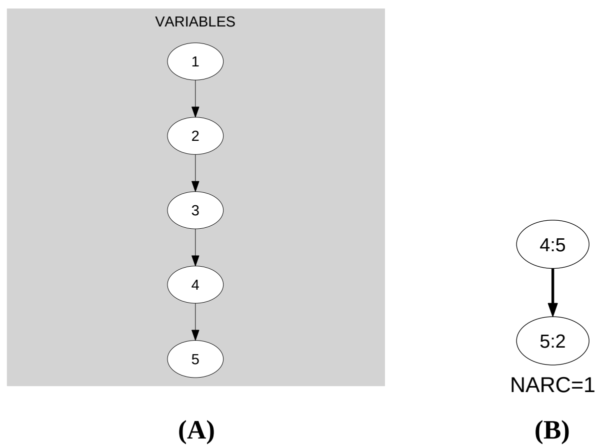



Figure 5.712: Initial and final graph of the smooth constraint

**Automaton** Figure 5.713 depicts a first automaton that only accepts all the solutions to the `smooth` constraint. This automaton uses a counter in order to record the number of satisfied constraints of the form $(|\mathtt{VAR}_i - \mathtt{VAR}_{i+1}|) > \mathtt{TOLERANCE}$ already encountered. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection `VARIABLES` corresponds a 0-1 signature variable $S_i$. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and $S_i$: $(|\mathtt{VAR}_i - \mathtt{VAR}_{i+1}|) > \mathtt{TOLERANCE} \Leftrightarrow S_i = 1$.

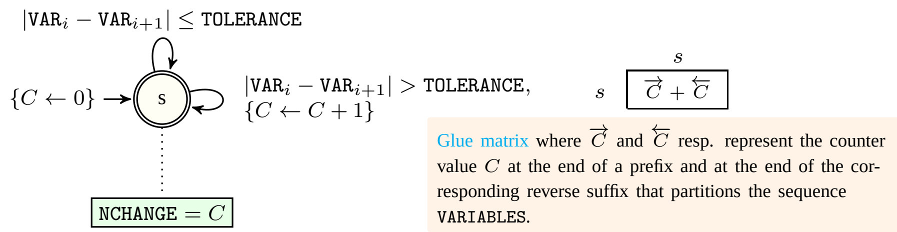



Figure 5.713: Automaton (with one counter) of the `smooth` constraint and its glue matrix

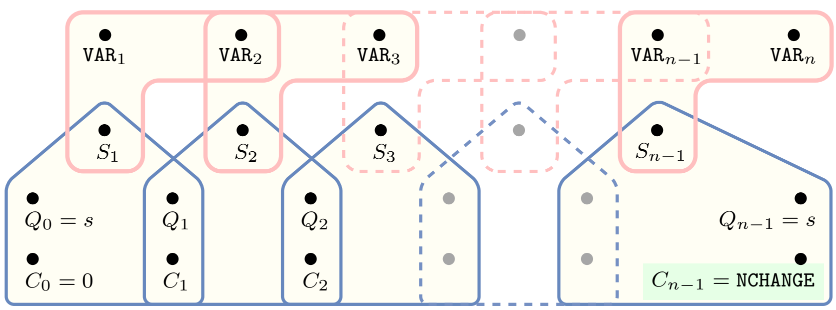



Figure 5.714: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the `smooth` constraint

Since the reformulation associated with the previous automaton is not Berge-acyclic, we now describe a second counter free automaton that also only accepts all the solutions to the `smooth` constraint. Without loss of generality, assume that the collection of variables `VARIABLES` contains at least two variables (i.e., $|\mathtt{VARIABLES}| \geq 2$). Let $n$, $min$, $max$, and $\mathcal{D}$ respectively denote the number of variables of the collection `VARIABLES`, the smallest value that can be assigned to the variables of `VARIABLES`, the largest value that can be assigned to the variables of `VARIABLES`, and the union of the domains of the variables of `VARIABLES`. Clearly, the maximum number of changes (i.e., the number of times the constraint $(|\mathtt{VAR}_i - \mathtt{VAR}_{i+1}|) > \mathtt{TOLERANCE}$ $(1 \leq i < n)$ holds) cannot exceed the quantity $m = \min(n-1, \overline{\mathtt{NCHANGE}})$. The $(m+1) \cdot |\mathcal{D}| + 2$ states of the automaton that only accepts all the solutions to the `smooth` constraint are defined in the following way:

- We have an initial state labelled by $s_I$.
- We have $m \cdot |\mathcal{D}|$ intermediate states labelled by $s_{ij}$ $(i \in \mathcal{D}, j \in [0, m])$. The first subscript $i$ of state $s_{ij}$ corresponds to the value currently encountered. The second

subscript $j$ denotes the number of already encountered satisfied constraints of the form $(|\texttt{VAR}_k - \texttt{VAR}_{k+1}|) > \texttt{TOLERANCE}$ from the initial state $s_I$ to the state $s_{ij}$.

- We have an accepting state labelled by $s_F$.

Four classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by $i$ from the initial state $s_I$ to the state $s_{i0}$, $(i \in \mathcal{D})$.
2. There is a transition, labelled by $j$, from every state $s_{ij}$, $(i \in \mathcal{D}, j \in [0, m])$, to the accepting state $s_F$.
3. $\forall i \in \mathcal{D}$, $\forall j \in [0, m]$, $\forall k \in \mathcal{D} \cap [\max(min, i - \texttt{TOLERANCE}), \min(max, i + \texttt{TOLERANCE})]$ there is a transition labelled by $k$ from $s_{ij}$ to $s_{kj}$ (i.e., the counter $j$ does not change for values $k$ that are too closed from value $i$).
4. $\forall i \in \mathcal{D}$, $\forall j \in [0, m-1]$, $\forall k \in \mathcal{D} \setminus [\max(min, i - \texttt{TOLERANCE}), \min(max, i + \texttt{TOLERANCE})]$ there is a transition labelled by $k$ from $s_{ij}$ to $s_{kj+1}$ (i.e., the counter $j$ is incremented by $+1$ for values $k$ that are too far from $i$).

We have $|\mathcal{D}|$ transitions of type $1$, $|\mathcal{D}| \cdot (m+1)$ transitions of type $2$, and at least $|\mathcal{D}|^2 \cdot m$ transitions of types $3$ and $4$. Since the maximum value of $m$ is equal to $n-1$, in the worst case we have at least $|\mathcal{D}|^2 \cdot (n-1)$ transitions. This leads to a worst case time complexity of $O(|\mathcal{D}|^2 \cdot n^2)$ if we use Pesant's algorithm for filtering the `regular` constraint [306].

Figure 5.715 depicts the corresponding counter free non deterministic automaton associated with the `smooth` constraint under the hypothesis that (1) all variables of VARIABLES are assigned a value in $\{0, 1, 2, 3\}$, (2) $|\texttt{VARIABLES}|$ is equal to $4$, and (3) TOLERANCE is equal to $1$.

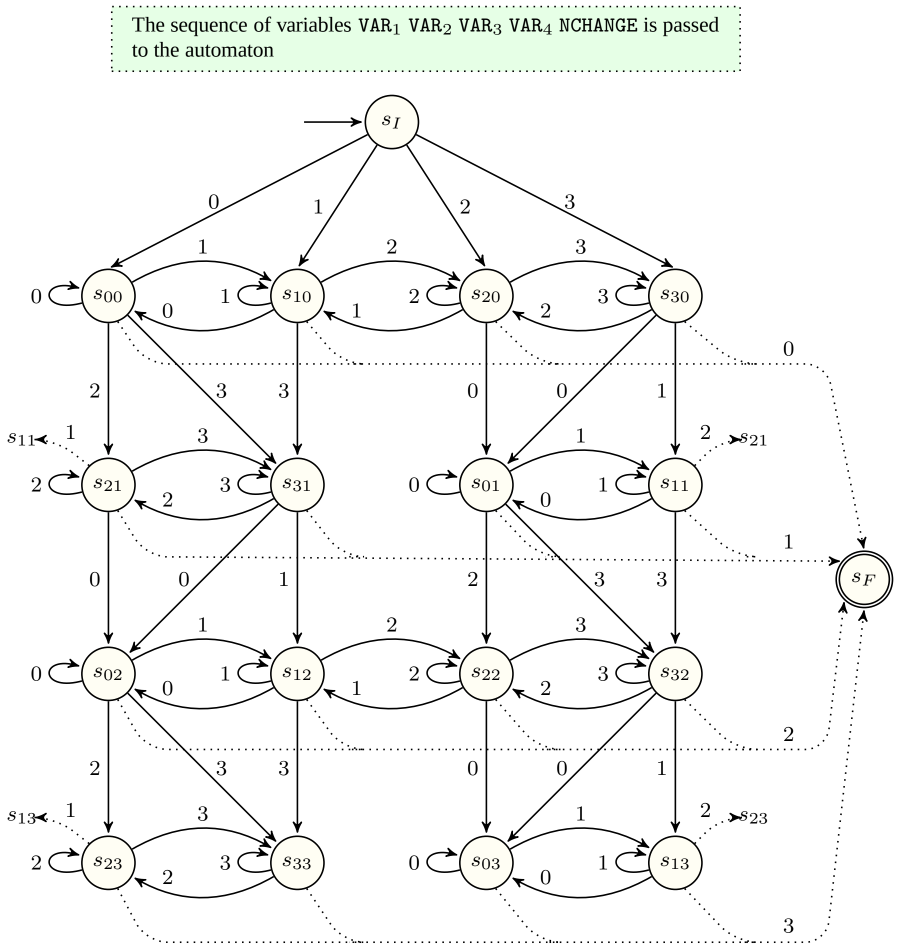



Figure 5.715: Counter free non deterministic automaton of the $\mathtt{smooth}(\mathtt{NCHANGE}, 1, \langle \mathtt{VAR}_1, \mathtt{VAR}_2, \mathtt{VAR}_3, \mathtt{VAR}_4 \rangle)$ constraint assuming $\mathtt{VAR}_i \in [0, 3]$ $(1 \leq i \leq 3)$, with initial state $s_I$ and accepting state $s_F$

## 5.357 soft_all_equal_max_var

DESCRIPTION LINKS GRAPH

**Origin** [149]

**Constraint** soft_all_equal_max_var(N, VARIABLES)

**Arguments**

N : dvar
VARIABLES : collection(var−dvar)

**Restrictions**

$N \geq 0$
$N \leq |VARIABLES|$
required(VARIABLES, var)

**Purpose**

Let $M$ be the number of occurrences of the most often assigned value to the variables of the VARIABLES collection. N is less than or equal to the total number of variables of the VARIABLES collection minus $M$ (i.e., N is less than or equal to the minimum number of variables that need to be reassigned in order to obtain a solution where all variables are assigned a same value).

**Example**

$(1, \langle 5, 1, 5, 5\rangle)$

Within the collection $\langle 5, 1, 5, 5\rangle$, $3$ is the number of occurrences of the most assigned value. Consequently, the soft_all_equal_max_var constraint holds since the argument $N = 1$ is less than or equal to the total number of variables $4$ minus $3$.

**Typical**

$N > 0$
$N < |VARIABLES|$
$N < |VARIABLES|/10 + 2$
$|VARIABLES| > 1$

**Symmetries**

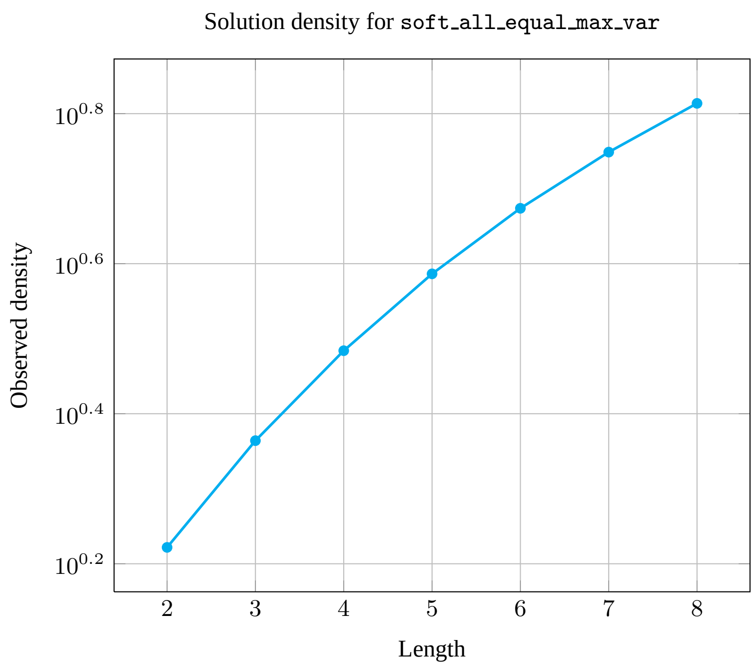
- N can be decreased to any value $\geq 0$.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

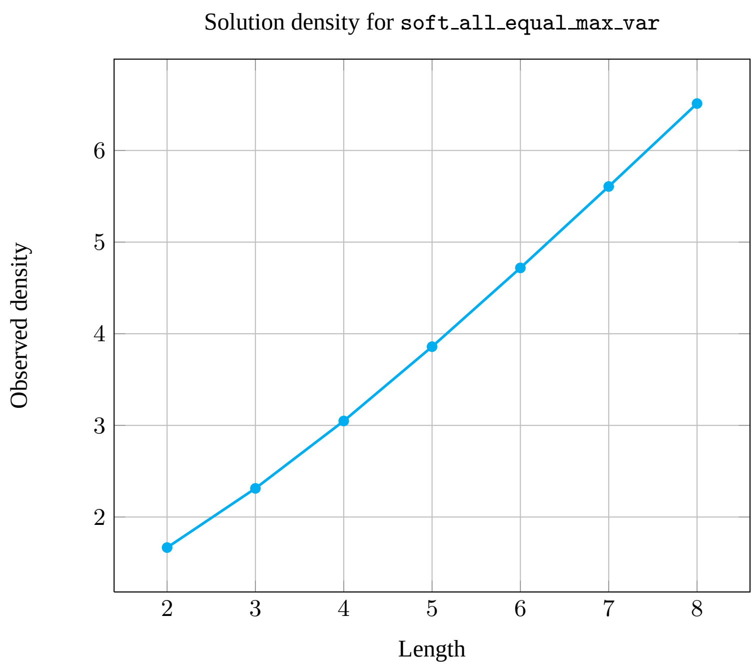
**Algorithm** [149].

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 15 | 148 | 1905 | 30006 | 555121 | 11758048 | 280310337 |

Number of solutions for soft_all_equal_max_var: domains $0..n$

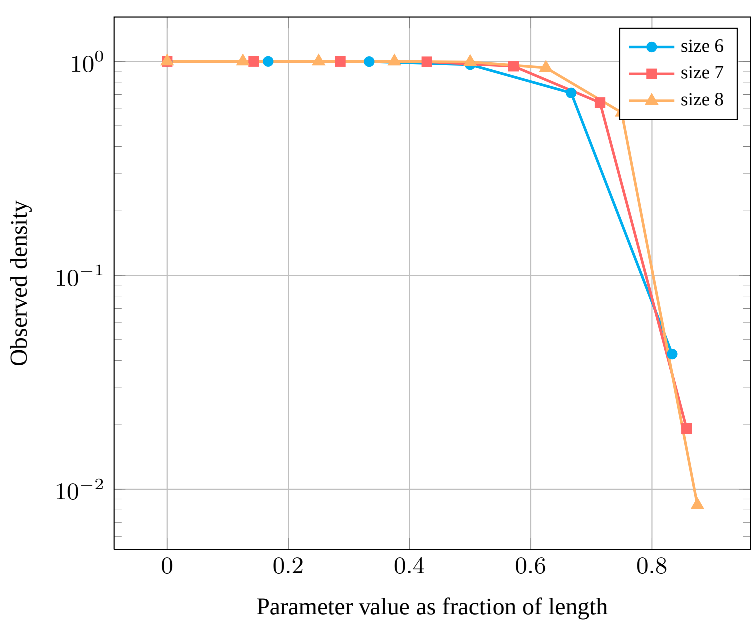

Solution density for soft_all_equal_max_var
Observed density
$10^{0.8}$
$10^{0.6}$
$10^{0.4}$
$10^{0.2}$
2 3 4 5 6 7 8
Length

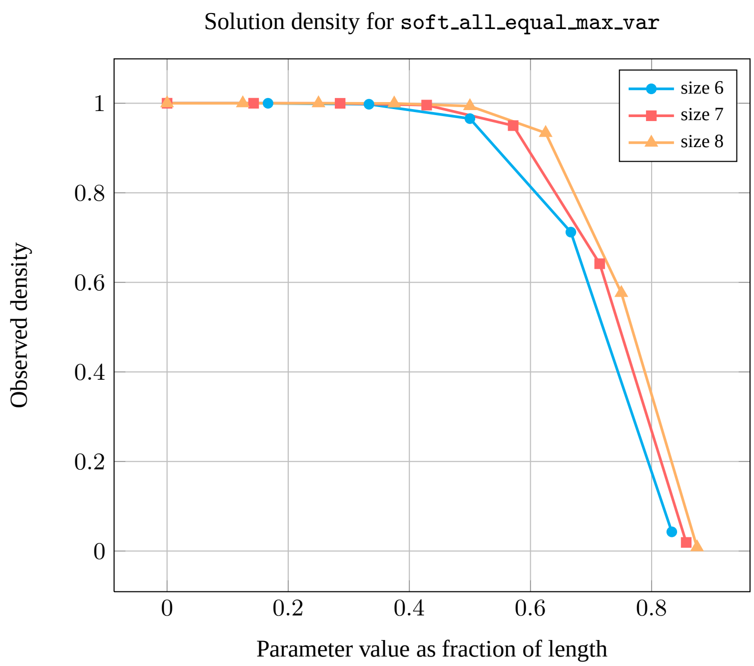

Solution density for soft_all_equal_max_var
Observed density
6
5
4
3
2
2 3 4 5 6 7 8
Length

| Length ($n$) | | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Total | | 15 | 148 | 1905 | 30006 | 555121 | 11758048 | 280310337 |
| Parameter value | 0 | 9 | 64 | 625 | 7776 | 117649 | 2097152 | 43046721 |
| | 1 | 6 | 60 | 620 | 7770 | 117642 | 2097144 | 43046712 |
| | 2 | - | 24 | 540 | 7620 | 117390 | 2096752 | 43046136 |
| | 3 | - | - | 120 | 6120 | 113610 | 2088520 | 43030008 |
| | 4 | - | - | - | 720 | 83790 | 1992480 | 42771960 |
| | 5 | - | - | - | - | 5040 | 1345680 | 40194000 |
| | 6 | - | - | - | - | - | 40320 | 24811920 |
| | 7 | - | - | - | - | - | - | 362880 |

Solution count for `soft_all_equal_max_var`: domains $0..n$

Solution density for `soft_all_equal_max_var`






**See also**

**common keyword:** soft_all_equal_min_ctr, soft_all_equal_min_var, soft_alldifferent_ctr, soft_alldifferent_var (*soft constraint*).

**hard version:** all_equal.

**implied by:** xor.

**related:** atmost_nvalue.

**Keywords**

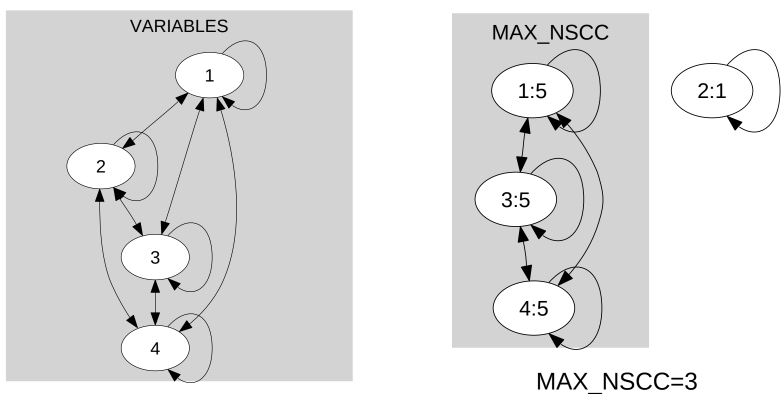
**constraint type:** soft constraint, value constraint, relaxation, variable-based violation measure.

**filtering:** arc-consistency, bound-consistency.

| | |
|---|---|
| **Arc input(s)** | VARIABLES |
| **Arc generator** | $CLIQUE \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var = variables2.var |
| **Graph property(ies)** | **MAX_NSCC** $\leq$ \|VARIABLES\| − N |

**Graph model** We generate an initial graph with binary *equalities* constraints between each vertex and its successors. The graph property states that N is less than or equal to the difference between the total number of vertices of the initial graph and the number of vertices of the largest strongly connected component of the final graph.

Parts (A) and (B) of Figure 5.716 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph.




**(A)** **(B)**

Figure 5.716: Initial and final graph of the soft_all_equal_max_var constraint

## 5.358 soft_all_equal_min_ctr

DESCRIPTION LINKS GRAPH

**Origin** [205]

**Constraint** `soft_all_equal_min_ctr(N, VARIABLES)`

**Synonyms** `soft_alldiff_max_ctr`, `soft_alldifferent_max_ctr`, `soft_alldistinct_max_ctr`.

**Arguments**

| | | |
|---|---|---|
| N | : | int |
| VARIABLES | : | collection(var−dvar) |

**Restrictions**

$\texttt{N} \geq 0$
$\texttt{N} \leq |\texttt{VARIABLES}| * |\texttt{VARIABLES}| - |\texttt{VARIABLES}|$
`required(VARIABLES, var)`

**Purpose** Consider the *equality* constraints involving two distinct variables of the collection VARIABLES. Among the previous set of constraints, N is less than or equal to the number of *equality* constraints that hold.

**Example** $(6, \langle 5, 1, 5, 5\rangle)$

Within the collection $\langle 5, 1, 5, 5\rangle$ six equality constraints holds. Consequently, the `soft_all_equal_ctr` constraint holds since the argument $\texttt{N} = 6$ is less than or equal to the number of equality constraints that hold.

**Typical**

$\texttt{N} > 0$
$\texttt{N} < |\texttt{VARIABLES}| * |\texttt{VARIABLES}| - |\texttt{VARIABLES}|$
$|\texttt{VARIABLES}| > 1$

**Symmetries**

- N can be decreased to any value $\geq 0$.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

**Remark** It was shown in [205] that, finding out whether the `soft_all_equal_ctr` constraint has a solution or not is NP-hard. This was achieved by reduction from 3-dimensional-matching. Hebrard *et al.* also identify a tractable class when no value occurs in more than two variables of the collection VARIABLES that is equivalent to the vertex matching problem. One year later, [149] shows how to achieve bound-consistency in polynomial time.

**See also** **common keyword:** `soft_all_equal_max_var`, `soft_all_equal_min_var`, `soft_alldifferent_ctr`, `soft_alldifferent_var` *(soft constraint)*.

**hard version:** `all_equal`.

**implied by:** `and`, `balance`, `equivalent`, `nor`.

**related:** `atmost_nvalue`.

**Keywords**

**complexity:** 3-dimensional-matching.

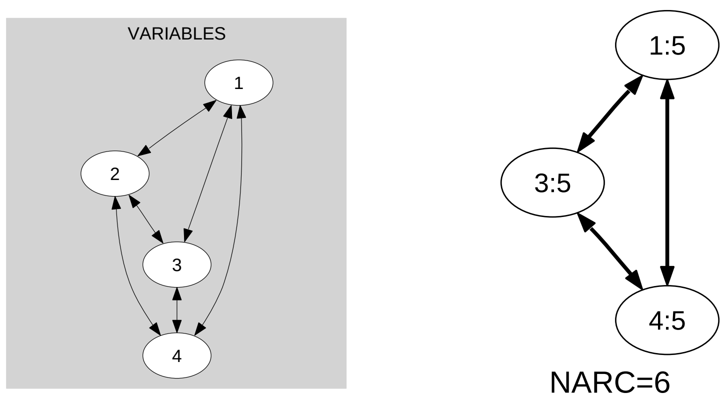
**constraint type:** soft constraint, value constraint, relaxation, decomposition-based violation measure.

**filtering:** bound-consistency.

| | |
|---|---|
| **Arc input(s)** | VARIABLES |
| **Arc generator** | $CLIQUE(\neq) \mapsto$collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var = variables2.var |
| **Graph property(ies)** | $\mathbf{NARC} \geq$ N |

**Graph model** We generate an initial graph with binary *equalities* constraints between each vertex and its successors. We use the arc generator $CLIQUE(\neq)$ in order to avoid considering *equality* constraints between the same variable. The graph property states that N is less than or equal to the number of *equalities* that hold in the final graph.

Parts (A) and (B) of Figure 5.717 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold. Six equality constraints remain in the final graph.




**(A)** **(B)**

Figure 5.717: Initial and final graph of the soft_all_equal_min_ctr constraint

## 5.359 soft_all_equal_min_var

DESCRIPTION LINKS GRAPH

**Origin** [149]

**Constraint** soft_all_equal_min_var(N, VARIABLES)

**Arguments**

| | | |
|---|---|---|
| N | : | dvar |
| VARIABLES | : | collection(var−dvar) |

**Restrictions**

$\texttt{N} \geq 0$
required(VARIABLES, var)

**Purpose** Let $M$ be the number of occurrences of the most often assigned value to the variables of the VARIABLES collection. N is greater than or equal to the total number of variables of the VARIABLES collection minus $M$ (i.e., N is greater than or equal to the minimum number of variables that need to be reassigned in order to obtain a solution where all variables are assigned a same value).

**Example** $(1, \langle 5, 1, 5, 5\rangle)$

Within the collection $\langle 5, 1, 5, 5\rangle$, $3$ is the number of occurrences of the most assigned value. Consequently, the soft_all_equal_min_var constraint holds since the argument $\texttt{N} = 1$ is greater than or equal to the total number of variables $4$ minus $3$.

**Typical**

$\texttt{N} > 0$
$\texttt{N} < |\texttt{VARIABLES}|$
$\texttt{N} < |\texttt{VARIABLES}|/10 + 2$
$|\texttt{VARIABLES}| > 1$

**Symmetries**

- N can be increased.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

**Algorithm** Let $m$ denote the total number of potential values that can be assigned to the variables of the VARIABLES collection. In [149], E. Hebrard *et al.* provides an $O(m)$ filtering algorithm achieving arc-consistency on the soft_all_equal_min_var constraint. The same paper also provides an algorithm with a lower complexity for achieving range consistency. Both algorithms are based on the following ideas:

- In a first phase, they both compute an *envelope* of the union $\mathcal{D}$ of the domains of the variables of the VARIABLES collection, i.e., an array $A$ that indicates for each potential value $v$ of $\mathcal{D}$, the maximum number of variables that could possibly be assigned value $v$. Let $max\_occ$ denote the maximum value over the entries of array

$A$, and let $\mathcal{V}_{max\_occ}$ denote the set of values which all occur in $max\_occ$ variables of the `VARIABLES` collection. The quantity $|$`VARIABLES`$| - max\_occ$ is a lower bound of `N`.

- In a second phase, depending on the relative ordering between $max\_occ$ and the minimum value of $|$`VARIABLES`$| -$ `N`, i.e., $|$`VARIABLES`$| - \overline{\mathtt{N}}$, we have the three following cases:
  1. When $max\_occ < |$`VARIABLES`$| - \overline{\mathtt{N}}$, the constraint `soft_all_equal_min_var` simply fails since not enough variables of the `VARIABLES` collection can be assigned the same value.
  2. When $max\_occ = |$`VARIABLES`$| - \overline{\mathtt{N}}$, the constraint `soft_all_equal_min_var` can be satisfied. In this context, a value $v$ can be removed from the domain of a variable $V$ of the `VARIABLES` collection if and only if:
     (a) value $v$ does not belong to $\mathcal{V}_{max\_occ}$,
     (b) the domain of variable $V$ contains all values of $\mathcal{V}_{max\_occ}$.
     On the one hand, the first condition can be understand as the fact that value $v$ is not a value that allows to have at least $|$`VARIABLES`$| - \overline{\mathtt{N}}$ variables assigned the same value. On the other hand, the second condition can be interpreted as the fact that variable $V$ is absolutely required in order to have at least $|$`VARIABLES`$| - \overline{\mathtt{N}}$ variables assigned the same value.
  3. When $max\_occ > |$`VARIABLES`$| - \overline{\mathtt{N}}$, the constraint `soft_all_equal_min_var` can be satisfied, but no value can be pruned.

Note that, in the context of range consistency, the first phase of the filtering algorithm can be interpreted as a sweep algorithm were:

- On the one hand, the *sweep status* corresponds to the maximum number of occurrence of variables that can be assigned a given value.
- On the other hand, the *event point series* correspond to the minimum values of the variables of the `VARIABLES` collection as well as to the maximum values $(+1)$ of the same variables.

Figure 5.718 illustrates the previous filtering algorithm on an example where `N` is equal to $1$, and where we have four variables $V_1$, $V_2$, $V_3$ and $V_4$ respectively taking their values within intervals $[1, 3]$, $[3, 7]$, $[0, 8]$ and $[5, 6]$ (see Part (A) of Figure 5.718, where the values of each variable are assigned a same colour that we retrieve in the other parts of Figure 5.718).

Part (B) of Figure 5.718 illustrates the first phase of the filtering algorithm, namely the computation of the envelope of the domains of variables $V_1$, $V_2$, $V_3$ and $V_4$. The *start events* $s_1$, $s_2$, $s_3$, $s_4$ (i.e., the events respectively associated with the minimum value of variables $V_1$, $V_2$, $V_3$, $V_4$) where the envelope is increased by $1$ are represented by the character $\uparrow$. Similarly, the *end events* (i.e., the events $e_1$, $e_2$, $e_3$, $e_4$ respectively associated with the maximum value $(+1)$ of $V_1$, $V_2$, $V_3$, $V_4$ are represented by the character $\downarrow$). Since the highest peak of the envelope is equal to $3$ we have that $max\_occ$ is equal to $3$. The values that allow to reach this highest peak are equal to $\mathcal{V}_{max\_occ} = \{3, 5, 6\}$ (i.e., shown in red in Part (B) of Figure 5.718).

Finally, Part (C) of Figure 5.718 illustrates the second phase of the filtering algorithm. Since $max\_occ = 3$ is equal to $|$`VARIABLES`$| - \overline{\mathtt{N}} = 4 - 1$ we remove from the variables whose domains contain $\mathcal{V}_{max\_occ} = \{3, 5, 6\}$ (i.e., variables $V_2$ and $V_3$) all values not in $\mathcal{V}_{max\_occ} = \{3, 5, 6\}$ (i.e., values $4$, $7$ for variable $V_2$ and values $0$, $1$, $2$, $4$, $7$, $8$ for variable $V_3$).

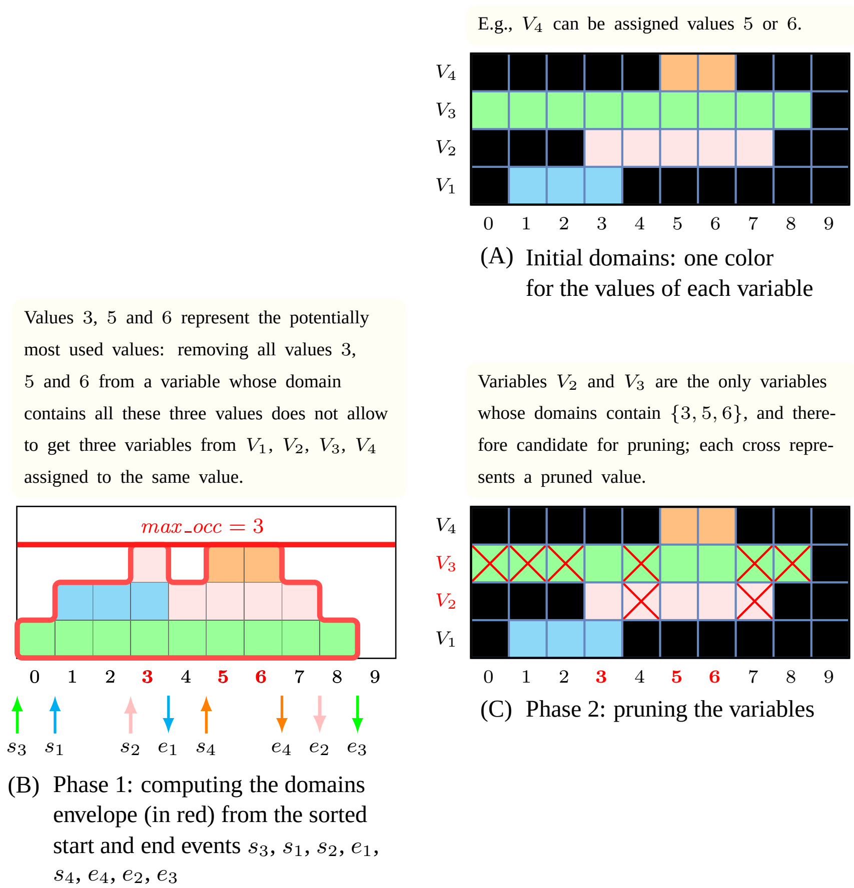
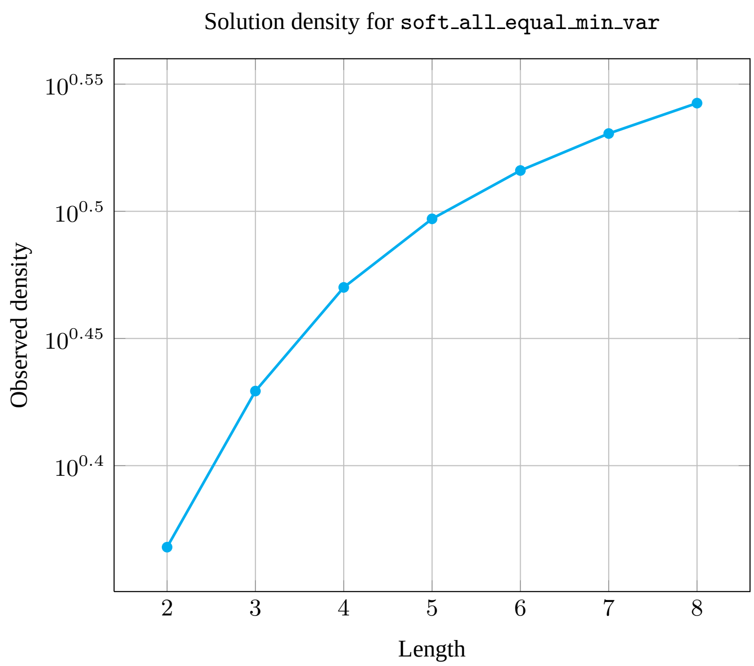


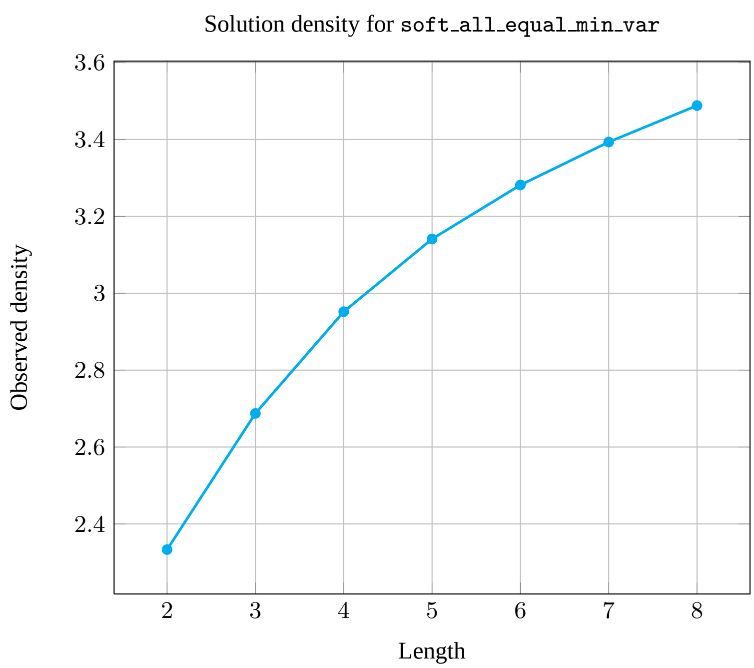

Figure 5.718: Illustration of the two phases filtering algorithm of the `soft_all_equal_min_var`$(1, \langle V_1, V_2, V_3, V_4\rangle)$ constraint with $V_1 \in [1,3]$, $V_2 \in [3,7]$, $V_3 \in [0,8]$ and $V_4 \in [5,6]$

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 21 | 172 | 1845 | 24426 | 386071 | 7116320 | 150156873 |

Number of solutions for `soft_all_equal_min_var`: domains $0..n$

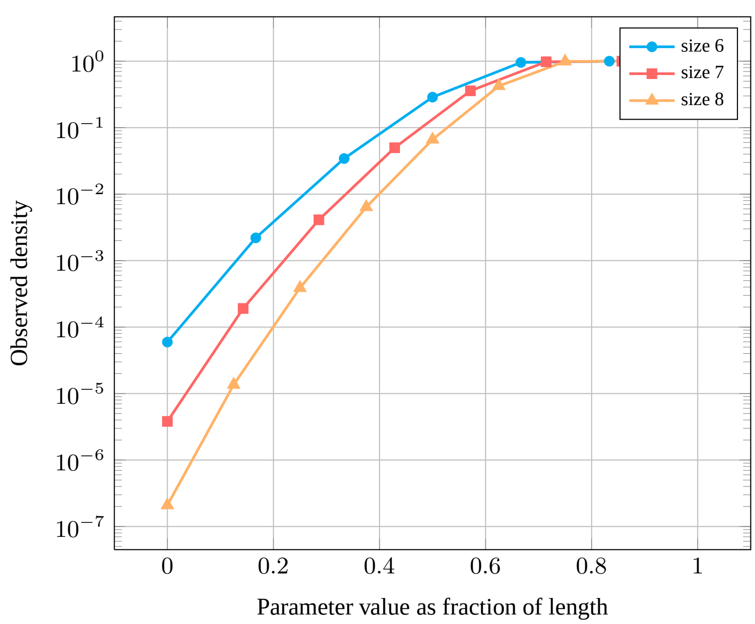

Solution density for soft_all_equal_min_var
Observed density
$10^{0.55}$
$10^{0.5}$
$10^{0.45}$
$10^{0.4}$
2
3
4
5
6
7
8
Length

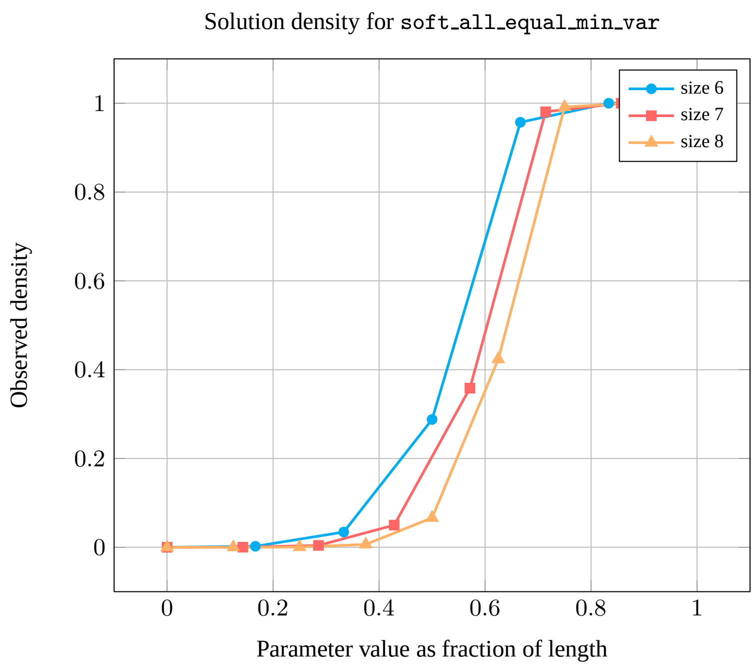

Solution density for soft_all_equal_min_var
Observed density
3.6
3.4
3.2
3
2.8
2.6
2.4
2
3
4
5
6
7
8
Length

| Length ($n$) | | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Total | | 21 | 172 | 1845 | 24426 | 386071 | 7116320 | 150156873 |
| Parameter value | 0 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 1 | 9 | 40 | 85 | 156 | 259 | 400 | 585 |
| | 2 | 9 | 64 | 505 | 1656 | 4039 | 8632 | 16713 |
| | 3 | - | 64 | 625 | 7056 | 33859 | 104672 | 274761 |
| | 4 | - | - | 625 | 7776 | 112609 | 751472 | 2852721 |
| | 5 | - | - | - | 7776 | 117649 | 2056832 | 18234801 |
| | 6 | - | - | - | - | 117649 | 2097152 | 42683841 |
| | 7 | - | - | - | - | - | 2097152 | 43046721 |
| | 8 | - | - | - | - | - | - | 43046721 |

Solution count for `soft_all_equal_min_var`: domains $0..n$

Solution density for `soft_all_equal_min_var`






**See also**

**common keyword:** `soft_all_equal_max_var`, `soft_all_equal_min_ctr`, `soft_alldifferent_ctr`, `soft_alldifferent_var` (*soft constraint*).

**hard version:** `all_equal`.

**implied by:** `xor`.

**related:** `atmost_nvalue`.

**Keywords**

**constraint type:** soft constraint, value constraint, relaxation, variable-based violation measure.

**filtering:** arc-consistency, sweep.

| | |
|---|---|
| **Arc input(s)** | VARIABLES |
| **Arc generator** | $CLIQUE \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var = variables2.var |
| **Graph property(ies)** | $\mathbf{MAX\_NSCC} \geq$ \|VARIABLES\| − N |

**Graph model** We generate an initial graph with binary *equalities* constraints between each vertex and its successors. The graph property states that N is greater than or equal to the difference between the total number of vertices of the initial graph and the number of vertices of the largest strongly connected component of the final graph.

Parts (A) and (B) of Figure 5.719 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph.

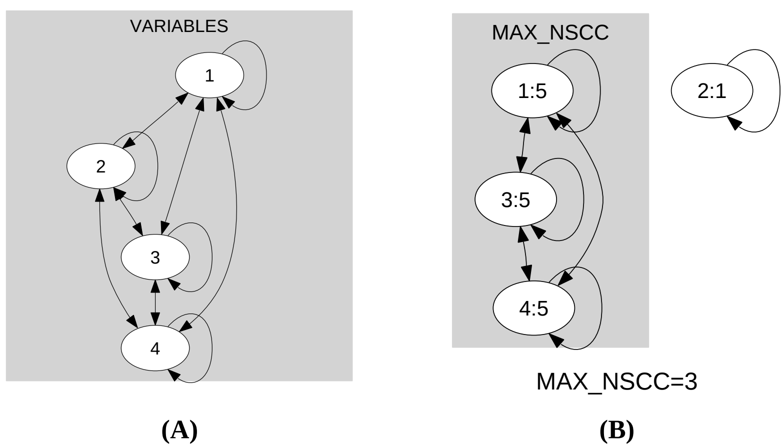



Figure 5.719: Initial and final graph of the soft_all_equal_min_var constraint

## 5.360 soft_alldifferent_ctr

DESCRIPTION LINKS GRAPH

**Origin** [314]

**Constraint** soft_alldifferent_ctr(C, VARIABLES)

**Synonyms** soft_alldiff_ctr, soft_alldistinct_ctr, soft_alldiff_min_ctr, soft_alldifferent_min_ctr, soft_alldistinct_min_ctr, soft_all_equal_max_ctr.

**Arguments**

| | | |
|---|---|---|
| C | : | dvar |
| VARIABLES | : | collection(var−dvar) |

**Restrictions**

$\mathtt{C} \geq 0$
required(VARIABLES, var)

**Purpose** Consider the *disequality* constraints involving two distinct variables VARIABLES$[i]$.var and VARIABLES$[j]$.var $(i < j)$ of the collection VARIABLES. Among the previous set of constraints, C is greater than or equal to the number of *disequality* constraints that do not hold.

**Example**

$(4, \langle 5, 1, 9, 1, 5, 5 \rangle)$
$(1, \langle 5, 1, 9, 1, 2, 6 \rangle)$
$(0, \langle 5, 1, 9, 0, 2, 6 \rangle)$

Within the collection $\langle 5, 1, 9, 1, 5, 5 \rangle$ the first and fifth values, the first and sixth values, the second and fourth values, and the fifth and sixth values are identical. Consequently, the argument $\mathtt{C} = 4$ is greater than or equal to the number of *disequality* constraints that do not hold (i.e, 4) and the soft_alldifferent_ctr constraint holds.

**Typical**

$\mathtt{C} > 0$
$\mathtt{C} \leq |\mathtt{VARIABLES}| * (|\mathtt{VARIABLES}| - 1)/2$
$|\mathtt{VARIABLES}| > 1$

**Symmetries**

- C can be increased.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

**Arg. properties** Contractible wrt. VARIABLES.

**Usage** A soft alldifferent constraint.

**Remark** The soft_alldifferent_ctr constraint is called soft_alldiff_min_ctr or soft_all_equal_max_ctr in [149].

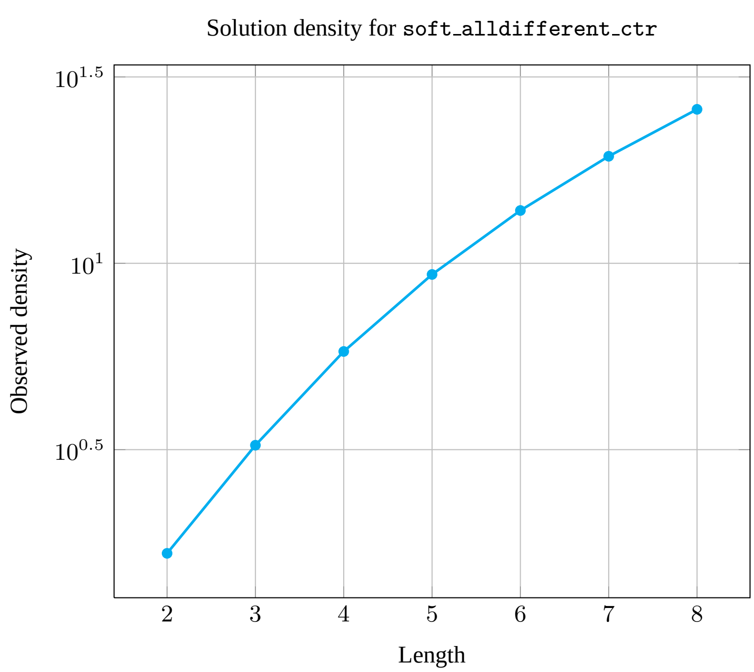
**Algorithm**

Since it focus on the soft aspect of the `alldifferent` constraint, the original article [314] that introduces this constraint describes how to evaluate the minimum value of `C` and how to prune according to the maximum value of `C`. The corresponding filtering algorithm does not achieve arc-consistency. W.-J. van Hoeve [422] presents a new filtering algorithm that achieves arc-consistency. This algorithm is based on a reformulation into a minimum-cost flow problem.

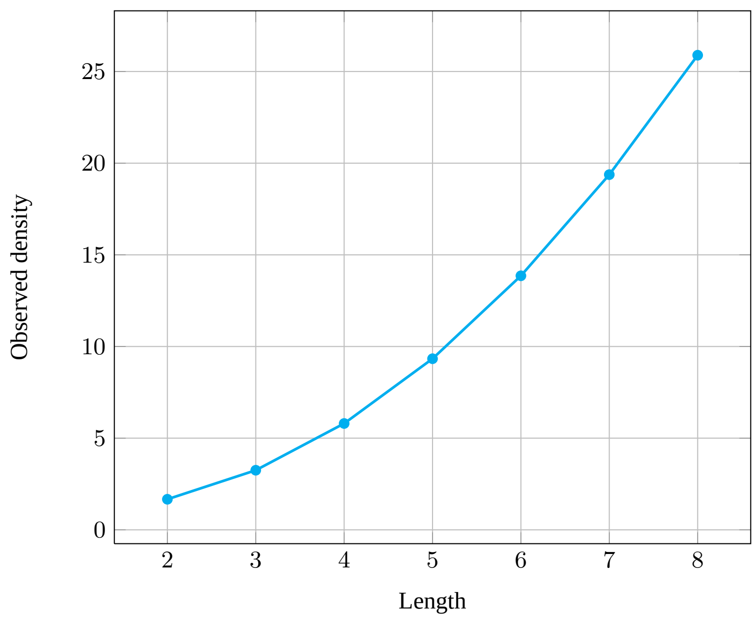
**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 15 | 208 | 3625 | 72576 | 1630279 | 40632320 | 1114431777 |

Number of solutions for `soft_alldifferent_ctr`: domains $0..n$

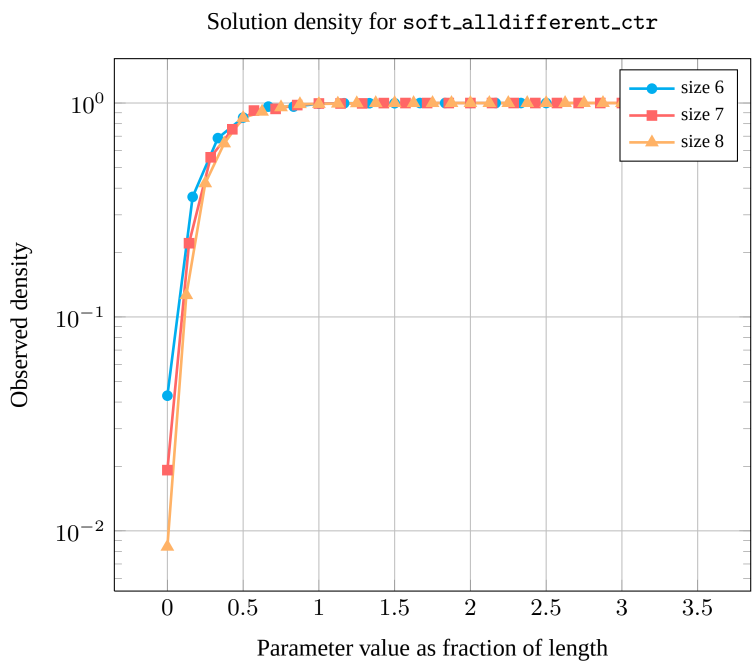


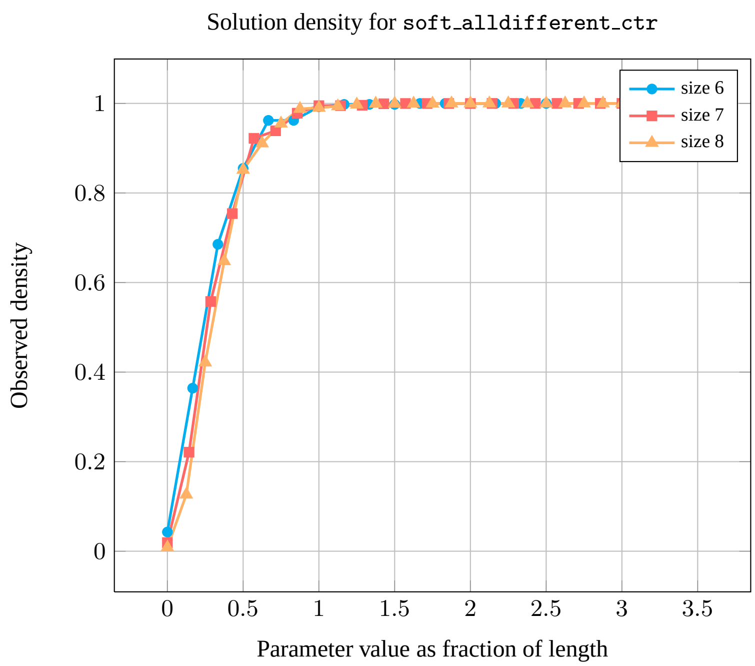
Solution density for `soft_alldifferent_ctr`



| Length ($n$) | | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Total | | 15 | 208 | 3625 | 72576 | 1630279 | 40632320 | 1114431777 |
| Parameter value | 0 | 6 | 24 | 120 | 720 | 5040 | 40320 | 362880 |
| | 1 | 9 | 60 | 480 | 4320 | 42840 | 463680 | 5443200 |
| | 2 | - | 60 | 540 | 6120 | 80640 | 1169280 | 18144000 |
| | 3 | - | 64 | 620 | 7320 | 100590 | 1580880 | 27881280 |
| | 4 | - | - | 620 | 7620 | 113190 | 1933680 | 36666000 |
| | 5 | - | - | 620 | 7620 | 113190 | 1968960 | 39206160 |
| | 6 | - | - | 625 | 7770 | 116760 | 2051280 | 41111280 |
| | 7 | - | - | - | 7770 | 117390 | 2086560 | 42522480 |
| | 8 | - | - | - | 7770 | 117390 | 2086560 | 42628320 |
| | 9 | - | - | - | 7770 | 117390 | 2088520 | 42769440 |
| | 10 | - | - | - | 7776 | 117642 | 2095576 | 42938784 |
| | 11 | - | - | - | - | 117642 | 2096752 | 43023456 |
| | 12 | - | - | - | - | 117642 | 2096752 | 43025976 |
| | 13 | - | - | - | - | 117642 | 2096752 | 43030008 |
| | 14 | - | - | - | - | 117642 | 2096752 | 43030008 |
| | 15 | - | - | - | - | 117649 | 2097144 | 43044120 |
| | 16 | - | - | - | - | - | 2097144 | 43046136 |
| | 17 | - | - | - | - | - | 2097144 | 43046136 |
| | 18 | - | - | - | - | - | 2097144 | 43046136 |
| | 19 | - | - | - | - | - | 2097144 | 43046136 |
| | 20 | - | - | - | - | - | 2097144 | 43046136 |
| | 21 | - | - | - | - | - | 2097152 | 43046712 |
| | 22 | - | - | - | - | - | - | 43046712 |
| | 23 | - | - | - | - | - | - | 43046712 |
| | 24 | - | - | - | - | - | - | 43046712 |
| | 25 | - | - | - | - | - | - | 43046712 |
| | 26 | - | - | - | - | - | - | 43046712 |
| | 27 | - | - | - | - | - | - | 43046712 |
| | 28 | - | - | - | - | - | - | 43046721 |

Solution count for `soft_alldifferent_ctr`: domains $0..n$




**See also** **common keyword:** `soft_all_equal_max_var`, `soft_all_equal_min_ctr`, `soft_all_equal_min_var`, `soft_alldifferent_var` (*soft constraint*).

**hard version:** `alldifferent`.

**implied by:** `equivalent`, `imply`.

**implies:** `soft_alldifferent_var`.

**related:** `atmost_nvalue`.

**Keywords**

**characteristic of a constraint:** all different, disequality.

**constraint type:** soft constraint, value constraint, relaxation, decomposition-based violation measure.

**filtering:** minimum cost flow.

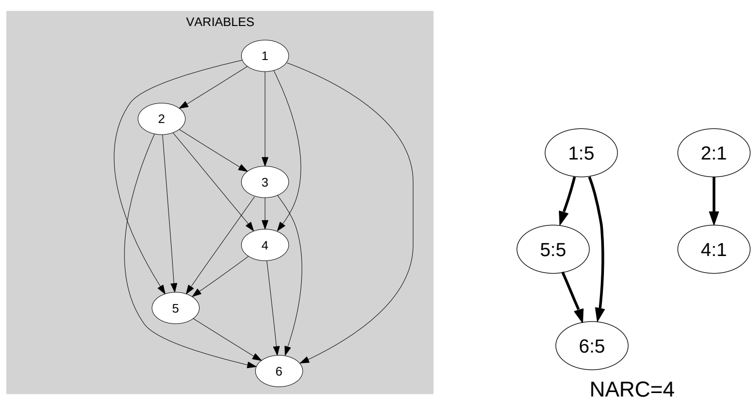
**modelling:** degree of diversity of a set of solutions.

**modelling exercises:** degree of diversity of a set of solutions.

| | |
|---|---|
| **Arc input(s)** | `VARIABLES` |
| **Arc generator** | $CLIQUE(<) \mapsto$ `collection(variables1, variables2)` |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `variables1.var` $=$ `variables2.var` |
| **Graph property(ies)** | **NARC** $\leq$ `C` |

**Graph model** We generate an initial graph with binary *equalities* constraints between each vertex and its successors. We use the arc generator $CLIQUE(<)$ in order to avoid counting twice the same *equality* constraint. The graph property states that `C` is greater than or equal to the number of *equalities* that hold in the final graph.

Parts (A) and (B) of Figure 5.720 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Since four equality constraints remain in the final graph the *cost* variable `C` is greater than or equal to 4.




**(A)** **(B)**

Figure 5.720: Initial and final graph of the `soft_alldifferent_ctr` constraint

## 5.361 soft_alldifferent_var

DESCRIPTION LINKS GRAPH

**Origin** [314]

**Constraint** soft_alldifferent_var(C, VARIABLES)

**Synonyms** soft_alldiff_var, soft_alldistinct_var, soft_alldiff_min_var, soft_alldifferent_min_var, soft_alldistinct_min_var.

**Arguments**

| | | |
|---|---|---|
| C | : | dvar |
| VARIABLES | : | collection(var−dvar) |

**Restrictions**

$\mathtt{C} \geq 0$
required(VARIABLES, var)

**Purpose** C is greater than or equal to the minimum number of variables of the collection VARIABLES for which the value needs to be changed in order that all variables of VARIABLES take a distinct value.

**Example**

$(3, \langle 5, 1, 9, 1, 5, 5\rangle)$
$(1, \langle 5, 1, 9, 6, 5, 3\rangle)$
$(0, \langle 8, 1, 9, 6, 5, 3\rangle)$

Within the collection $\langle 5, 1, 9, 1, 5, 5\rangle$ of the first example, $3$ and $2$ items are respectively fixed to values $5$ and $1$. Therefore one must change the values of at least $(3 - 1) + (2 - 1) = 3$ items to get back to $6$ distinct values. Consequently, the corresponding soft_alldifferent_var constraint holds since its first argument C is greater than or equal to $3$.

**Typical**

$\mathtt{C} > 0$
$2 * \mathtt{C} \leq |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$
some_equal(VARIABLES)

**Symmetries**

- C can be increased.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

**Arg. properties** Contractible wrt. VARIABLES.

**Usage** A soft alldifferent constraint.

**Remark**

Since it focus on the soft aspect of the alldifferent constraint, the original article [314], which introduce this constraint, describes how to evaluate the minimum value of C and how to prune according to the maximum value of C.

The soft_alldifferent_var constraint is called soft_alldiff_min_var in [149].

**Algorithm**

A first filtering algorithm presented in [314] achieves arc-consistency. A second filtering algorithm also achieving arc-consistency is described in [129, 130].

**Reformulation**

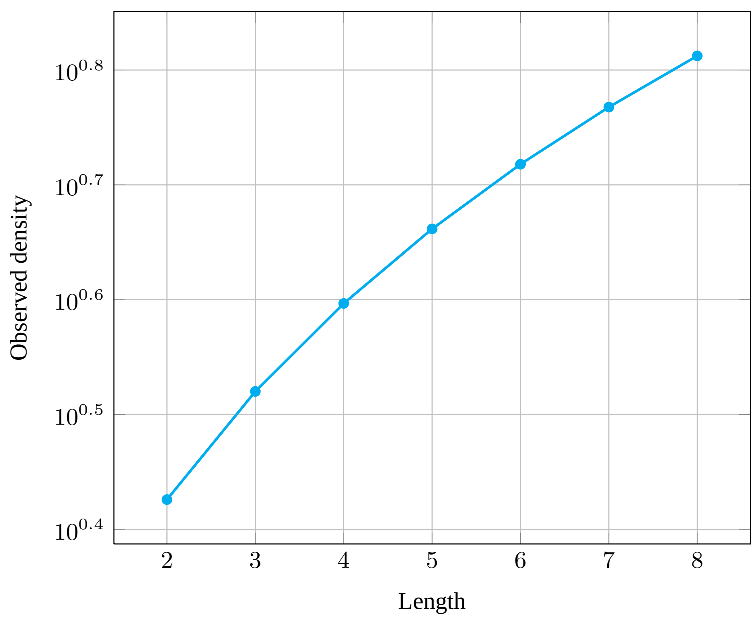
By introducing a variable $M$ that gives the number of distinct values used by variables of the collection VARIABLES, the soft_alldifferent_var$(\mathtt{C}, \mathtt{VARIABLES})$ constraint can be expressed as a conjunction of the nvalue$(M, \mathtt{VARIABLES})$ constraint and of the linear constraint $\mathtt{C} \geq |\mathtt{VARIABLES}| - M$.

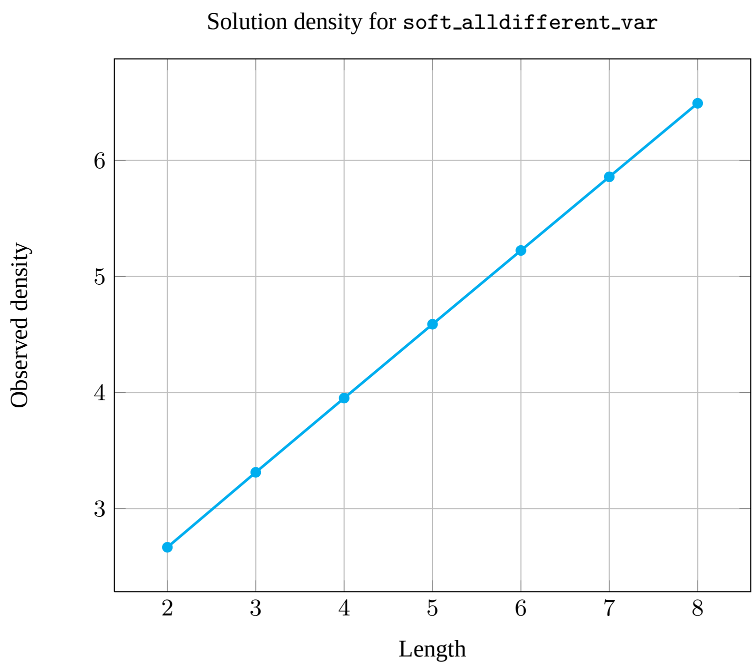
**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 24 | 212 | 2470 | 35682 | 614600 | 12286024 | 279472266 |

Number of solutions for soft_alldifferent_var: domains $0..n$

Solution density for soft_alldifferent_var

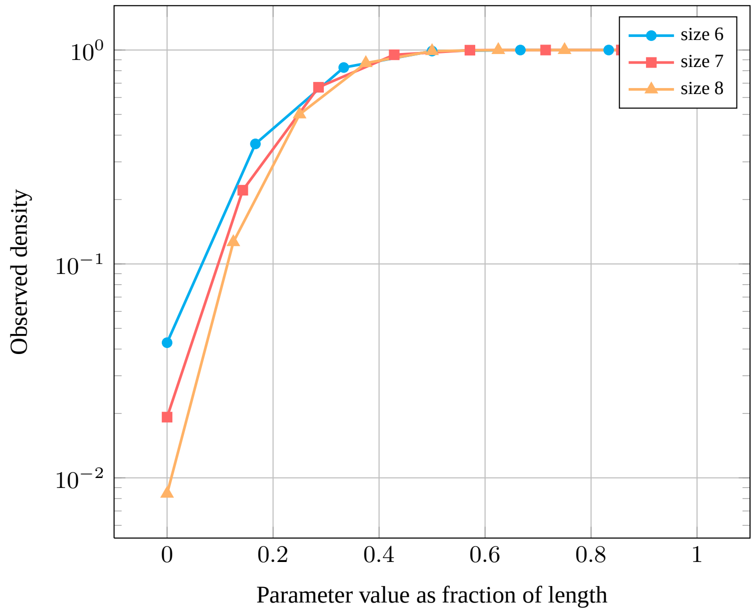




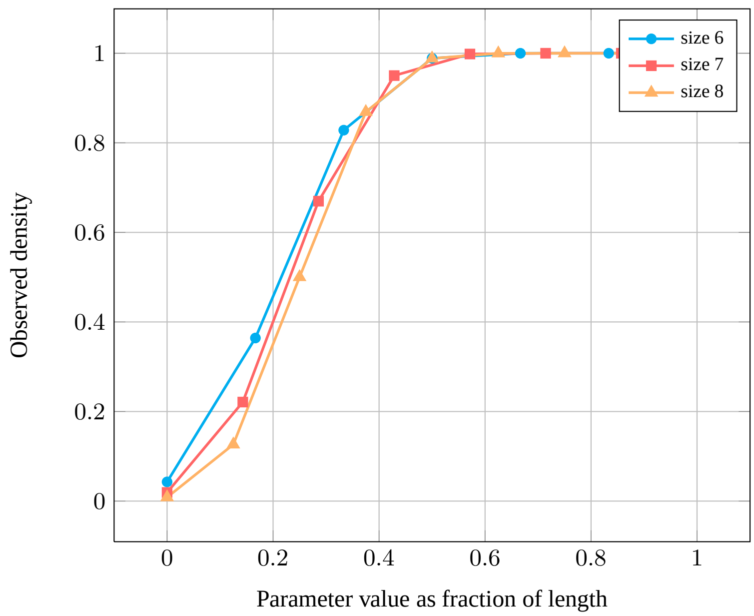

| Length ($n$) | | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Total | | 24 | 212 | 2470 | 35682 | 614600 | 12286024 | 279472266 |
| Parameter value | 0 | 6 | 24 | 120 | 720 | 5040 | 40320 | 362880 |
| | 1 | 9 | 60 | 480 | 4320 | 42840 | 463680 | 5443200 |
| | 2 | 9 | 64 | 620 | 7320 | 97440 | 1404480 | 21530880 |
| | 3 | - | 64 | 625 | 7770 | 116340 | 1992480 | 37406880 |
| | 4 | - | - | 625 | 7776 | 117642 | 2093616 | 42550704 |
| | 5 | - | - | - | 7776 | 117649 | 2097144 | 43037568 |
| | 6 | - | - | - | - | 117649 | 2097152 | 43046712 |
| | 7 | - | - | - | - | - | 2097152 | 43046721 |
| | 8 | - | - | - | - | - | - | 43046721 |

Solution count for `soft_alldifferent_var`: domains $0..n$

Solution density for `soft_alldifferent_var`




Solution density for `soft_alldifferent_var`




**See also** **common keyword:** `soft_all_equal_max_var`, `soft_all_equal_min_ctr`, `soft_all_equal_min_var`, `soft_alldifferent_ctr`, `weighted_partial_alldiff` *(soft constraint)*.

**hard version:** `alldifferent`.

**implied by:** `all_min_dist`, `alldifferent_modulo`, `soft_alldifferent_ctr`.

**related:** `atmost_nvalue`, `nvalue`.

**Keywords**

**characteristic of a constraint:** all different, disequality.

**constraint type:** soft constraint, value constraint, relaxation, variable-based violation measure.

**filtering:** bipartite matching.

**final graph structure:** strongly connected component, equivalence.

| | |
|---|---|
| **Arc input(s)** | VARIABLES |
| **Arc generator** | $CLIQUE \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var = variables2.var |
| **Graph property(ies)** | **NSCC** $\geq$ \|VARIABLES\| − C |

**Graph model** We generate a clique with binary *equalities* constraints between each pairs of vertices (this include an arc between a vertex and itself) and we state that C is equal to the difference between the total number of variables and the number of strongly connected components.

Parts (A) and (B) of Figure 5.721 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component of the final graph includes all variables that take the same value. Since we have $6$ variables and $3$ strongly connected components the *cost* variable C is greater than or equal to $6 - 3$.

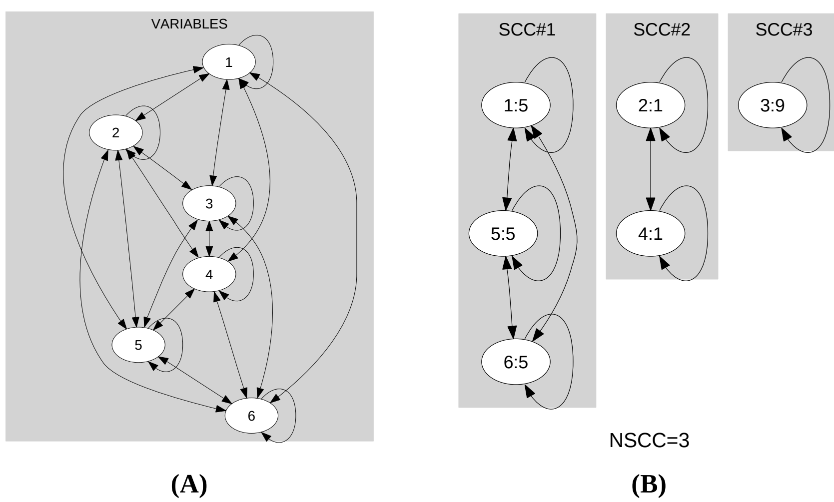



Figure 5.721: Initial and final graph of the soft_alldifferent_var constraint

## 5.362 soft_cumulative

DESCRIPTION LINKS

**Origin** Derived from cumulative

**Constraint** soft_cumulative(TASKS, LIMIT, INTERMEDIATE_LEVEL, SURFACE_ON_TOP)

**Arguments**

| | | |
|---|---|---|
| TASKS | : | collection(origin−dvar, duration−dvar, end−dvar, height−dvar) |
| LIMIT | : | int |
| INTERMEDIATE_LEVEL | : | int |
| SURFACE_ON_TOP | : | dvar |

**Restrictions**

require_at_least(2, TASKS, [origin, duration, end])
required(TASKS, height)
TASKS.duration $\geq 0$
TASKS.origin $\leq$ TASKS.end
TASKS.height $\geq 0$
LIMIT $\geq 0$
INTERMEDIATE_LEVEL $\geq 0$
INTERMEDIATE_LEVEL $\leq$ LIMIT
SURFACE_ON_TOP $\geq 0$

**Purpose**

Consider a set $\mathcal{T}$ of $n$ tasks described by the TASKS collection, where $\texttt{origin}_j$, $\texttt{duration}_j$, $\texttt{end}_j$, $\texttt{height}_j$ are shortcuts for TASKS$[j]$.origin, TASKS$[j]$.duration, TASKS$[j]$.end, TASKS$[j]$.height. In addition let $\alpha$ and $\beta$ respectively denote the earliest possible start over all tasks and the latest possible end over all tasks. The soft_cumulative constraint forces the three following conditions:

1. For each task TASKS$[j]$ $(1 \leq j \leq n)$ of $\mathcal{T}$ we have $\texttt{origin}_j + \texttt{duration}_j = \texttt{end}_j$.
2. At each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit LIMIT (i.e., $\forall i \in [\alpha, \beta]$ : $\sum_{j \in [1,n] | \texttt{origin}_j \leq i < \texttt{end}_j} \texttt{height}_j \leq \texttt{LIMIT}$).
3. The surface of the profile resource utilisation, which is greater than INTERMEDIATE_LEVEL, is equal to SURFACE_ON_TOP (i.e., $\sum_{i \in [\alpha,\beta]} \max(0, (\sum_{j \in [1,n] | \texttt{origin}_j \leq i < \texttt{end}_j} \texttt{height}_j) - \texttt{INTERMEDIATE\_LEVEL}) = \texttt{SURFACE\_ON\_TOP}$).

**Example**

$$\left(\left\langle \begin{array}{llll} \texttt{origin}-1 & \texttt{duration}-4 & \texttt{end}-5 & \texttt{height}-1, \\ \texttt{origin}-1 & \texttt{duration}-1 & \texttt{end}-2 & \texttt{height}-2, \\ \texttt{origin}-3 & \texttt{duration}-3 & \texttt{end}-6 & \texttt{height}-2 \end{array} \right\rangle, 3, 2, 3\right)$$

Figure 5.722 shows the cumulated profile associated with the example. To each task of the `cumulative` constraint corresponds a set of rectangles coloured with the same colour: the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the resource consumption of the task. The `soft_cumulative` constraint holds since:

1. For each task we have that its end is equal to the sum of its origin and its duration.
2. At each point in time we do not have a cumulated resource consumption strictly greater than the upper limit $\mathtt{LIMIT} = 3$ enforced by the second argument of the `soft_cumulative` constraint.
3. The surface of the cumulated profile located on top of the intermediate level $\mathtt{INTERMEDIATE\_LEVEL} = 2$ is equal to $\mathtt{SURFACE\_ON\_TOP} = 3$.

TASKS

| | | | | |
|---|---|---|---|---|
| ① | o − 1 | d − 4 | e − 5 | h − 1 |
| ② | o − 1 | d − 1 | e − 2 | h − 2 |
| ③ | o − 3 | d − 3 | e − 6 | h − 2 |

( `o` for `origin`, `d` for `duration`, `e` for `end`, `h` for `height` )

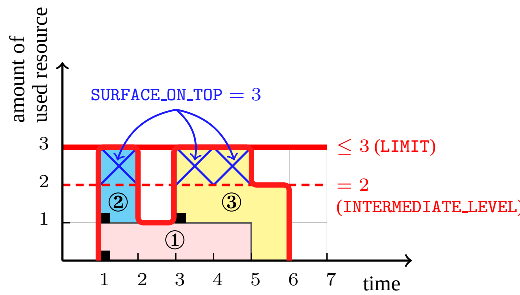



Figure 5.722: Resource consumption profile associated with the three tasks of the **Example** slot, where parts on top of the intermediate level 2 are marked by a cross

**Typical**

$|\mathtt{TASKS}| > 1$
$\mathrm{range}(\mathtt{TASKS.origin}) > 1$
$\mathrm{range}(\mathtt{TASKS.duration}) > 1$
$\mathrm{range}(\mathtt{TASKS.end}) > 1$
$\mathrm{range}(\mathtt{TASKS.height}) > 1$
$\mathtt{TASKS.duration} > 0$
$\mathtt{TASKS.height} > 0$
$\mathtt{LIMIT} < \mathrm{sum}(\mathtt{TASKS.height})$
$\mathtt{INTERMEDIATE\_LEVEL} > 0$
$\mathtt{INTERMEDIATE\_LEVEL} < \mathtt{LIMIT}$
$\mathtt{SURFACE\_ON\_TOP} > 0$

**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`.
- `LIMIT` can be increased.

**Remark**

The `soft_cumulative` constraint was initially introduced in **CHIP** [124] as a variant of the `cumulative` constraint. An extension of this constraint where one can restrict the surface on top of the intermediate level on different time intervals was first proposed in [311] and was generalised in [118].

**See also** **hard version:** `cumulative`.

**Keywords** **constraint type:** predefined constraint, soft constraint, scheduling constraint, resource constraint, temporal constraint, relaxation.

## 5.363 soft_same_interval_var

DESCRIPTION LINKS GRAPH

**Origin** Derived from `same_interval`

**Constraint** `soft_same_interval_var(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)`

**Synonym** `soft_same_interval`.

**Arguments**

| | | |
|---|---|---|
| C | : | dvar |
| VARIABLES1 | : | collection(var−dvar) |
| VARIABLES2 | : | collection(var−dvar) |
| SIZE_INTERVAL | : | int |

**Restrictions**

$\texttt{C} \geq 0$
$\texttt{C} \leq |\texttt{VARIABLES1}|$
$|\texttt{VARIABLES1}| = |\texttt{VARIABLES2}|$
`required(VARIABLES1, var)`
`required(VARIABLES2, var)`
$\texttt{SIZE\_INTERVAL} > 0$

**Purpose** Let $N_i$ (respectively $M_i$) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval $[\texttt{SIZE\_INTERVAL} \cdot i, \texttt{SIZE\_INTERVAL} \cdot i + \texttt{SIZE\_INTERVAL} - 1$. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all integer $i$ we have $N_i = M_i$.

**Example** $(4, \langle 9, 9, 9, 9, 9, 1 \rangle, \langle 9, 1, 1, 1, 1, 8 \rangle, 3)$

In the example, the fourth argument $\texttt{SIZE\_INTERVAL} = 3$ defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where $k$ is an integer. Consequently the values of the collections $\langle 9, 9, 9, 9, 9, 1 \rangle$ and $\langle 9, 1, 1, 1, 1, 8 \rangle$ are respectively located within intervals $[9, 11]$, $[9, 11]$, $[9, 11]$, $[9, 11]$, $[9, 11]$, $[0, 2]$ and intervals $[9, 11]$, $[0, 2]$, $[0, 2]$, $[0, 2]$, $[0, 2]$, $[6, 8]$. Since there is a correspondence between two pairs of intervals we must unset at least $6 - 2$ items ($6$ is the number of items of the VARIABLES1 and VARIABLES2 collections). Consequently, the `soft_same_interval_var` constraint holds since its first argument C is set to $6 - 2$.

**Typical**

$\texttt{C} > 0$
$|\texttt{VARIABLES1}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$
$\texttt{SIZE\_INTERVAL} > 1$
$\texttt{SIZE\_INTERVAL} < \texttt{range}(\texttt{VARIABLES1.var})$
$\texttt{SIZE\_INTERVAL} < \texttt{range}(\texttt{VARIABLES2.var})$

**Symmetries**

- Arguments are permutable w.r.t. permutation (C) (VARIABLES1, VARIABLES2) (SIZE_INTERVAL).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var 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 VARIABLES2.var that belongs to the $k$-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

**Usage** A soft same_interval constraint.

**Algorithm** See algorithm of the soft_same_var constraint.

**See also** **hard version:** same_interval.

**implies:** soft_used_by_interval_var.

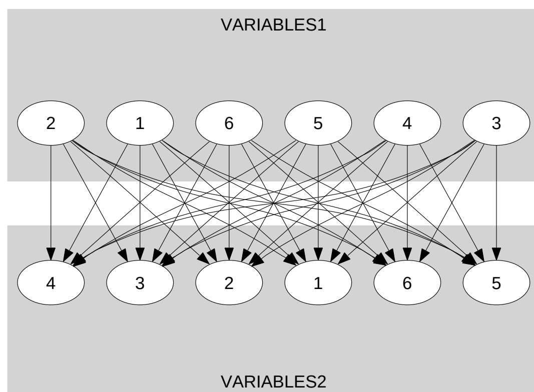
**Keywords** **constraint arguments:** constraint between two collections of variables.

**constraint type:** soft constraint, relaxation, variable-based violation measure.

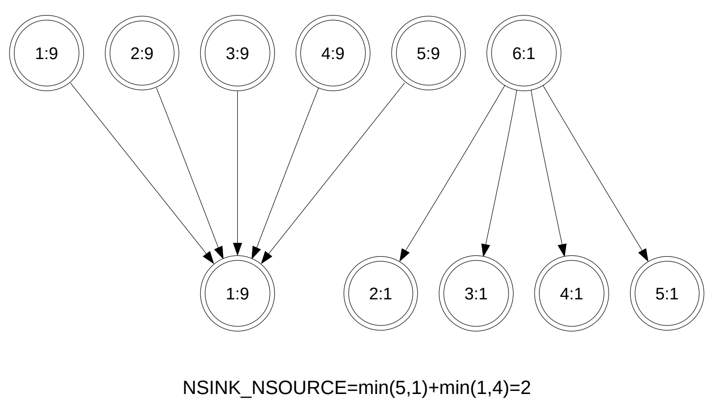
**modelling:** interval.

| | |
|---|---|
| **Arc input(s)** | VARIABLES1 VARIABLES2 |
| **Arc generator** | *PRODUCT*$\mapsto$collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL |
| **Graph property(ies)** | **NSINK_NSOURCE**= \|VARIABLES1\| − C |

**Graph model** Parts (A) and (B) of Figure 5.723 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_same_interval_var` constraint holds since the cost $4$ corresponds to the difference between the number of variables of VARIABLES1 and the sum over the different connected components of the minimum number of sources and sinks.




**(A)**




**(B)**

Figure 5.723: Initial and final graph of the `soft_same_interval_var` constraint

## 5.364 soft_same_modulo_var

DESCRIPTION LINKS GRAPH

**Origin** Derived from same_modulo

**Constraint** soft_same_modulo_var(C, VARIABLES1, VARIABLES2, M)

**Synonym** soft_same_modulo.

**Arguments**

| | | |
|---|---|---|
| C | : | dvar |
| VARIABLES1 | : | collection(var−dvar) |
| VARIABLES2 | : | collection(var−dvar) |
| M | : | int |

**Restrictions**

$\mathtt{C} \geq 0$
$\mathtt{C} \leq |\mathtt{VARIABLES1}|$
$|\mathtt{VARIABLES1}| = |\mathtt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\mathtt{M} > 0$

**Purpose** For each integer $R$ in $[0, \mathtt{M}-1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have $R$ as a rest when divided by M. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all $R$ in $[0, \mathtt{M}-1]$ we have $N1_R = N2_R$.

**Example** $(4, \langle 9, 9, 9, 9, 9, 1\rangle, \langle 9, 1, 1, 1, 1, 8\rangle, 3)$

In the example, the values of the collections $\langle 9, 9, 9, 9, 9, 1\rangle$ and $\langle 9, 1, 1, 1, 1, 8\rangle$ are respectively associated with the equivalence classes $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $1 \bmod 3 = 1$ and $9 \bmod 3 = 0$, $1 \bmod 3 = 1$, $1 \bmod 3 = 1$, $1 \bmod 3 = 1$, $1 \bmod 3 = 1$, $8 \bmod 3 = 2$. Since there is a correspondence between two pairs of equivalence classes we must unset at least $6-2$ items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). Consequently, the soft_same_modulo_var constraint holds since its first argument C is set to $6-2$.

**Typical**

$\mathtt{C} > 0$
$|\mathtt{VARIABLES1}| > 1$
range(VARIABLES1.var) $> 1$
range(VARIABLES2.var) $> 1$
$\mathtt{M} > 1$
$\mathtt{M} <$ maxval(VARIABLES1.var)
$\mathtt{M} <$ maxval(VARIABLES2.var)

**Symmetries**

- Arguments are permutable w.r.t. permutation $(\mathtt{C})$ $(\mathtt{VARIABLES1}, \mathtt{VARIABLES2})$ $(\mathtt{M})$.
- Items of `VARIABLES1` are permutable.
- Items of `VARIABLES2` are permutable.
- An occurrence of a value $u$ of `VARIABLES1.var` can be replaced by any other value $v$ such that $v$ is congruent to $u$ modulo `M`.
- An occurrence of a value $u$ of `VARIABLES2.var` can be replaced by any other value $v$ such that $v$ is congruent to $u$ modulo `M`.

**Usage** A soft `same_modulo` constraint.

**Algorithm** See algorithm of the `soft_same_var` constraint.

**See also** **hard version:** `same_modulo`.

**implies:** `soft_used_by_modulo_var`.

**Keywords** **characteristic of a constraint:** modulo.

**constraint arguments:** constraint between two collections of variables.

**constraint type:** soft constraint, relaxation, variable-based violation measure.

| | |
|---|---|
| **Arc input(s)** | `VARIABLES1` `VARIABLES2` |
| **Arc generator** | *PRODUCT* $\mapsto$ `collection`(`variables1`, `variables2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `variables1.var` mod `M` = `variables2.var` mod `M` |
| **Graph property(ies)** | **NSINK_NSOURCE** = \|`VARIABLES1`\| − `C` |

**Graph model** Parts (A) and (B) of Figure 5.724 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_same_modulo_var` constraint holds since the cost 4 corresponds to the difference between the number of variables of `VARIABLES1` and the sum over the different connected components of the minimum number of sources and sinks.

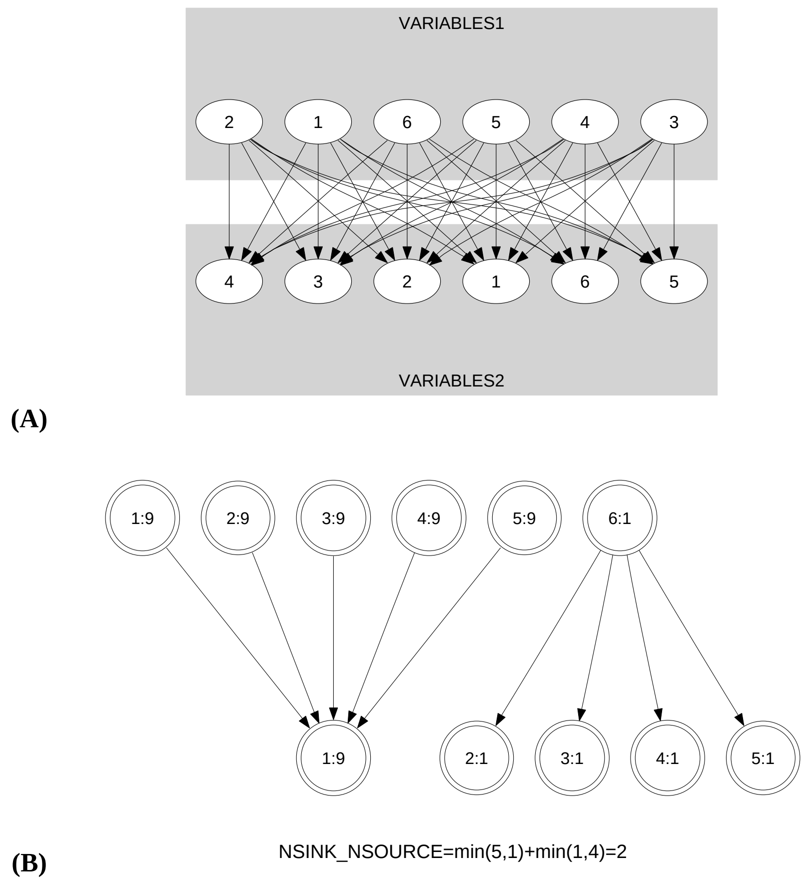



Figure 5.724: Initial and final graph of the `soft_same_modulo_var` constraint

## 5.365 soft_same_partition_var

DESCRIPTION LINKS GRAPH

**Origin** Derived from same_partition

**Constraint** soft_same_partition_var(C, VARIABLES1, VARIABLES2, PARTITIONS)

**Synonym** soft_same_partition.

**Type**

VALUES : collection(val−int)

**Arguments**

C : dvar
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

**Restrictions**

$C \geq 0$
$C \leq |VARIABLES1|$
$|VARIABLES1| = |VARIABLES2|$
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
$|PARTITIONS| \geq 2$
$|VALUES| \geq 1$
required(VALUES, val)
distinct(VALUES, val)

**Purpose**

For each integer $i$ in $[1, |PARTITIONS|]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that take their value in the $i^{th}$ partition of the collection PARTITIONS. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all $i$ in $[1, |PARTITIONS|]$ we have $N1_i = N2_i$.

**Example**

$$\left(\begin{array}{l} 4, \langle 9,9,9,9,9,1\rangle, \\ \langle 9,1,1,1,1,8\rangle, \\ \langle p-\langle 1,2\rangle, p-\langle 9\rangle, p-\langle 7,8\rangle\rangle \end{array}\right)$$

In the example, the values of the collections $\langle 9,9,9,9,9,1\rangle$ and $\langle 9,1,1,1,1,8\rangle$ are respectively associated with the partitions $p-\langle 9\rangle$, $p-\langle 9\rangle$, $p-\langle 9\rangle$, $p-\langle 9\rangle$, $p-\langle 9\rangle$, $p-\langle 1,2\rangle$ and $p-\langle 9\rangle$, $p-\langle 1,2\rangle$, $p-\langle 1,2\rangle$, $p-\langle 1,2\rangle$, $p-\langle 1,2\rangle$, $p-\langle 7,8\rangle$. Since there is a correspondence between two pairs of partitions we must unset at least $6-2$ items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). Consequently, the soft_same_partition_var constraint holds since its first argument C is set to $6-2$.

**Typical**

$\mathtt{C} > 0$
$|\mathtt{VARIABLES1}| > 1$
$\mathtt{range}(\mathtt{VARIABLES1.var}) > 1$
$\mathtt{range}(\mathtt{VARIABLES2.var}) > 1$
$|\mathtt{VARIABLES1}| > |\mathtt{PARTITIONS}|$
$|\mathtt{VARIABLES2}| > |\mathtt{PARTITIONS}|$

**Symmetries**

- Arguments are permutable w.r.t. permutation (C) (VARIABLES1, VARIABLES2) (PARTITIONS).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

**Usage** A soft same_partition constraint.

**Algorithm** See algorithm of the soft_same_var constraint.

**See also**

**hard version:** same_partition.

**implies:** soft_used_by_partition_var.

**Keywords**

**characteristic of a constraint:** partition.

**constraint arguments:** constraint between two collections of variables.

**constraint type:** soft constraint, relaxation, variable-based violation measure.

| | |
|---|---|
| **Arc input(s)** | VARIABLES1 VARIABLES2 |
| **Arc generator** | *PRODUCT* $\mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | in_same_partition(variables1.var, variables2.var, PARTITIONS) |
| **Graph property(ies)** | **NSINK_NSOURCE** $= \vert$VARIABLES1$\vert - $C |

**Graph model** Parts (A) and (B) of Figure 5.725 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The soft_same_partition_var constraint holds since the cost 4 corresponds to the difference between the number of variables of VARIABLES1 and the sum over the different connected components of the minimum number of sources and sinks.

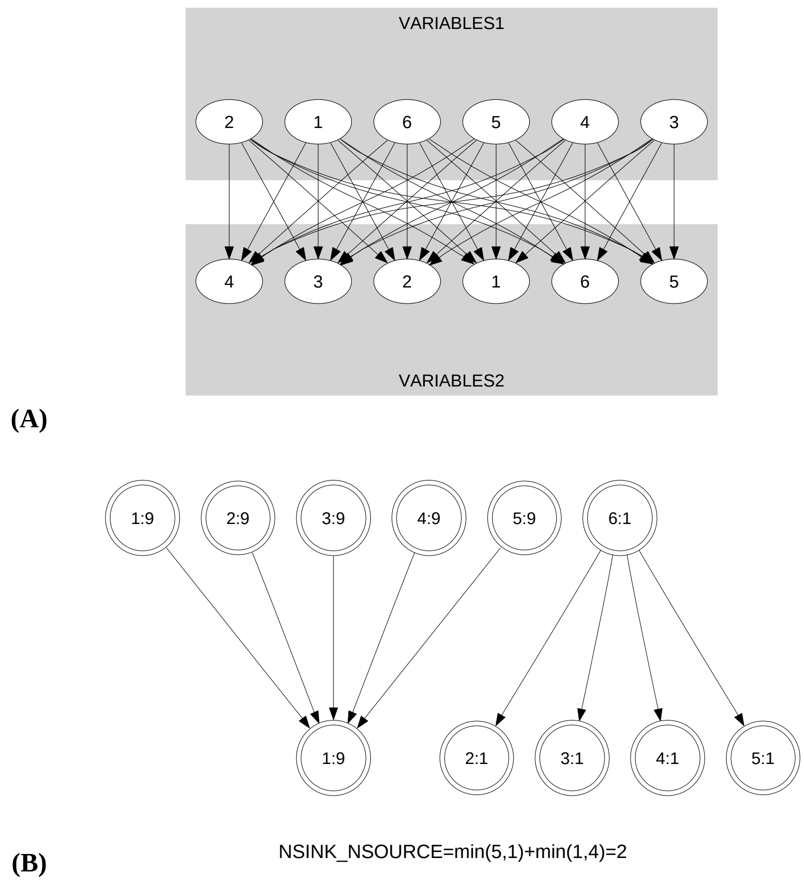



Figure 5.725: Initial and final graph of the soft_same_partition_var constraint

## 5.366 soft_same_var

DESCRIPTION LINKS GRAPH

**Origin** [423]

**Constraint** soft_same_var(C, VARIABLES1, VARIABLES2)

**Synonym** soft_same.

**Arguments**

| | | |
|---|---|---|
| C | : | dvar |
| VARIABLES1 | : | collection(var−dvar) |
| VARIABLES2 | : | collection(var−dvar) |

**Restrictions**

$C \geq 0$
$C \leq |VARIABLES1|$
$|VARIABLES1| = |VARIABLES2|$
required(VARIABLES1, var)
required(VARIABLES2, var)

**Purpose** C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that the variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation.

**Example** $(4, \langle 9, 9, 9, 9, 9, 1\rangle, \langle 9, 1, 1, 1, 1, 8\rangle)$

As illustrated by Figure 5.726, there is a correspondence between two pairs of values of the collections $\langle 9, 9, 9, 9, 9, 1\rangle$ and $\langle 9, 1, 1, 1, 1, 8\rangle$. Consequently, we must unset at least $6 - 2$ items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). The soft_same_var constraint holds since its first argument C is set to $6 - 2$.

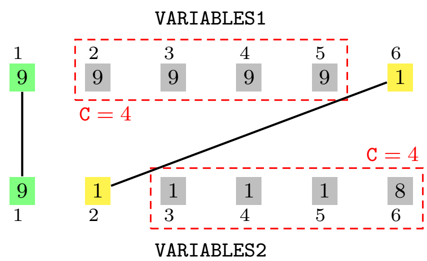



Figure 5.726: Illustration of the partial correspondence between the items of the VARIABLES1 and of the VARIABLES2 collections of the **Example** slot, i.e., C = 4 items of the VARIABLES1 or of the VARIABLES2 collections need to be changed in order to have a full correspondence

**Typical**

$\text{C} > 0$
$|\text{VARIABLES1}| > 1$
$\text{range}(\text{VARIABLES1.var}) > 1$
$\text{range}(\text{VARIABLES2.var}) > 1$

**Symmetries**

- Arguments are permutable w.r.t. permutation (C) (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

**Usage** A soft `same` constraint.

**Algorithm** A first filtering algorithm is described in [423, page 80]. A second filtering algorithm is presented in [129, 130].

**See also**

**hard version:** `same`.

**implies:** `soft_used_by_var`.

**Keywords**

**constraint arguments:** constraint between two collections of variables.

**constraint type:** soft constraint, relaxation, variable-based violation measure.

**filtering:** minimum cost flow, bipartite matching.

| | |
|---|---|
| **Arc input(s)** | VARIABLES1 VARIABLES2 |
| **Arc generator** | $PRODUCT \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var = variables2.var |
| **Graph property(ies)** | **NSINK_NSOURCE** = \|VARIABLES1\| − C |

**Graph model** Parts (A) and (B) of Figure 5.727 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The soft_same_var constraint holds since the cost $4$ corresponds to the difference between the number of variables of VARIABLES1 and the sum over the different connected components of the minimum number of sources and sinks.

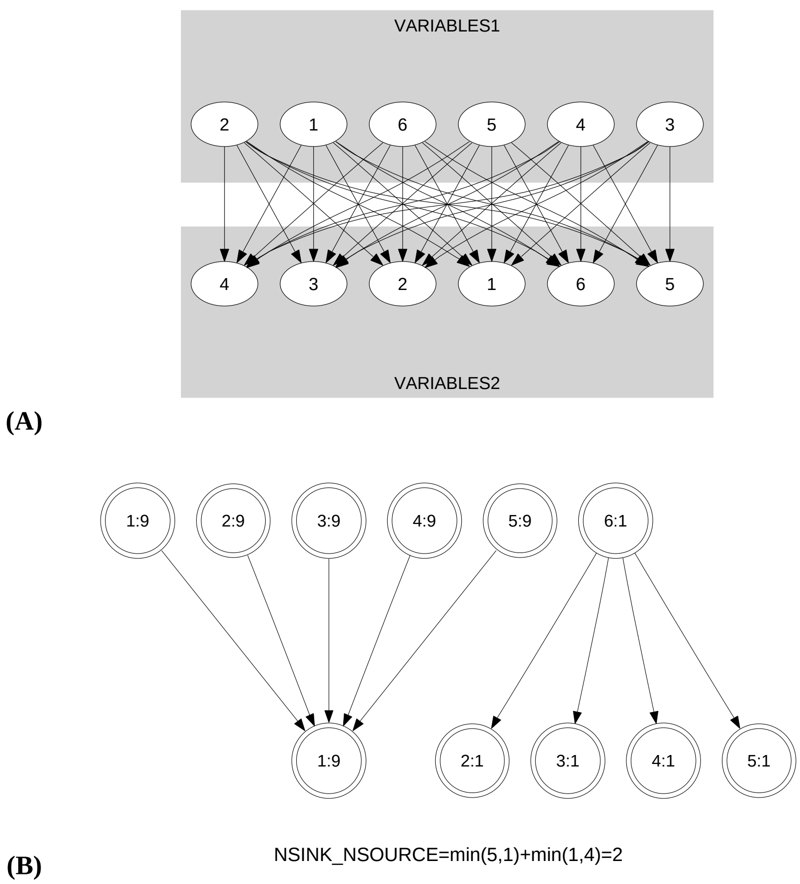



Figure 5.727: Initial and final graph of the soft_same_var constraint

## 5.367 soft_used_by_interval_var

DESCRIPTION LINKS GRAPH

**Origin**

Derived from used_by_interval.

**Constraint**

soft_used_by_interval_var(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)

**Synonym**

soft_used_by_interval.

**Arguments**

```
C             : dvar
VARIABLES1    : collection(var−dvar)
VARIABLES2    : collection(var−dvar)
SIZE_INTERVAL : int
```

**Restrictions**

$\texttt{C} \geq 0$
$\texttt{C} \leq |\texttt{VARIABLES2}|$
$|\texttt{VARIABLES1}| \geq |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\texttt{SIZE\_INTERVAL} > 0$

**Purpose**

Let $N_i$ (respectively $M_i$) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval $[\texttt{SIZE\_INTERVAL} \cdot i, \texttt{SIZE\_INTERVAL} \cdot i + \texttt{SIZE\_INTERVAL} - 1]$. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all integer $i$ we have $M_i > 0 \Rightarrow N_i \geq M_i$.

**Example**

$(2, \langle 9, 1, 1, 8, 8\rangle, \langle 9, 9, 9, 1\rangle, 3)$

In the example, the fourth argument $\texttt{SIZE\_INTERVAL} = 3$ defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where $k$ is an integer. Consequently the values of the collections $\langle 9, 1, 1, 8, 8\rangle$ and $\langle 9, 9, 9, 1\rangle$ are respectively located within intervals $[9, 11]$, $[0, 2]$, $[0, 2]$, $[6, 8]$, $[6, 8]$ and intervals $[9, 11]$, $[9, 11]$, $[9, 11]$, $[0, 2]$. Since there is a correspondence between two pairs of intervals we must unset at least $4 - 2$ items ($4$ is the number of items of the VARIABLES2 collection). Consequently, the soft_used_by_interval_var constraint holds since its first argument C is set to $4 - 2$.

**Typical**

$\texttt{C} > 0$
$|\texttt{VARIABLES1}| > 1$
$|\texttt{VARIABLES2}| > 1$
range(VARIABLES1.var) $> 1$
range(VARIABLES2.var) $> 1$
$\texttt{SIZE\_INTERVAL} > 1$
SIZE_INTERVAL $<$ range(VARIABLES1.var)
SIZE_INTERVAL $<$ range(VARIABLES2.var)

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var 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 VARIABLES2.var that belongs to the $k$-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

**Usage**

A soft used_by_interval constraint.

**See also**

**hard version:** used_by_interval.

**implied by:** soft_same_interval_var.

**Keywords**

**constraint arguments:** constraint between two collections of variables.

**constraint type:** soft constraint, relaxation, variable-based violation measure.

**modelling:** interval.

| | |
|---|---|
| **Arc input(s)** | `VARIABLES1` `VARIABLES2` |
| **Arc generator** | $PRODUCT \mapsto$ `collection(variables1, variables2)` |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `variables1.var/SIZE_INTERVAL` $=$ `variables2.var/SIZE_INTERVAL` |
| **Graph property(ies)** | **NSINK_NSOURCE** $= \lvert$`VARIABLES2`$\rvert -$ `C` |

**Graph model** Parts (A) and (B) of Figure 5.728 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_used_by_interval_var` constraint holds since the cost 2 corresponds to the difference between the number of variables of `VARIABLES2` and the sum over the different connected components of the minimum number of sources and sinks.

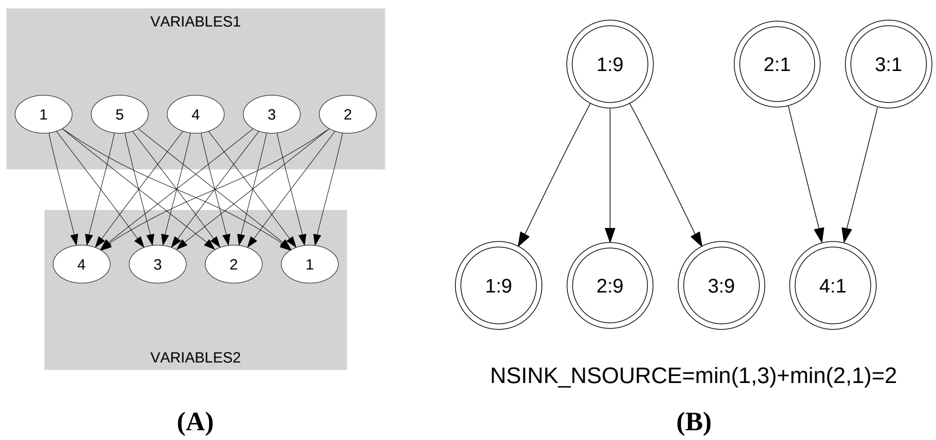



Figure 5.728: Initial and final graph of the `soft_used_by_interval_var` constraint

## 5.368 soft_used_by_modulo_var

DESCRIPTION LINKS GRAPH

**Origin**

Derived from `used_by_modulo`

**Constraint**

`soft_used_by_modulo_var`(`C`, `VARIABLES1`, `VARIABLES2`, `M`)

**Synonym**

`soft_used_by_modulo`.

**Arguments**

| | | |
|---|---|---|
| `C` | : | `dvar` |
| `VARIABLES1` | : | `collection`(`var`−`dvar`) |
| `VARIABLES2` | : | `collection`(`var`−`dvar`) |
| `M` | : | `int` |

**Restrictions**

$\texttt{C} \geq 0$
$\texttt{C} \leq |\texttt{VARIABLES2}|$
$|\texttt{VARIABLES1}| \geq |\texttt{VARIABLES2}|$
`required`(`VARIABLES1`, `var`)
`required`(`VARIABLES2`, `var`)
$\texttt{M} > 0$

**Purpose**

For each integer $R$ in $[0, \texttt{M}-1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of `VARIABLES1` (respectively `VARIABLES2`) that have $R$ as a rest when divided by `M`. `C` is the minimum number of values to change in the `VARIABLES1` and `VARIABLES2` collections so that for all $R$ in $[0, \texttt{M}-1]$ we have $N2_R > 0 \Rightarrow N1_R \geq N2_R$.

**Example**

$(2, \langle 9, 1, 1, 8, 8\rangle, \langle 9, 9, 9, 1\rangle, 3)$

In the example, the values of the collections $\langle 9, 1, 1, 8, 8\rangle$ and $\langle 9, 9, 9, 1\rangle$ are respectively associated with the equivalence classes $9 \bmod 3 = 0$, $1 \bmod 3 = 1$, $1 \bmod 3 = 1$, $8 \bmod 3 = 2$, $8 \bmod 3 = 2$ and $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $1 \bmod 3 = 1$. Since there is a correspondence between two pairs of equivalence classes we must unset at least $4 - 2$ items ($4$ is the number of items of the `VARIABLES2` collection). Consequently, the `soft_used_by_modulo_var` constraint holds since its first argument `C` is set to $4 - 2$.

**Typical**

$\texttt{C} > 0$
$|\texttt{VARIABLES1}| > 1$
$|\texttt{VARIABLES2}| > 1$
`range`(`VARIABLES1.var`) $> 1$
`range`(`VARIABLES2.var`) $> 1$
$\texttt{M} > 1$
$\texttt{M} <$ `maxval`(`VARIABLES1.var`)
$\texttt{M} <$ `maxval`(`VARIABLES2.var`)

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value $u$ of VARIABLES1.var can be replaced by any other value $v$ such that $v$ is congruent to $u$ modulo M.
- An occurrence of a value $u$ of VARIABLES2.var can be replaced by any other value $v$ such that $v$ is congruent to $u$ modulo M.

**Usage** A soft used_by_modulo constraint.

**See also** **hard version:** used_by_modulo.

**implied by:** soft_same_modulo_var.

**Keywords** **characteristic of a constraint:** modulo.

**constraint arguments:** constraint between two collections of variables.

**constraint type:** soft constraint, relaxation, variable-based violation measure.

| | |
|---|---|
| **Arc input(s)** | `VARIABLES1` `VARIABLES2` |
| **Arc generator** | $PRODUCT \mapsto$ `collection`(`variables1`, `variables2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `variables1.var` mod `M` = `variables2.var` mod `M` |
| **Graph property(ies)** | **NSINK_NSOURCE**= \|`VARIABLES2`\| − `C` |

**Graph model** Parts (A) and (B) of Figure 5.729 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_used_by_modulo_var` constraint holds since the cost 2 corresponds to the difference between the number of variables of `VARIABLES2` and the sum over the different connected components of the minimum number of sources and sinks.

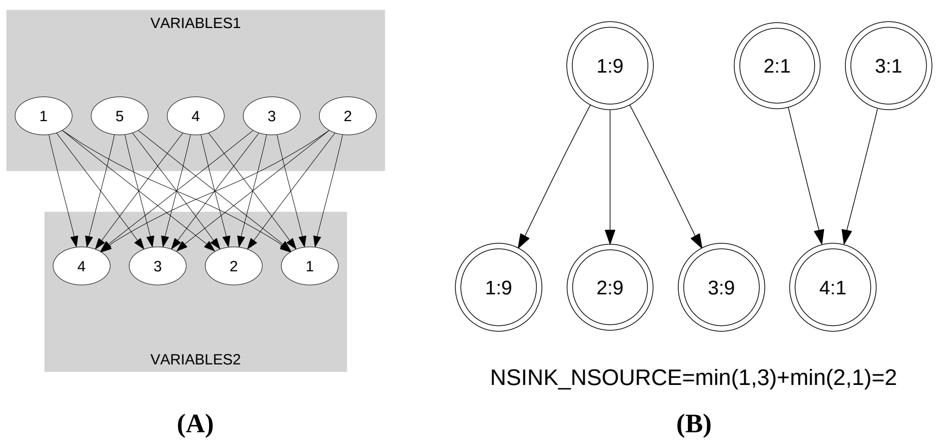



Figure 5.729: Initial and final graph of the `soft_used_by_modulo_var` constraint

## 5.369 soft_used_by_partition_var

DESCRIPTION LINKS GRAPH

**Origin** Derived from used_by_partition.

**Constraint** soft_used_by_partition_var(C, VARIABLES1, VARIABLES2, PARTITIONS)

**Synonym** soft_used_by_partition.

**Type**

VALUES : collection(val−int)

**Arguments**

C : dvar
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

**Restrictions**

$\mathtt{C} \geq 0$
$\mathtt{C} \leq |\mathtt{VARIABLES2}|$
$|\mathtt{VARIABLES1}| \geq |\mathtt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
$|\mathtt{PARTITIONS}| \geq 2$
$|\mathtt{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)

**Purpose**

For each integer $i$ in $[1, |\mathtt{PARTITIONS}|]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that take their value in the $i^{th}$ partition of the collection PARTITIONS. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all $i$ in $[1, |\mathtt{PARTITIONS}|]$ we have $N2_i > 0 \Rightarrow N1_i \geq N2_i$.

**Example**

$$\left(\begin{array}{l} 2, \langle 9, 1, 1, 8, 8\rangle, \\ \langle 9, 9, 9, 1\rangle, \\ \langle \mathtt{p} - \langle 1, 2\rangle, \mathtt{p} - \langle 9\rangle, \mathtt{p} - \langle 7, 8\rangle\rangle \end{array}\right)$$

In the example, the values of the collections $\langle 9,1,1,8,8\rangle$ and $\langle 9,9,9,1\rangle$ are respectively associated with the partitions $\mathtt{p} - \langle 9\rangle$, $\mathtt{p} - \langle 1,2\rangle$, $\mathtt{p} - \langle 1,2\rangle$, $\mathtt{p} - \langle 7,8\rangle$, $\mathtt{p} - \langle 7,8\rangle$ and $\mathtt{p} - \langle 9\rangle$, $\mathtt{p} - \langle 9\rangle$, $\mathtt{p} - \langle 9\rangle$, $\mathtt{p} - \langle 1,2\rangle$. Since there is a correspondence between two pairs of partitions we must unset at least $4 - 2$ items ($4$ is the number of items of the VARIABLES2 collection). Consequently, the soft_used_by_partition_var constraint holds since its first argument C is set to $4 - 2$.

**Typical**

$\texttt{C} > 0$
$|\texttt{VARIABLES1}| > 1$
$|\texttt{VARIABLES2}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$
$|\texttt{VARIABLES1}| > |\texttt{PARTITIONS}|$
$|\texttt{VARIABLES2}| > |\texttt{PARTITIONS}|$

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

**Usage** A soft used_by_partition constraint.

**See also** **hard version:** used_by_partition.

**implied by:** soft_same_partition_var.

**Keywords** **characteristic of a constraint:** partition.

**constraint arguments:** constraint between two collections of variables.

**constraint type:** soft constraint, relaxation, variable-based violation measure.

| | |
|---|---|
| **Arc input(s)** | VARIABLES1 VARIABLES2 |
| **Arc generator** | $PRODUCT \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | in_same_partition(variables1.var, variables2.var, PARTITIONS) |
| **Graph property(ies)** | **NSINK_NSOURCE** $= \vert$VARIABLES2$\vert -$ C |

**Graph model** Parts (A) and (B) of Figure 5.730 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The soft_used_by_partition_var constraint holds since the cost 2 corresponds to the difference between the number of variables of VARIABLES2 and the sum over the different connected components of the minimum number of sources and sinks.

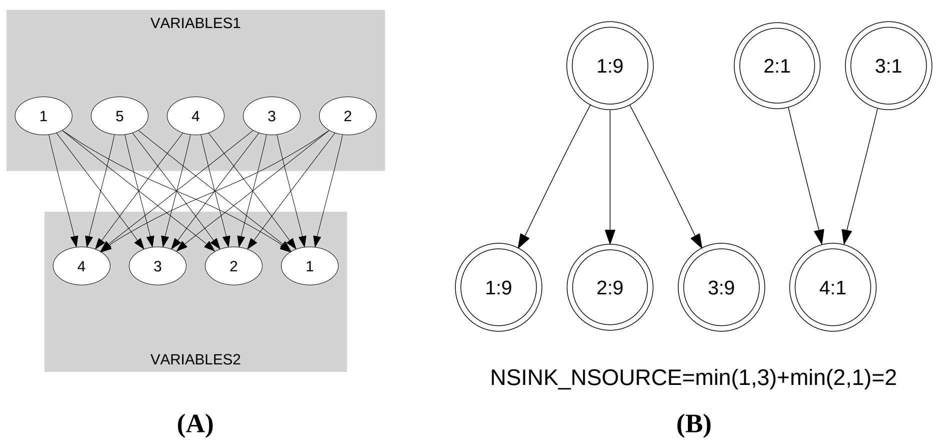



Figure 5.730: Initial and final graph of the soft_used_by_partition_var constraint

## 5.370 soft_used_by_var

DESCRIPTION LINKS GRAPH

**Origin** Derived from used_by

**Constraint** soft_used_by_var(C, VARIABLES1, VARIABLES2)

**Synonym** soft_used_by.

**Arguments**

| | | |
|---|---|---|
| C | : | dvar |
| VARIABLES1 | : | collection(var−dvar) |
| VARIABLES2 | : | collection(var−dvar) |

**Restrictions**

$\mathtt{C} \geq 0$
$\mathtt{C} \leq |\mathtt{VARIABLES2}|$
$|\mathtt{VARIABLES1}| \geq |\mathtt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)

**Purpose** C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that all the values of the variables of collection VARIABLES2 are used by the variables of collection VARIABLES1.

**Example** $(2, \langle 9, 1, 1, 8, 8\rangle, \langle 9, 9, 9, 1\rangle)$

As illustrated by Figure 5.731, there is a correspondence between two pairs of values of the collections $\langle 9, 1, 1, 8, 8\rangle$ and $\langle 9, 9, 9, 1\rangle$. Consequently, we must unset at least $4 - 2$ items (4 is the number of items of the VARIABLES2 collection). The soft_used_by_var constraint holds since its first argument C is set to $4 - 2$.

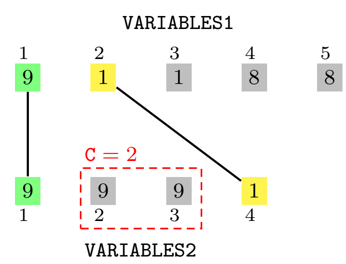



Figure 5.731: Illustration of the partial correspondence between the items of the $\mathtt{VARIABLES2} = \langle 9, 9, 9, 1\rangle$ and of the $\mathtt{VARIABLES1} = \langle 9, 1, 1, 8, 8\rangle$ collections of the **Example** slot, i.e., $\mathtt{C} = 2$ items of the VARIABLES2 or of the VARIABLES1 collections need to be changed in order to cover all elements of VARIABLES2

**Typical**

$\texttt{C} > 0$
$|\texttt{VARIABLES1}| > 1$
$|\texttt{VARIABLES2}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

**Usage** A soft used_by constraint.

**Algorithm** A filtering algorithm achieving arc-consistency is described in [129, 130].

**See also**

**hard version:** used_by.

**implied by:** soft_same_var.

**Keywords**

**constraint arguments:** constraint between two collections of variables.

**constraint type:** soft constraint, relaxation, variable-based violation measure.

**filtering:** bipartite matching.

| | |
|---|---|
| **Arc input(s)** | `VARIABLES1` `VARIABLES2` |
| **Arc generator** | *PRODUCT* $\mapsto$ `collection`(`variables1`, `variables2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `variables1.var` = `variables2.var` |
| **Graph property(ies)** | **NSINK_NSOURCE** = \|`VARIABLES2`\| − `C` |

**Graph model** Parts (A) and (B) of Figure 5.732 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The `soft_used_by_var` constraint holds since the cost 2 corresponds to the difference between the number of variables of `VARIABLES2` and the sum over the different connected components of the minimum number of sources and sinks.

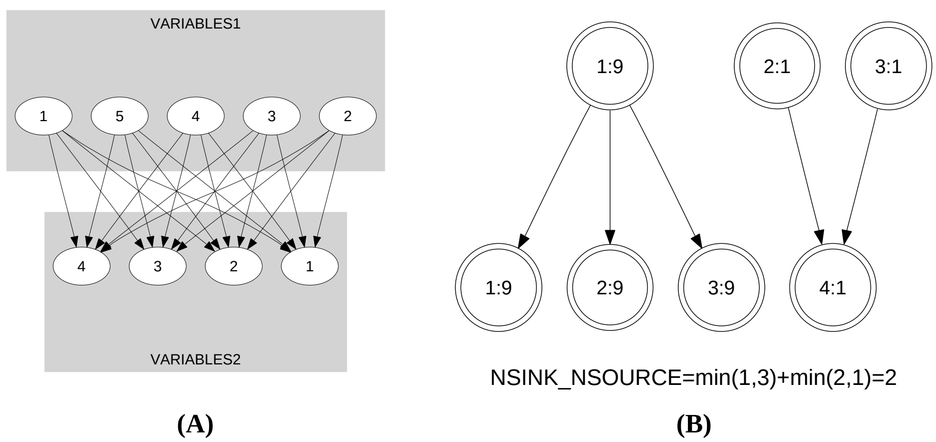



Figure 5.732: Initial and final graph of the `soft_used_by_var` constraint

## 5.371 some_equal

DESCRIPTION LINKS GRAPH

**Origin**

Derived from `alldifferent`

**Constraint**

`some_equal(VARIABLES)`

**Synonyms**

`some_eq`, `not_alldifferent`, `not_alldiff`, `not_alldistinct`, `not_distinct`.

**Argument**

`VARIABLES : collection(var-dvar)`

**Restrictions**

`required(VARIABLES, var)`
$|$`VARIABLES`$| > 1$

**Purpose**

Enforce at least two variables of the collection `VARIABLES` to be assigned the same value.

**Example**

$(\langle 1, 4, 1, 6\rangle)$

The `some_equal` constraint holds since the first and the third variables are both assigned the same value 1.

**Typical**

$|$`VARIABLES`$| > 2$
`nval(VARIABLES.var)` $> 2$

**Symmetries**

- Items of `VARIABLES` are permutable.
- All occurrences of two distinct values of `VARIABLES.var` can be swapped; all occurrences of a value of `VARIABLES.var` can be renamed to any unused value.

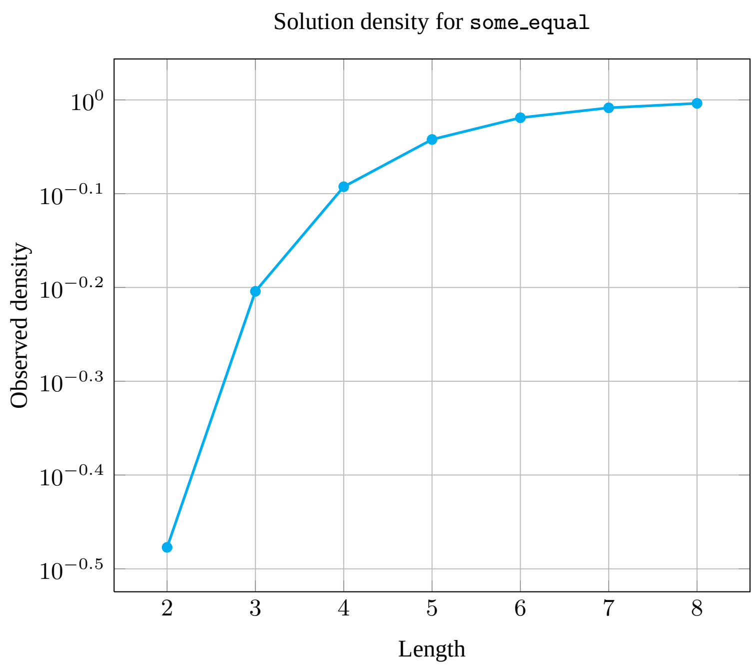
**Arg. properties**

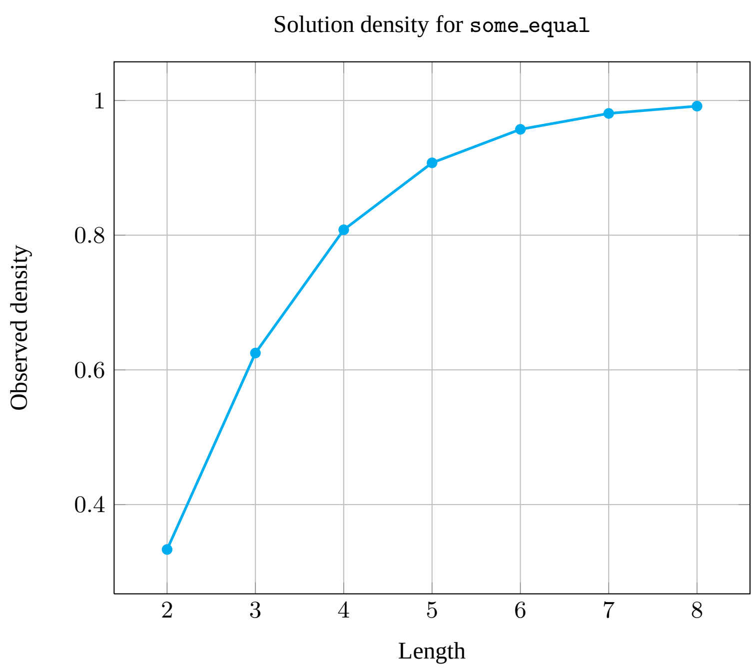
Extensible wrt. `VARIABLES`.

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 3 | 40 | 505 | 7056 | 112609 | 2056832 | 42683841 |

Number of solutions for `some_equal`: domains $0..n$




**Used in** soft_alldifferent_var.

**See also** **negation:** alldifferent.

**Keywords** **characteristic of a constraint:** sort based reformulation.
**constraint type:** value constraint.

| | |
|---|---|
| **Arc input(s)** | VARIABLES |
| **Arc generator** | $CLIQUE(<) \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var $=$ variables2.var |
| **Graph property(ies)** | **NARC** $> 0$ |

**Graph model** We generate a *clique* with an equality constraint between each pair of distinct vertices and state that the number of arcs of the final graph should be strictly greater than 0.

Parts (A) and (B) of Figure 5.733 respectively show the initial and final graph associated with the **Example** slot. The `some_equal` constraint holds since the final graph has at one arc, i.e. two variables are assigned the same value.

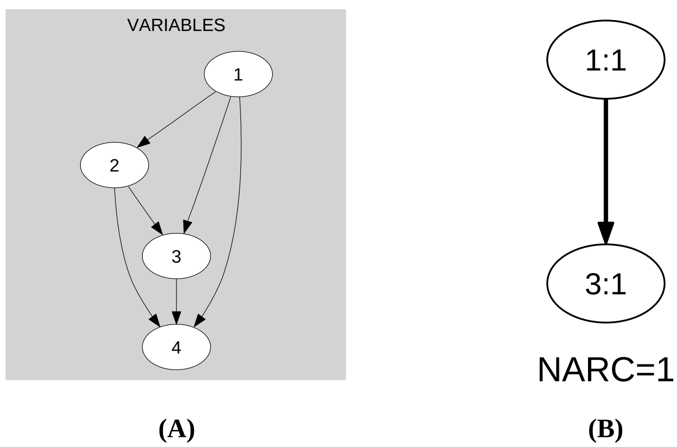



Figure 5.733: Initial and final graph of the `some_equal` constraint

## 5.372 sort

DESCRIPTION LINKS GRAPH

**Origin** [297]

**Constraint** sort(VARIABLES1, VARIABLES2)

**Synonyms** sortedness, sorted, sorting.

**Arguments**

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

**Restrictions**

|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)

**Purpose** First, the variables of the collection VARIABLES2 correspond to a permutation of the variables of VARIABLES1. Second, the variables of VARIABLES2 are sorted in increasing order.

**Example** $(\langle 1, 9, 1, 5, 2, 1\rangle, \langle 1, 1, 1, 2, 5, 9\rangle)$

The sort constraint holds since:

- Values 1, 2, 5 and 9 have the same number of occurrences within both collections $\langle 1, 9, 1, 5, 2, 1\rangle$ and $\langle 1, 1, 1, 2, 5, 9\rangle$. Figure 5.734 illustrates this correspondence.
- The items of collection $\langle 1, 1, 1, 2, 5, 9\rangle$ are sorted in increasing order.

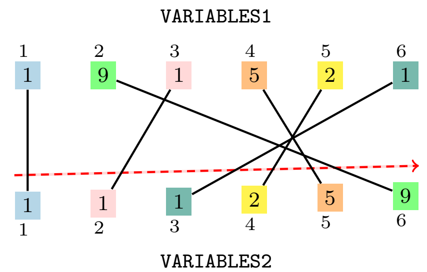



Figure 5.734: Illustration of the correspondence between the items of the VARIABLES1 and of the VARIABLES2 collections of the **Example** slot (note that the items of the VARIABLES2 are sorted in increasing order)

**All solutions** Figure 5.735 gives all solutions to the following non ground instance of the sort constraint: $V_1 \in [2,3]$, $V_2 \in [2,3]$, $V_3 \in [1,2]$, $V_4 \in [4,5]$, $V_5 \in [2,4]$, $S_1 \in [2,3]$, $S_2 \in [2,3]$, $S_3 \in [1,3]$, $S_4 \in [4,5]$, $S_5 \in [2,5]$, sort($\langle V_1, V_2, V_3, V_4, V_5\rangle$, $\langle S_1, S_2, S_3, S_4, S_5\rangle$).

① $(\langle 2, 2, 2, 4, 4\rangle, \langle 2, 2, 2, 4, 4\rangle)$
② $(\langle 2, 2, 2, 5, 4\rangle, \langle 2, 2, 2, 4, 5\rangle)$
③ $(\langle 2, 3, 2, 4, 4\rangle, \langle 2, 2, 3, 4, 4\rangle)$
④ $(\langle 2, 3, 2, 5, 4\rangle, \langle 2, 2, 3, 4, 5\rangle)$
⑤ $(\langle 3, 2, 2, 4, 4\rangle, \langle 2, 2, 3, 4, 4\rangle)$
⑥ $(\langle 3, 2, 2, 5, 4\rangle, \langle 2, 2, 3, 4, 5\rangle)$
⑦ $(\langle 3, 3, 2, 4, 4\rangle, \langle 2, 3, 3, 4, 4\rangle)$
⑧ $(\langle 3, 3, 2, 5, 4\rangle, \langle 2, 3, 3, 4, 5\rangle)$

Figure 5.735: All solutions corresponding to the non ground example of the `sort` constraint of the **All solutions** slot

**Typical**

$|$`VARIABLES1`$| > 1$
`range`(`VARIABLES1.var`) $> 1$

**Symmetries**

- Items of `VARIABLES1` are permutable.
- One and the same constant can be added to the `var` attributes of all items of `VARIABLES1` and `VARIABLES2`.

**Arg. properties**

Functional dependency: `VARIABLES2` determined by `VARIABLES1`.

**Usage**

The main usage of the `sort` constraint, that was not foreseen when the `sort` constraint was invented, is its use in many reformulations. Many constraints involving one or several collections of variables *become much simpler to express when the variables of these collections are sorted*. In addition these reformulations typically have a size that is linear in the number of variables of the original constraint. This justifies why the `sort` constraint is considered to be a core constraint. As illustrative examples of these types of reformulations we successively consider the `alldifferent` and the `same` constraints:

- The `alldifferent`$(\langle v_1, v_2, \ldots, v_n\rangle)$ constraint can be reformulated as the conjunction `sort`$(\langle v_1, v_2, \ldots, v_n\rangle, \langle w_1, w_2, \ldots, w_n\rangle) \wedge$ `strictly_increasing`$(\langle w_1, w_2, \ldots, w_n\rangle)$.
- The `same`$(\langle u_1, u_2, \ldots, u_n\rangle, \langle v_1, v_2, \ldots, v_n\rangle)$ constraint can be reformulated as the conjunction `sort`$(\langle u_1, u_2, \ldots, u_n\rangle, \langle w_1, w_2, \ldots, w_n\rangle) \wedge$ `sort`$(\langle v_1, v_2, \ldots, v_n\rangle, \langle w_1, w_2, \ldots, w_n\rangle)$.

**Remark**

A variant of this constraint called `sort_permutation` was introduced in [449]. In this variant an additional list of domain variables represents the permutation that allows to go from `VARIABLES1` to `VARIABLES2`.

**Algorithm**

[78, 281].

**Systems**

`sorting` in **Choco**, `sorted` in **Gecode**, `sort` in **MiniZinc**, `sorting` in **SICStus**.

**See also**

**generalisation:** `sort_permutation` *(*`PERMUTATION` *parameter added)*.

**implies:** `lex_greatereq`, `same`.

**uses in its reformulation:** `alldifferent`, `same`.

**Keywords**

**characteristic of a constraint:** core, sort.

**combinatorial object:** permutation.

**constraint arguments:** constraint between two collections of variables, pure functional dependency.

**filtering:** bound-consistency.

**modelling:** functional dependency.

| | |
|---|---|
| **Arc input(s)** | `VARIABLES1` `VARIABLES2` |
| **Arc generator** | $PRODUCT \mapsto$ `collection`(`variables1`, `variables2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `variables1.var` = `variables2.var` |
| **Graph property(ies)** | • for all connected components: **NSOURCE**=**NSINK**<br>• **NSOURCE**= \|`VARIABLES1`\|<br>• **NSINK**= \|`VARIABLES2`\| |

| | |
|---|---|
| **Arc input(s)** | `VARIABLES2` |
| **Arc generator** | $PATH \mapsto$ `collection`(`variables1`, `variables2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `variables1.var` $\leq$ `variables2.var` |
| **Graph property(ies)** | **NARC**= \|`VARIABLES2`\| $-$ 1 |

**Graph model** Parts (A) and (B) of Figure 5.736 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. Since it uses the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of this final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. The `sort` constraint holds since:

- Each connected component of the final graph of the first graph constraint has the same number of sources and of sinks.
- The number of sources of the final graph of the first graph constraint is equal to |`VARIABLES1`|.
- The number of sinks of the final graph of the first graph constraint is equal to |`VARIABLES2`|.
- Finally the second graph constraint holds also since its corresponding final graph contains exactly |`VARIABLES1` $-$ 1| arcs: all the inequalities constraints between consecutive variables of `VARIABLES2` holds.

**Signature** Consider the first graph constraint. Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the $PRODUCT$ arc generator on the collections `VARIABLES1` and `VARIABLES2`, we have that the maximum number of sources and sinks of the final graph is respectively equal to |`VARIABLES1`| and |`VARIABLES2`|. Therefore we can rewrite **NSOURCE** = |`VARIABLES1`| to **NSOURCE** $\geq$ |`VARIABLES1`| and simplify $\underline{\overline{\mathbf{NSOURCE}}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite **NSINK** = |`VARIABLES2`| to **NSINK** $\geq$ |`VARIABLES2`| and simplify $\underline{\overline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.

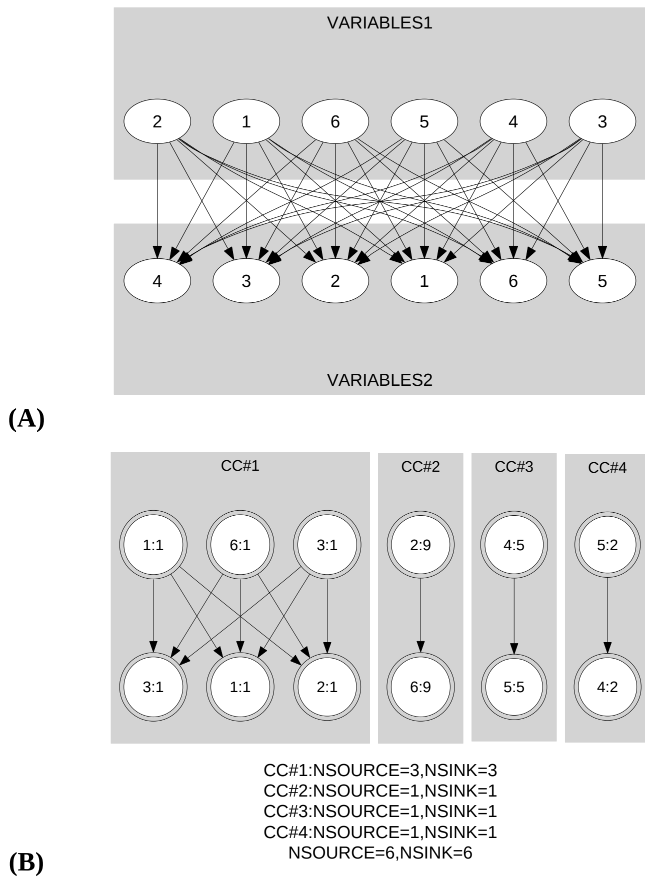



Figure 5.736: Initial and final graph of the `sort` constraint

Consider now the second graph constraint. Since we use the $PATH$ arc generator with an arity of 2 on the `VARIABLES2` collection, the maximum number of arcs of the final graph is equal to $|\texttt{VARIABLES2}| - 1$. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{VARIABLES2}| - 1$ to $\mathbf{NARC} \geq |\texttt{VARIABLES2}| - 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

**Quiz**

**EXERCISE 1 (checking whether a ground instance holds or not)**[a]

**A.** Does the constraint $\texttt{sort}(\langle 1, 0, 0, 1\rangle, \langle 0, 0, 1\rangle)$ hold?

**B.** Does the constraint $\texttt{sort}(\langle 3, 5, 3, 1\rangle, \langle 1, 3, 5\rangle)$ hold?

**C.** Does the constraint $\texttt{sort}(\langle 2, 4, 2, 2, 4\rangle, \langle 2, 2, 2, 4, 4\rangle)$ hold?

**D.** Does the constraint $\texttt{sort}(\langle 2, 4, 2, 2, 4\rangle, \langle 4, 4, 2, 2, 2\rangle)$ hold?

[a] Hint: go back to the definition of `sort`.

**EXERCISE 2 (finding all solutions)**[a]

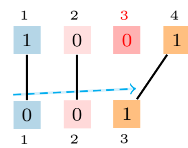
Give all the solutions to the constraint:

$$\left\{\begin{array}{l}
X_1 \in [2,4], \quad X_2 \in [2,3], \quad X_3 \in [0,5], \quad X_4 \in [6,8], \quad X_5 \in [3,6], \\
Y_1 \in [3,4], \quad Y_2 \in [2,3], \quad Y_3 \in [0,5], \quad Y_4 \in [6,8], \quad Y_5 \in [3,6], \\
\\
\texttt{sort}\left(\begin{array}{lllll} \langle X_1, & X_2, & X_3, & X_4, & X_5\rangle, \\ \langle Y_1, & Y_2, & Y_3, & Y_4, & Y_5\rangle \end{array}\right).
\end{array}\right.$$

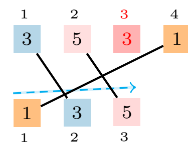
[a] Hint: first filter the bounds of the variables of the second argument wrt the chain of precedences; second, since the second argument can be computed from the first one, focus on the variables of the first argument and enumerate solutions in lexicographic order.

**SOLUTION TO EXERCISE 1**

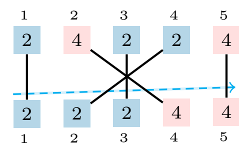
**A.** *No, since* $\langle 1, 0, 0, 1\rangle$ *and* $\langle 0, 0, 1\rangle$ *do not have the same number of elements.*



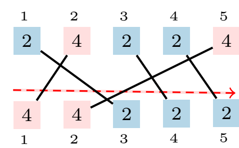

**B.** *No, since* $\langle 3, 5, 3, 1\rangle$ *and* $\langle 1, 3, 5\rangle$ *do not have the same number of elements.*




**C.** *Yes, since* $\langle 2, 2, 2, 4, 4\rangle$ *is a permutation of* $\langle 2, 4, 2, 2, 4\rangle$ *and since the elements* $2, 2, 2, 4, 4$ *are sorted in non-decreasing order.*




**D.** *No, since the elements of* $\langle 4, 4, 2, 2, 2\rangle$ *are not sorted in non-decreasing order.*



## SOLUTION TO EXERCISE 2

**the four solutions**

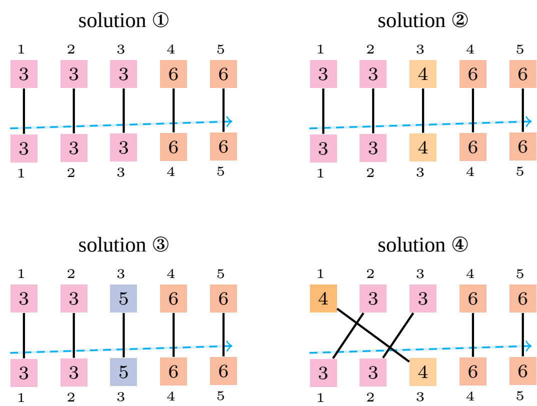
$$\langle X_1, X_2, X_3, X_4, X_5\rangle, \langle Y_1, Y_2, Y_3, Y_4, Y_5\rangle$$

① $(\langle 3, 3, 3, 6, 6\rangle, \ \langle 3, 3, 3, 6, 6\rangle)$

② $(\langle 3, 3, 4, 6, 6\rangle, \ \langle 3, 3, 4, 6, 6\rangle)$

③ $(\langle 3, 3, 5, 6, 6\rangle, \ \langle 3, 3, 5, 6, 6\rangle)$

④ $(\langle 4, 3, 3, 6, 6\rangle, \ \langle 3, 3, 4, 6, 6\rangle)$



## 5.373 sort_permutation

DESCRIPTION LINKS GRAPH

**Origin** [449]

**Constraint** sort_permutation(FROM, PERMUTATION, TO)

**Usual name** sort

**Synonyms** extended_sortedness, sortedness, sorted, sorting.

**Arguments**

FROM : collection(var−dvar)
PERMUTATION : collection(var−dvar)
TO : collection(var−dvar)

**Restrictions**

|PERMUTATION| = |FROM|
|PERMUTATION| = |TO|
PERMUTATION.var $\geq 1$
PERMUTATION.var $\leq$ |PERMUTATION|
alldifferent(PERMUTATION)
required(FROM, var)
required(PERMUTATION, var)
required(TO, var)

**Purpose** The variables of collection FROM correspond to the variables of collection TO according to the permutation PERMUTATION (i.e., FROM$[i]$.var = TO[PERMUTATION$[i]$.var].var). The variables of collection TO are also sorted in increasing order.

**Example** $(\langle 1, 9, 1, 5, 2, 1\rangle, \langle 1, 6, 3, 5, 4, 2\rangle, \langle 1, 1, 1, 2, 5, 9\rangle)$

The sort_permutation constraint holds since:

- – The first item FROM$[1]$.var $= 1$ of collection FROM corresponds to the PERMUTATION$[1]$.var $= 1^{th}$ item of collection TO.
  – The second item FROM$[2]$.var $= 9$ of collection FROM corresponds to the PERMUTATION$[2]$.var $= 6^{th}$ item of collection TO.
  – The third item FROM$[3]$.var $= 1$ of collection FROM corresponds to the PERMUTATION$[3]$.var $= 3^{th}$ item of collection TO.
  – The fourth item FROM$[4]$.var $= 5$ of collection FROM corresponds to the PERMUTATION$[4]$.var $= 5^{th}$ item of collection TO.
  – The fifth item FROM$[5]$.var $= 2$ of collection FROM corresponds to the PERMUTATION$[5]$.var $= 4^{th}$ item of collection TO.
  – The sixth item FROM$[6]$.var $= 1$ of collection FROM corresponds to the PERMUTATION$[6]$.var $= 2^{th}$ item of collection TO.
- The items of collection TO $= \langle 1, 1, 1, 2, 5, 9\rangle$ are sorted in increasing order.

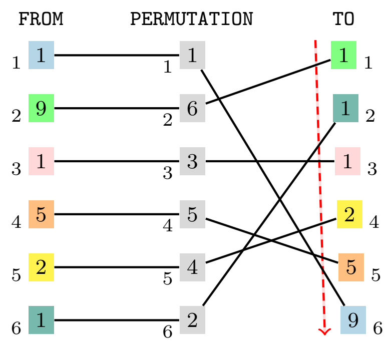



Figure 5.737: Illustration of the correspondence between the items of the FROM and the TO collections according to the permutation defined by the items of the PERMUTATION collection of the **Example** slot (note that the items of the TO collection are sorted in increasing order)

**Typical**

$|\mathtt{FROM}| > 1$
$\mathtt{range}(\mathtt{FROM.var}) > 1$
$\mathtt{lex\_different}(\mathtt{FROM}, \mathtt{TO})$

**Symmetry**

One and the same constant can be added to the var attributes of all items of FROM and TO.

**Arg. properties**

- Functional dependency: TO determined by FROM.
- Functional dependency: PERMUTATION determined by FROM and TO.

**Remark**

This constraint is referenced under the name sorting in **SICStus Prolog**.

**Algorithm**

[449].

**Reformulation**

Let $n$ denote the number of variables in the collection FROM. The sort_permutation constraint can be reformulated as a conjunction of the form:

$\mathtt{element}(\mathtt{PERMUTATION}[1],\ \mathtt{FROM},\ \mathtt{TO}[1])$,
$\mathtt{element}(\mathtt{PERMUTATION}[2],\ \mathtt{FROM},\ \mathtt{TO}[2])$,
...
$\mathtt{element}(\mathtt{PERMUTATION}[n],\ \mathtt{FROM},\ \mathtt{TO}[n])$,
$\mathtt{alldifferent}(\mathtt{PERMUTATION})$,
$\mathtt{increasing}(\mathtt{TO})$.

To enhance the previous model, the following necessary condition was proposed by P. Schaus. $\forall i \in [1, n] : \sum_{j=1}^{j=n} (\mathtt{FROM}[j] < \mathtt{TO}[i]) \leq i - 1$ (i.e., at most $i - 1$ variables of the collection FROM are assigned a value strictly less than $\mathtt{TO}[i]$). Similarly, we have that $\forall i \in [1, n] : \sum_{j=1}^{j=n} (\mathtt{FROM}[j] > \mathtt{TO}[i]) \geq n - i$ (i.e., at most $n - i$ variables of the collection FROM are assigned a value are strictly greater than $\mathtt{TO}[i]$).

**Systems**

sorted in **Gecode**, sorting in **SICStus**.

**See also**

**common keyword:** `order` *(sort, permutation)*.

**implies:** `correspondence`.

**specialisation:** `sort` *(*`PERMUTATION` *parameter removed)*.

**used in reformulation:** `alldifferent`, `element`, `increasing`.

**Keywords**

**characteristic of a constraint:** sort, derived collection.

**combinatorial object:** permutation.

**constraint arguments:** constraint between three collections of variables.

**modelling:** functional dependency.

**Derived Collection**

$\texttt{col}\left(\begin{array}{l}\texttt{FROM\_PERMUTATION}-\texttt{collection}(\texttt{var}-\texttt{dvar},\texttt{ind}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{FROM.var},\texttt{ind}-\texttt{PERMUTATION.var})]\end{array}\right)$

| | |
|---|---|
| **Arc input(s)** | FROM_PERMUTATION TO |
| **Arc generator** | $PRODUCT \mapsto$ collection(from_permutation, to) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • from_permutation.var = to.var<br>• from_permutation.ind = to.key |
| **Graph property(ies)** | **NARC**= \|PERMUTATION\| |

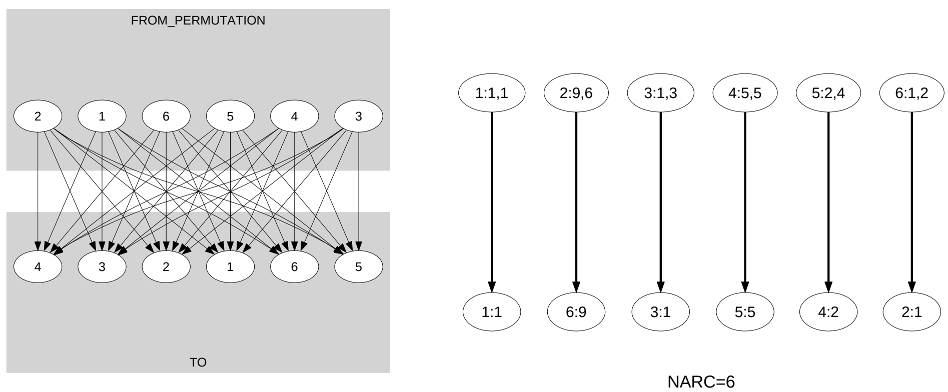
| | |
|---|---|
| **Arc input(s)** | TO |
| **Arc generator** | $PATH \mapsto$ collection(to1, to2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | to1.var $\leq$ to2.var |
| **Graph property(ies)** | **NARC**= \|TO\| $-$ 1 |

**Graph model** Parts (A) and (B) of Figure 5.738 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. In both graphs the source vertices correspond to the items of the derived collection FROM_PERMUTATION, while the sink vertices correspond to the items of the TO collection. Since the first graph constraint uses the **NARC** graph property, the arcs of its final graph are stressed in bold. The sort_permutation constraint holds since:

- The first graph constraint holds since its final graph contains exactly PERMUTATION arcs.
- Finally the second graph constraint holds also since its corresponding final graph contains exactly |PERMUTATION $-$ 1| arcs: all the inequalities constraints between consecutive variables of TO holds.

**Signature** Consider the first graph constraint where we use the $PRODUCT$ arc generator. Since all the key attributes of the TO collection are distinct, and because of the second condition from_permutation.ind = to.key of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to |PERMUTATION|. So we can rewrite the graph property $\mathbf{NARC}$ = |PERMUTATION| to $\mathbf{NARC} \geq$ |PERMUTATION| and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Consider now the second graph constraint. Since we use the $PATH$ arc generator with an arity of 2 on the TO collection, the maximum number of arcs of the corresponding final graph is equal to |TO| $-$ 1. Therefore we can rewrite $\mathbf{NARC}$ = |TO| $-$ 1 to $\mathbf{NARC} \geq$ |TO| $-$ 1 and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.




**(A)** **(B)**

Figure 5.738: Initial and final graph of the `sort_permutation` constraint

## 5.374 stable_compatibility

DESCRIPTION LINKS GRAPH

**Origin** P. Flener, [43]

**Constraint** `stable_compatibility(NODES)`

**Argument**

$$\text{NODES} : \text{collection}\begin{pmatrix}\texttt{index}-\texttt{int},\\ \texttt{father}-\texttt{dvar},\\ \texttt{prec}-\texttt{sint},\\ \texttt{inc}-\texttt{sint}\end{pmatrix}$$

**Restrictions**

required(NODES, [index, father, prec, inc])
NODES.index $\geq 1$
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.father $\geq 1$
NODES.father $\leq$ |NODES|
NODES.prec $\geq 1$
NODES.prec $\leq$ |NODES|
NODES.inc $\geq 1$
NODES.inc $\leq$ |NODES|
NODES.inc $>$ NODES.index

**Purpose**

Enforce the construction of a *stably compatible* supertree that is compatible with several given trees. The notion of stable compatibility and its context are detailed in the **Usage** slot.

**Example**

$$\left\langle\begin{pmatrix}
\texttt{index}-1 & \texttt{father}-4 & \texttt{prec}-\{11,12\} & \texttt{inc}-\emptyset,\\
\texttt{index}-2 & \texttt{father}-3 & \texttt{prec}-\{8,9\} & \texttt{inc}-\emptyset,\\
\texttt{index}-3 & \texttt{father}-4 & \texttt{prec}-\{2,10\} & \texttt{inc}-\emptyset,\\
\texttt{index}-4 & \texttt{father}-5 & \texttt{prec}-\{1,3\} & \texttt{inc}-\emptyset,\\
\texttt{index}-5 & \texttt{father}-7 & \texttt{prec}-\{4,13\} & \texttt{inc}-\emptyset,\\
\texttt{index}-6 & \texttt{father}-2 & \texttt{prec}-\{8,14\} & \texttt{inc}-\emptyset,\\
\texttt{index}-7 & \texttt{father}-7 & \texttt{prec}-\{6,13\} & \texttt{inc}-\emptyset,\\
\texttt{index}-8 & \texttt{father}-6 & \texttt{prec}-\emptyset & \texttt{inc}-\{9,10,11,12,13,14\},\\
\texttt{index}-9 & \texttt{father}-2 & \texttt{prec}-\emptyset & \texttt{inc}-\{10,11,12,13\},\\
\texttt{index}-10 & \texttt{father}-3 & \texttt{prec}-\emptyset & \texttt{inc}-\{11,12,13\},\\
\texttt{index}-11 & \texttt{father}-1 & \texttt{prec}-\emptyset & \texttt{inc}-\{12,13\},\\
\texttt{index}-12 & \texttt{father}-1 & \texttt{prec}-\emptyset & \texttt{inc}-\{13\},\\
\texttt{index}-13 & \texttt{father}-5 & \texttt{prec}-\emptyset & \texttt{inc}-\{14\},\\
\texttt{index}-14 & \texttt{father}-6 & \texttt{prec}-\emptyset & \texttt{inc}-\emptyset
\end{pmatrix}\right\rangle$$

Figure 5.739 shows the two trees we want to merge. Note that the leaves a and f occur in both trees.

The left part of Figure 5.740 gives one way to merge the two previous trees. This solution corresponds to the ground instance provided by the example. Note that there exist 7 other ways to merge these two trees. They are respectively depicted by Figures 5.740 to 5.743.

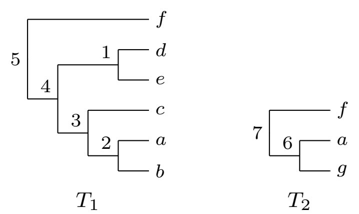



Figure 5.739: The two trees to merge

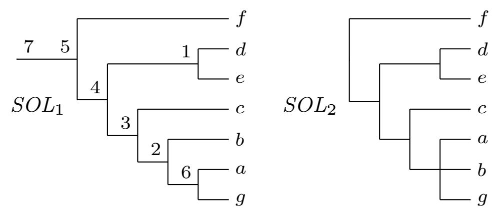



Figure 5.740: First solution (corresponding to the ground instance of the example) and second solution on how to merge the two trees $T_1$ and $T_2$ of Figure 5.739

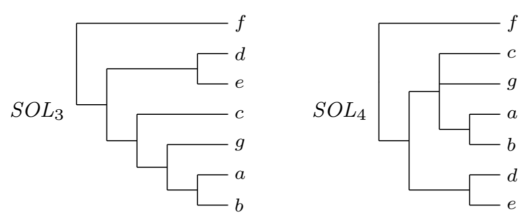



Figure 5.741: Third and fourth solutions on how to merge the two trees $T_1$ and $T_2$ of Figure 5.739

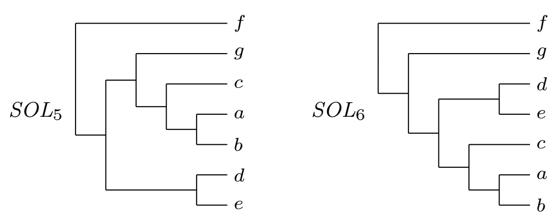



Figure 5.742: Fifth and sixth solutions on how to merge the two trees $T_1$ and $T_2$ of Figure 5.739

| | |
|---|---|
| **Typical** | $\vert$NODES$\vert > 2$<br>range(NODES.father) $> 1$ |
| **Symmetry** | Items of NODES are permutable. |

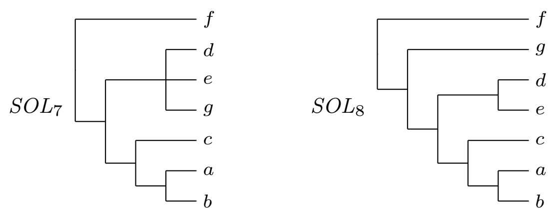



Figure 5.743: Seventh and eight solutions on how to merge the two trees $T_1$ and $T_2$ of Figure 5.739

**Usage**

One objective of phylogeny is to construct the genealogy of the species, called the *tree of life*, whose leaves represent the contemporary species and whose internal nodes represent extinct species that are not necessarily named. An important problem in phylogeny is the construction of a supertree [76] that is compatible with several given trees. There are several definitions of tree compatibility in the literature:

- A tree $\mathcal{T}$ is *strongly compatible* with a tree $\mathcal{T}'$ if $\mathcal{T}'$ is topologically equivalent to a subtree $\mathcal{T}$ that respects the node labelling. [294]
- A tree $\mathcal{T}$ is *weakly compatible* with a tree $\mathcal{T}'$ if $\mathcal{T}'$ can be obtained from $\mathcal{T}$ by a series of arc contractions. [398]
- A tree $\mathcal{T}$ is *stably compatible* with a set $\mathcal{S}$ of trees if $\mathcal{T}$ is weakly compatible with each tree in $\mathcal{S}$ and each internal node of $\mathcal{T}$ can be labelled by at least one corresponding internal node of some tree in $\mathcal{S}$.

For the supertree problem, strong and weak compatibility coincide if and only if all the given trees are binary [294]. The existence of solutions is not lost when restricting weak compatibility to stable compatibility.

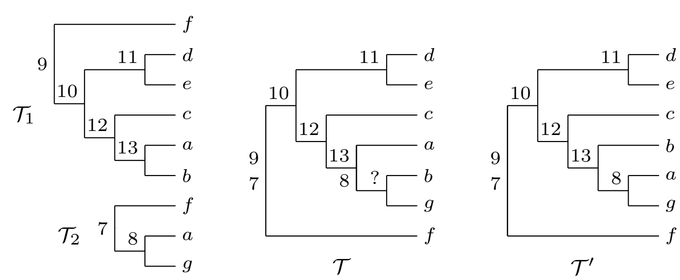



Figure 5.744: Supertree problem instance and two of its solutions

For example, the trees $\mathcal{T}_1$ and $\mathcal{T}_2$ of Figure 5.744 have $\mathcal{T}$ and $\mathcal{T}'$ as supertrees under both weak and strong compatibility. As shown, all the internal nodes of $\mathcal{T}'$ can be labelled by corresponding internal nodes of the two given trees, but this is not the case for the father of $b$ and $g$ in $\mathcal{T}$. Hence $\mathcal{T}$ and four other such supertrees are debatable because they speculate about the existence of extinct species that were not in any of the given trees. Consider also the three small trees in Figure 5.745: $\mathcal{T}_3$ and $\mathcal{T}_4$ have $\mathcal{T}_4$ as a supertree under weak compatibility, as it suffices to contract the arc $(3, 2)$ to get $\mathcal{T}_3$ from $\mathcal{T}_4$. However, $\mathcal{T}_3$ and

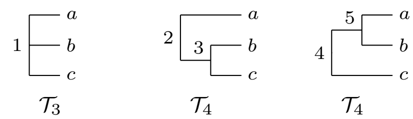



Figure 5.745: Three small phylogenetic trees

$\mathcal{T}_4$ have no supertree under strong compatibility, as the most recent common ancestor of $b$ and $c$, denoted by $mrca(b, c)$, is the same as $mrca(a, b)$ in $\mathcal{T}_3$, namely $1$, but not the same in $\mathcal{T}_4$, as $mrca(b, c) = 3$ is an evolutionary descendant of $mrca(a, b) = 2$. Also, $\mathcal{T}_4$ and $\mathcal{T}_5$ have neither weakly nor strongly compatible supertrees.

Under strong compatibility, a first supertree algorithm was given in [4], with an application for database management systems; it takes $O(l^2)$ time, where $l$ is the number of leaves in the given trees. Derived algorithms have emerged from phylogeny, for instance *One-Tree* [294]. The first constraint program was proposed in [191], using standard, non-global constraints. Under weak compatibility, a phylogenetic supertree algorithm can be found in [398] for instance. Under stable compatibility, the algorithm from computational linguistics of [79] has supertree construction as special case.

**See also** **root concept:** `tree`.

**Keywords** **application area:** bioinformatics, phylogeny.

**constraint type:** graph constraint.

**final graph structure:** tree.

| | |
|---|---|
| **Arc input(s)** | `NODES` |
| **Arc generator** | $CLIQUE \mapsto$ `collection`(`nodes1`, `nodes2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `nodes1.father` $=$ `nodes2.index` |
| **Graph property(ies)** | • **MAX_NSCC** $\leq 1$<br>• **NCC** $= 1$<br>• **MAX_ID** $\leq 2$<br>• **PATH_FROM_TO**(`index`, `index`, `prec`) $= 1$<br>• **PATH_FROM_TO**(`index`, `index`, `inc`) $= 0$<br>• **PATH_FROM_TO**(`index`, `inc`, `index`) $= 0$ |

**Graph model** To each distinct leave (i.e., each species) of the trees to merge corresponds a vertex of the initial graph. To each internal vertex of the trees to merge corresponds also a vertex of the initial graph. Each vertex of the initial graph has the following attributes:

- An *index* corresponding to a unique identifier.
- A *father* corresponding to the father of the vertex in the final tree. Since the leaves of the trees to merge must remain leaves we remove the index value of all the leaves from all the father variables.
- A *set of precedence constraints* corresponding to all the arcs of the trees to merge.
- A *set of incomparability constraints* corresponding to the incomparable vertices of each tree to merge.

The arc constraint describes the fact that we link a vertex to its father variable. Finally we use the following six graph properties on our final graph:

- The first graph property **MAX_NSCC** $\leq 1$ enforces the fact that the size of the largest strongly connected component does not exceed one. This avoid having circuits containing more than one vertex. In fact the root of the merged tree is a strongly connected component with a single vertex.
- The second graph property **NCC** $= 1$ imposes having only a single tree.
- The third graph property **PATH_FROM_TO**(`index`, `index`, `prec`) $= 1$ enforces for each vertex $i$ a set of precedence constraints; for each vertex $j$ of the precedence set there is a path from $i$ to $j$ in the final graph.
- The fourth graph property **MAX_ID** $\leq 2$ enforces that the number of predecessors (i.e., arcs from a vertex to itself are not counted) of each vertex does not exceed 2 (i.e., the final graph is a binary tree).
- The fifth and sixth graph properties **PATH_FROM_TO**(`index`, `index`, `inc`) $=$ $0$ and **PATH_FROM_TO**(`index`, `inc`, `index`) $= 0$ enforces for each vertex $i$ a set of incomparability constraints; for each vertex $j$ of the incomparability set there is neither a path from $i$ to $j$, nor a path from $j$ to $i$.

Figures 5.746 and 5.747 respectively show the precedence and the incomparability graphs associated with the **Example** slot. As it contains too many arcs the initial graph is not shown. Figures 5.740 shows the first solution satisfying all the precedence and incomparability constraints.

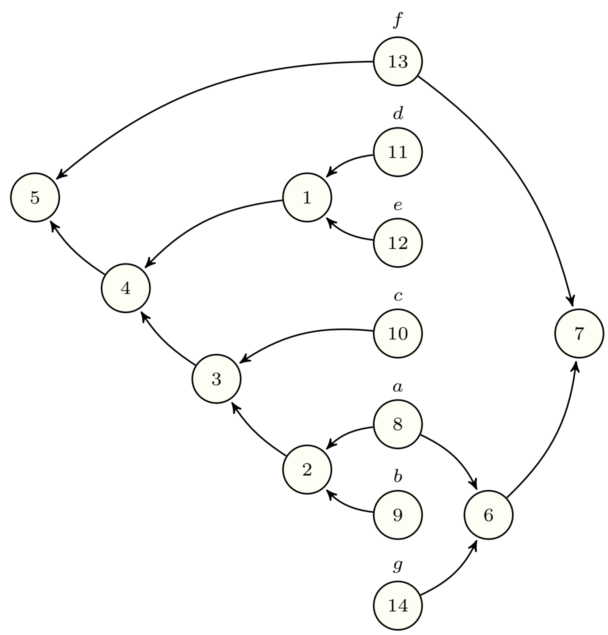



Figure 5.746: Precedence graph associated with the two trees to merge described by Figure 5.739

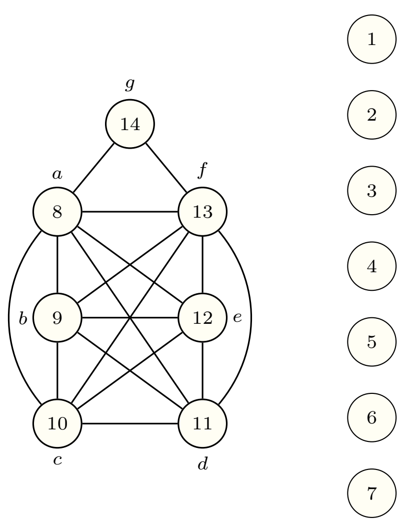



Figure 5.747: Incomparability graph associated with the two trees to merge described by Figure 5.739; the two cliques respectively correspond to the leaves of the two trees to merge.

## 5.375 stage_element

DESCRIPTION LINKS GRAPH AUTOMATON

**Origin**

Choco, derived from element.

**Constraint**

stage_element(ITEM, TABLE)

**Usual name**

stage_elt

**Synonym**

stage_elem.

**Arguments**

$$\begin{array}{lll} \texttt{ITEM} & : & \texttt{collection}(\texttt{index}-\texttt{dvar}, \texttt{value}-\texttt{dvar}) \\ \texttt{TABLE} & : & \texttt{collection}(\texttt{low}-\texttt{int}, \texttt{up}-\texttt{int}, \texttt{value}-\texttt{int}) \end{array}$$

**Restrictions**

$$\begin{array}{l} \texttt{required}(\texttt{ITEM}, [\texttt{index}, \texttt{value}]) \\ |\texttt{ITEM}| = 1 \\ |\texttt{TABLE}| > 0 \\ \texttt{required}(\texttt{TABLE}, [\texttt{low}, \texttt{up}, \texttt{value}]) \\ \texttt{TABLE.low} \leq \texttt{TABLE.up} \\ \texttt{increasing\_seq}(\texttt{TABLE}, [\texttt{low}]) \end{array}$$

**Purpose**

Let $\texttt{low}_i$, $\texttt{up}_i$ and $\texttt{value}_i$ respectively denote the values of the low, up and value attributes of the $i^{th}$ item of the TABLE collection. First we have that: $\texttt{low}_i \leq \texttt{up}_i$ and $\texttt{up}_i + 1 = \texttt{low}_{i+1}$.
Second, the stage_element constraint forces the following equivalence:
$\texttt{low}_i \leq \texttt{ITEM.index} \wedge \texttt{ITEM.index} \leq \texttt{up}_i \Leftrightarrow \texttt{ITEM.value} = \texttt{value}_i$.

**Example**

$$\left(\begin{array}{l} \langle \texttt{index}-5\ \texttt{value}-6 \rangle, \\ \left\langle \begin{array}{lll} \texttt{low}-3 & \texttt{up}-7 & \texttt{value}-6, \\ \texttt{low}-8 & \texttt{up}-8 & \texttt{value}-8, \\ \texttt{low}-9 & \texttt{up}-14 & \texttt{value}-2, \\ \texttt{low}-15 & \texttt{up}-19 & \texttt{value}-9 \end{array} \right\rangle \end{array}\right)$$

Figure 5.748 depicts the function associated with the items of the TABLE collection. The stage_element constraint holds since:

- The value of ITEM[1].index is located between the values of the low and up attributes of the first item of the TABLE collection (i.e., $5 \in [3, 7]$).
- The value of ITEM[1].value corresponds to the value attribute of the first item of the TABLE collection (i.e., 6).

**Typical**

$$\begin{array}{l} |\texttt{TABLE}| > 1 \\ \texttt{range}(\texttt{TABLE.value}) > 1 \\ \texttt{TABLE.low} < \texttt{TABLE.up} \end{array}$$

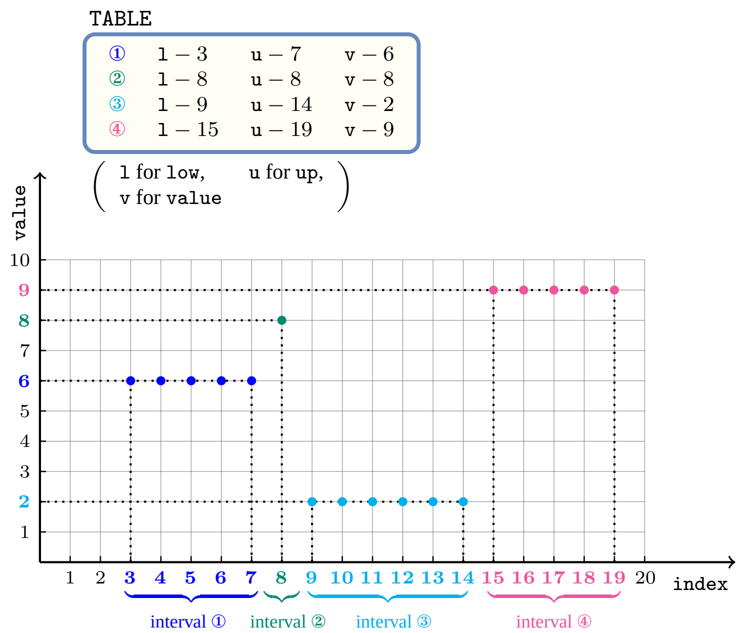



Figure 5.748: Function defined on four intervals ①, ②, ③ and ④ associated with the `TABLE` collection of the **Example** slot for linking the `index` and `value` attributes of the `ITEM` collection

**Symmetry**

All occurrences of two distinct values in `ITEM.value` or `TABLE.value` can be swapped; all occurrences of a value in `ITEM.value` or `TABLE.value` can be renamed to any unused value.

**Arg. properties**

- Functional dependency: `ITEM.value` determined by `ITEM.index` and `TABLE`.
- Suffix-extensible wrt. `TABLE`.

**See also**

**common keyword:** `elem`, `element` (*data constraint*).

**Keywords**

**characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

**constraint arguments:** binary constraint, pure functional dependency.

**constraint network structure:** centered cyclic(2) constraint network(1).

**constraint type:** data constraint.

**filtering:** arc-consistency.

**modelling:** table, functional dependency.

| | |
|---|---|
| **Arc input(s)** | TABLE |
| **Arc generator** | $PATH \mapsto$ `collection`(`table1`, `table2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • `table1.low` $\leq$ `table1.up`<br>• `table1.up` $+ 1 =$ `table2.low`<br>• `table2.low` $\leq$ `table2.up` |
| **Graph property(ies)** | $\mathbf{NARC} = \lvert$TABLE$\rvert - 1$ |

| | |
|---|---|
| **Arc input(s)** | ITEM TABLE |
| **Arc generator** | $PRODUCT \mapsto$ `collection`(`item`, `table`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • `item.index` $\geq$ `table.low`<br>• `item.index` $\leq$ `table.up`<br>• `item.value` $=$ `table.value` |
| **Graph property(ies)** | $\mathbf{NARC} = 1$ |

**Graph model** The first graph constraint models the restrictions on the `low` and `up` attributes of the TABLE collection, while the second graph constraint is similar to the one used for defining the `element` constraint.

Parts (A) and (B) of Figure 5.749 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

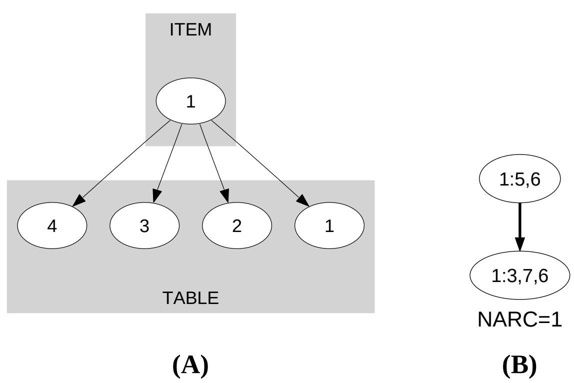



Figure 5.749: Initial and final graph of the `stage_element` constraint

**Automaton** Figure 5.750 depicts the automaton associated with the `stage_element` constraint. Let `INDEX` and `VALUE` respectively be the `index` and the `value` attributes of the unique item of the `ITEM` collection. Let $\texttt{LOW}_i$, $\texttt{UP}_i$ and $\texttt{VALUE}_i$ respectively be the `low`, the `up` and the `value` attributes of the $i^{th}$ item of the `TABLE` collection. To each quintuple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{LOW}_i, \texttt{UP}_i, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable $S_i$ as well as the following signature constraint: $((\texttt{LOW}_i \leq \texttt{INDEX}) \wedge (\texttt{INDEX} \leq \texttt{UP}_i) \wedge (\texttt{VALUE} = \texttt{VALUE}_i)) \Leftrightarrow S_i$.

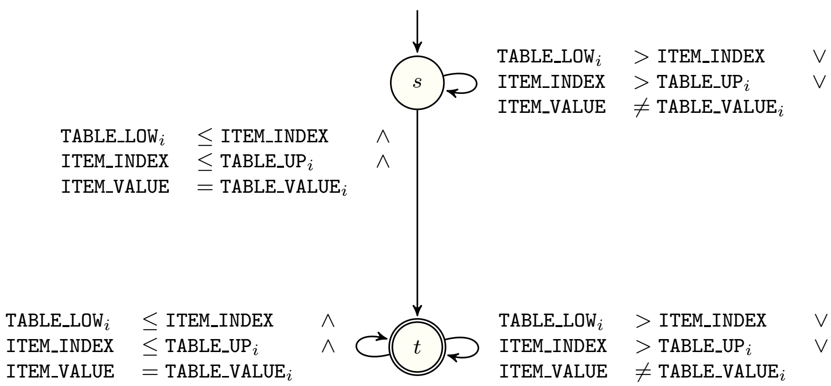



Figure 5.750: Automaton of the `stage_element` constraint

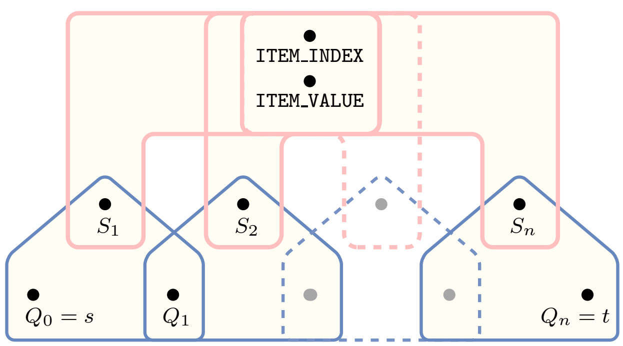



Figure 5.751: Hypergraph of the reformulation corresponding to the automaton of the `stage_element` constraint

## 5.376 stretch_circuit

DESCRIPTION LINKS GRAPH

**Origin** [305]

**Constraint** `stretch_circuit(VARIABLES, VALUES)`

**Usual name** `stretch`

**Arguments**

| | | |
|---|---|---|
| `VARIABLES` | : | `collection(var-dvar)` |
| `VALUES` | : | `collection(val-int, lmin-int, lmax-int)` |

**Restrictions**

$|$`VARIABLES`$| > 0$
`required(VARIABLES, var)`
$|$`VALUES`$| > 0$
`required(VALUES, [val, lmin, lmax])`
`distinct(VALUES, val)`
`VALUES.lmin` $\leq$ `VALUES.lmax`
`VALUES.lmin` $\leq |$`VARIABLES`$|$
`sum(VALUES.lmin)` $\leq |$`VARIABLES`$|$

**Purpose**

In order to define the meaning of the `stretch_path` constraint, we first introduce the notions of *stretch* and *span*. Let $n$ be the number of variables of the collection `VARIABLES` and let $i$, $j$ $(0 \leq i < n, 0 \leq j < n)$ be two positions within the collection of variables `VARIABLES` such that the following conditions apply:

- If $i \leq j$ then all variables $X_i, \ldots, X_j$ take a same value from the set of values of the `val` attribute.

  If $i > j$ then all variables $X_i, \ldots, X_{n-1}, X_0, \ldots, X_j$ take a same value from the set of values of the `val` attribute.
- $X_{(i-1) \bmod n}$ is different from $X_i$.
- $X_{(j+1) \bmod n}$ is different from $X_j$.

We call such a set of variables a *stretch*. The *span* of the stretch is equal to $1 + (j - i) \bmod n$, while the *value* of the stretch is $X_i$. We now define the condition enforced by the `stretch_circuit` constraint.

Each item $($`val` $- v,$ `lmin` $- s,$ `lmax` $- t)$ of the `VALUES` collection enforces the minimum value $s$ as well as the maximum value $t$ for the span of a stretch of value $v$.

Note that:

1. Having an item $($`val` $- v,$ `lmin` $- s,$ `lmax` $- t)$ with $s$ strictly greater than $0$ does not mean that value $v$ should be assigned to one of the variables of collection `VARIABLES`. It rather means that, when value $v$ is used, all stretches of value $v$ must have a span that belong to interval $[s, t]$.
2. A variable of the collection `VARIABLES` may be assigned a value that is not defined in the `VALUES` collection.

**Example**

$$\left(\begin{array}{l}\langle 6,6,3,1,1,1,6,6\rangle,\\ \left\langle\begin{array}{lll}\texttt{val}-1 & \texttt{lmin}-2 & \texttt{lmax}-4,\\ \texttt{val}-2 & \texttt{lmin}-2 & \texttt{lmax}-3,\\ \texttt{val}-3 & \texttt{lmin}-1 & \texttt{lmax}-6,\\ \texttt{val}-6 & \texttt{lmin}-2 & \texttt{lmax}-4\end{array}\right\rangle\end{array}\right)$$

The `stretch_circuit` constraint holds since the sequence $6\ 6\ 3\ 1\ 1\ 1\ 6\ 6$ contains three stretches $6\ 6\ 6\ 6$, $3$, and $1\ 1\ 1$ respectively verifying the following conditions:

- The span of the first stretch $6\ 6\ 6\ 6$ is located within interval $[2, 4]$ (i.e., the limit associated with value $6$).
- The span of the second stretch $3$ is located within interval $[1, 6]$ (i.e., the limit associated with value $3$).
- The span of the third stretch $1\ 1\ 1$ is located within interval $[2, 4]$ (i.e., the limit associated with value $1$).

**Typical**

$|\texttt{VARIABLES}| > 1$
$\text{range}(\texttt{VARIABLES.var}) > 1$
$|\texttt{VARIABLES}| > |\texttt{VALUES}|$
$|\texttt{VALUES}| > 1$
$\texttt{VALUES.lmax} \leq |\texttt{VARIABLES}|$

**Symmetries**

- Items of `VARIABLES` can be shifted.
- Items of `VALUES` are permutable.
- All occurrences of two distinct values in `VARIABLES.var` or `VALUES.val` can be swapped; all occurrences of a value in `VARIABLES.var` or `VALUES.val` can be renamed to any unused value.

**Usage**

The article [305], which originally introduced the `stretch` constraint, quotes rostering problems as typical examples of use of this constraint.

**Remark**

We split the origin `stretch` constraint into the `stretch_circuit` and the `stretch_path` constraints that respectively use the *PATH LOOP* and *CIRCUIT LOOP* arc generators. We also reorganise the parameters: the `VALUES` collection describes the attributes of each value that can be assigned to the variables of the `stretch_circuit` constraint. Finally we skipped the pattern constraint that tells what values can follow a given value.

**Algorithm**

A first filtering algorithm was described in the original article of G. Pesant [305]. An algorithm that also generates explanations is given in [360]. The first filtering algorithm achieving arc-consistency is depicted in [208, 209]. This algorithm is based on dynamic programming and handles the fact that some values can be followed by only a given subset of values.

**Reformulation**

The `stretch_circuit` constraint can be reformulated in term of a `stretch_path` constraint. Let $LMAX$ denote the maximum value taken by the `lmax` attribute within the items of the collection `VALUES`, let $n$ be the number of variables of the collection `VARIABLES`, and let $\delta = \min(LMAX, n)$. The first and second arguments of the `stretch_path` constraint are created in the following way:

- We pass to the stretch_path the variables of the collection VARIABLES to which we add the $\delta$ first variables of the collection VARIABLES.
- We pass to the stretch_path the values of the collection VALUES with the following modification: to each value $v$ for which the corresponding lmax attribute is greater than or equal to $n$ we reset its value to $n + \delta$.

Even if stretch_path can achieve arc-consistency this reformulation may not achieve arc-consistency since it duplicates variables.

Using this reformulation, the example

$$\texttt{stretch\_circuit}(\langle 6,6,3,1,1,1,6,6\rangle, \langle \texttt{val}-1\ \texttt{lmin}-2\ \texttt{lmax}-4,\ \texttt{val}-2\ \texttt{lmin}-2\ \texttt{lmax}-3,\ \texttt{val}-3\ \texttt{lmin}-1\ \texttt{lmax}-6,\ \texttt{val}-6\ \texttt{lmin}-2\ \texttt{lmax}-4\rangle)$$

of the **Example** slot is reformulated as:

$$\texttt{stretch\_path}(\langle 6,6,3,1,1,1,6,6,6,6,3,1,1,1\rangle, \langle \texttt{val}-1\ \texttt{lmin}-2\ \texttt{lmax}-4,\ \texttt{val}-2\ \texttt{lmin}-2\ \texttt{lmax}-3,\ \texttt{val}-3\ \texttt{lmin}-1\ \texttt{lmax}-6,\ \texttt{val}-6\ \texttt{lmin}-2\ \texttt{lmax}-4\rangle)$$

In the reformulation $\delta$ was equal to $6$, and the VALUES collection was left unchanged since no lmax attribute was equal to the number of variables of the VARIABLES collection (i.e., $8$).

**See also** **common keyword:** group *(timetabling constraint)*,
pattern *(sliding sequence constraint,timetabling constraint)*,
sliding_distribution *(sliding sequence constraint)*,
stretch_path *(sliding sequence constraint,timetabling constraint)*.

**used in reformulation:** stretch_path.

**Keywords** **characteristic of a constraint:** cyclic.

**constraint type:** timetabling constraint, sliding sequence constraint.

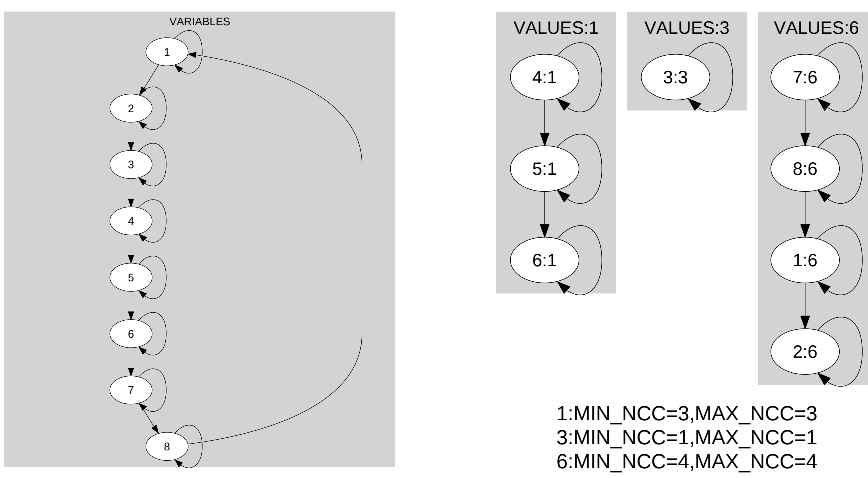
**filtering:** dynamic programming, arc-consistency, duplicated variables.

For all items of VALUES:

| | |
|---|---|
| **Arc input(s)** | VALUES |
| **Arc generator** | $CIRCUIT \mapsto$ `collection`(`variables1`, `variables2`)<br>$LOOP \mapsto$ `collection`(`variables1`, `variables2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • `variables1.var` = `VALUES.val`<br>• `variables2.var` = `VALUES.val` |
| **Graph property(ies)** | • `not_in`(**MIN_NCC**, 1, `VALUES.lmin` − 1)<br>• **MAX_NCC** ≤ `VALUES.lmax` |

**Graph model** Part (A) of Figure 5.752 shows the initial graphs associated with values 1, 2, 3 and 6 of the **Example** slot. Part (B) of Figure 5.752 shows the corresponding final graphs associated with values 1, 3 and 6. Since value 2 is not assigned to any variable of the VARIABLES collection the final graph associated with value 2 is empty. The `stretch_circuit` constraint holds since:

- For value 1 we have one connected component for which the number of vertices is greater than or equal to 2 and less than or equal to 4,
- For value 2 we do not have any connected component,
- For value 3 we have one connected component for which the number of vertices is greater than or equal to 1 and less than or equal to 6,
- For value 6 we have one connected component for which the number of vertices is greater than or equal to 2 and less than or equal to 4.




**(A)** **(B)**

Figure 5.752: Initial and final graph of the `stretch_circuit` constraint

## 5.377 stretch_path

DESCRIPTION LINKS GRAPH AUTOMATON

**Origin** [305]

**Constraint** stretch_path(VARIABLES, VALUES)

**Usual name** stretch

**Arguments**

| | | |
|---|---|---|
| VARIABLES | : | collection(var−dvar) |
| VALUES | : | collection(val−int, lmin−int, lmax−int) |

**Restrictions**

|VARIABLES| > 0
required(VARIABLES, var)
|VALUES| > 0
required(VALUES, [val, lmin, lmax])
distinct(VALUES, val)
VALUES.lmin ≥ 0
VALUES.lmin ≤ VALUES.lmax
VALUES.lmin ≤ |VARIABLES|

**Purpose**

In order to define the meaning of the stretch_path constraint, we first introduce the notions of *stretch* and *span*. Let $n$ be the number of variables of the collection VARIABLES. Let $X_i, \ldots, X_j$ $(1 \leq i \leq j \leq n)$ be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables $X_i, \ldots, X_j$ take a same value from the set of values of the val attribute,
- $i = 1$ or $X_{i-1}$ is different from $X_i$,
- $j = n$ or $X_{j+1}$ is different from $X_j$.

We call such a set of variables a *stretch*. The *span* of the stretch is equal to $j - i + 1$, while the *value* of the stretch is $X_i$. We now define the condition enforced by the stretch_path constraint.

Each item $(\mathtt{val} - v, \mathtt{lmin} - s, \mathtt{lmax} - t)$ of the VALUES collection enforces the minimum value $s$ as well as the maximum value $t$ for the span of a stretch of value $v$ over consecutive variables of the VARIABLES collection.

Note that:

1. Having an item $(\mathtt{val} - v, \mathtt{lmin} - s, \mathtt{lmax} - t)$ with $s$ strictly greater than $0$ does not mean that value $v$ should be assigned to one of the variables of collection VARIABLES. It rather means that, when value $v$ is used, all stretches of value $v$ must have a span that belong to interval $[s, t]$.
2. A variable of the collection VARIABLES may be assigned a value that is not defined in the VALUES collection.

**Example**

$$\left(\begin{array}{l}\langle 6,6,3,1,1,1,6,6\rangle, \\ \left\langle\begin{array}{lll}\texttt{val}-1 & \texttt{lmin}-2 & \texttt{lmax}-4, \\ \texttt{val}-2 & \texttt{lmin}-2 & \texttt{lmax}-3, \\ \texttt{val}-3 & \texttt{lmin}-1 & \texttt{lmax}-6, \\ \texttt{val}-6 & \texttt{lmin}-2 & \texttt{lmax}-2\end{array}\right\rangle\end{array}\right)$$

The `stretch_path` constraint holds since the sequence $6\ 6\ 3\ 1\ 1\ 1\ 6\ 6$ contains four stretches $6\ 6$, $3$, $1\ 1\ 1$, and $6\ 6$ respectively verifying the following conditions:

- The span of the first stretch $6\ 6$ is located within interval $[2,2]$ (i.e., the limit associated with value $6$).
- The span of the second stretch $3$ is located within interval $[1,6]$ (i.e., the limit associated with value $3$).
- The span of the third stretch $1\ 1\ 1$ is located within interval $[2,4]$ (i.e., the limit associated with value $1$).
- The span of the fourth stretch $6\ 6$ is located within interval $[2,2]$ (i.e., the limit associated with value $6$).

**Typical**

$|\texttt{VARIABLES}| > 1$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$|\texttt{VARIABLES}| > |\texttt{VALUES}|$
$|\texttt{VALUES}| > 1$
$\texttt{sum}(\texttt{VALUES.lmin}) \leq |\texttt{VARIABLES}|$
$\texttt{VALUES.lmax} \leq |\texttt{VARIABLES}|$

**Symmetries**

- Items of `VARIABLES` can be reversed.
- Items of `VALUES` are permutable.
- All occurrences of two distinct values in `VARIABLES.var` or `VALUES.val` can be swapped; all occurrences of a value in `VARIABLES.var` or `VALUES.val` can be renamed to any unused value.

**Usage**

The article [305], which originally introduced the `stretch` constraint, quotes rostering problems as typical examples of use of this constraint.

**Remark**

We split the original `stretch` constraint into the `stretch_path` and the `stretch_circuit` constraints that respectively use the $PATH$ $LOOP$ and the $CIRCUIT$ $LOOP$ arc generators. We also reorganise the parameters: the `VALUES` collection describes the attributes of each value that can be assigned to the variables of the `stretch_path` constraint. Finally we skipped the pattern constraint that tells what values can follow a given value. A extension of this constraint (i.e., stretch plus pattern), called `forced_shift_stretch`, where one can specify for each value $v$ with a $0$-$1$ variable, whether it should occur at least once or not at all, was proposed in [209]. By reduction to Hamiltonian path it was shown that enforcing arc-consistency for `forced_shift_stretch` is NP-hard [209].

**Algorithm**

A first filtering algorithm was described in the original article of G. Pesant [305]. A second filtering algorithm, based on dynamic programming, achieving arc-consistency is depicted in [208, 209]. It also handles the fact that some values can be followed by only a given subset of values. An other alternative achieving arc-consistency is to use the automaton described in the **Automaton** slot.

**Systems** stretchPath in **Choco**, stretch in **JaCoP**.

**See also** **common keyword:** change_continuity, group *(timetabling constraint)*, group_skip_isolated_item *(timetabling constraint,sequence)*, min_size_full_zero_stretch *(sequence)*, pattern *(sliding sequence constraint,timetabling constraint)*, sliding_distribution *(sliding sequence constraint)*, stretch_circuit *(sliding sequence constraint,timetabling constraint)*.

**generalisation:** stretch_path_partition *(*variable *replaced by* variable $\in$ partition*)*.

**uses in its reformulation:** stretch_circuit.

**Keywords** **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

**combinatorial object:** sequence.

**constraint network structure:** Berge-acyclic constraint network.

**constraint type:** timetabling constraint, sliding sequence constraint.

**filtering:** dynamic programming, arc-consistency.

**final graph structure:** consecutive loops are connected.

For all items of $\texttt{VALUES}$:

**Arc input(s)** $\texttt{VARIABLES}$

**Arc generator** $\mathit{PATH} \mapsto \texttt{collection}(\texttt{variables1}, \texttt{variables2})$
$\mathit{LOOP} \mapsto \texttt{collection}(\texttt{variables1}, \texttt{variables2})$

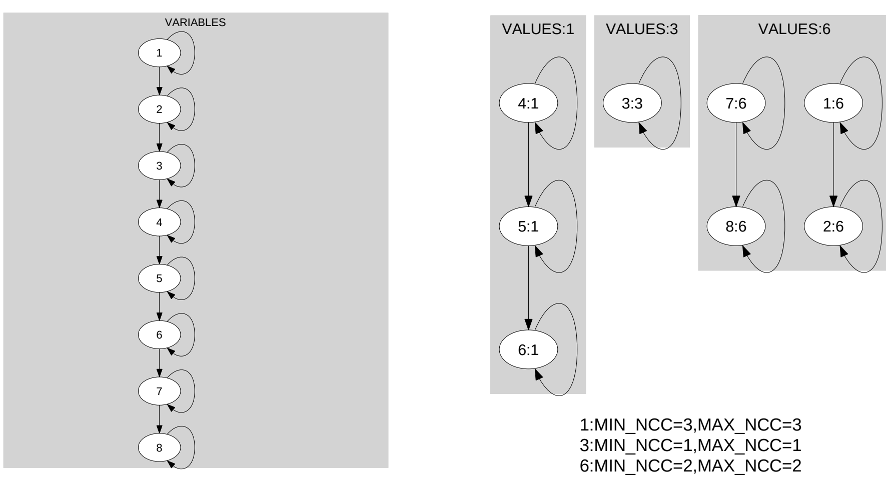
**Arc arity** 2

**Arc constraint(s)**
- $\texttt{variables1.var} = \texttt{VALUES.val}$
- $\texttt{variables2.var} = \texttt{VALUES.val}$

**Graph property(ies)**
- $\texttt{not\_in}(\mathbf{MIN\_NCC}, 1, \texttt{VALUES.lmin} - 1)$
- $\mathbf{MAX\_NCC} \leq \texttt{VALUES.lmax}$

**Graph model** Part (A) of Figure 5.753 shows the initial graphs associated with values 1, 2, 3 and 6 of the **Example** slot. Part (B) of Figure 5.753 shows the corresponding final graphs associated with values 1, 3 and 6. Since value 2 is not assigned to any variable of the $\texttt{VARIABLES}$ collection the final graph associated with value 2 is empty. The $\texttt{stretch\_path}$ constraint holds since:

- For value 1 we have one connected component for which the number of vertices 3 is greater than or equal to 2 and less than or equal to 4,
- For value 2 we do not have any connected component,
- For value 3 we have one connected component for which the number of vertices 1 is greater than or equal to 1 and less than or equal to 6,
- For value 6 we have two connected components that both contain two vertices: this is greater than or equal to 2 and less than or equal to 2.




**(A)** **(B)**

Figure 5.753: Initial and final graph of the $\texttt{stretch\_path}$ constraint

During the presentation of this constraint at CP'2001 the following point was mentioned: it could be useful to allow domain variables for the minimum and the maximum values of a stretch. This could be achieved in the following way: the `lmin` (respectively `lmax`) attribute would now be a domain variable that gives the size of the shortest (respectively longest) stretch. Finally within the **Graph property(ies)** slot we would replace $\geq$ (and $\leq$) by $=$.

**Automaton** Let $n$ and $m$ respectively denote the quantities $|\texttt{VARIABLES}|$ and $|\texttt{VALUES}|$. Furthermore, let $\texttt{val}_i$, $\texttt{lmin}_i$ and $\texttt{lmax}_i$, $(i \in [1, m])$, respectively be shortcuts for the expressions $\texttt{VALUES}[i].\texttt{val}$, $\texttt{VALUES}[i].\texttt{lmin}$ and $\texttt{VALUES}[i].\texttt{lmax}$. Without loss of generality, we assume that all the `lmin` attributes of the items of the `VALUES` collection are at least equal to 1. The following automaton $\mathcal{A}$ involving $1 + \texttt{lmax}_1 + \texttt{lmax}_2 + \cdots + \texttt{lmax}_m$ states only accepts solutions to the `stretch_path` constraint. Automaton $\mathcal{A}$ has the following states:

- an initial state $s$ that is also an accepting state,
- $\forall i \in [1, m], \forall j \in [1, \texttt{lmin}_i - 1]$, a non-accepting state $s_{i,j}$,
- $\forall i \in [1, m], \forall j \in [\texttt{lmin}_i, \texttt{lmax}_i]$, an accepting state $s_{i,j}$.

Transitions of $\mathcal{A}$ are defined in the following way:

- $\forall i \in [1, m]$, a transition from $s$ to $s_{i,1}$ labelled by condition $X_l = \texttt{val}_i$,
- a transition from $s$ to $s$ labelled by condition $X_l \neq \texttt{val}_1 \wedge X_l \neq \texttt{val}_2 \wedge \cdots \wedge X_l \neq \texttt{val}_m$,
- $\forall i \in [1, m], \forall j \in [\texttt{lmin}_i, \texttt{lmax}_i]$, a transition from $s_{i,j}$ to $s$ labelled by condition $X_l \neq \texttt{val}_1 \wedge X_l \neq \texttt{val}_2 \wedge \cdots \wedge X_l \neq \texttt{val}_m$,
- $\forall i \in [1, m], \forall j \in [1, \texttt{lmax}_i - 1]$, a transition from $s_{i,j}$ to $s_{i,j+1}$ labelled by condition $X_l = \texttt{val}_i$,
- $\forall i \in [1, m], \forall j \in [\texttt{lmin}_i, \texttt{lmax}_i], \forall k \neq i \in [1, m]$, a transition from $s_{i,j}$ to $s_{k,1}$ labelled by condition $X_l = \texttt{val}_k$.

Figure 5.754 depicts the automaton associated with the `stretch_path` constraint of the **Example** slot. Transitions labels 0, 1, 2, 3 and 4 respectively correspond to the conditions $X_l \neq 1 \wedge X_l \neq 2 \wedge X_l \neq 3 \wedge X_l \neq 6$, $X_l = 1$, $X_l = 2$, $X_l = 3$, $X_l = 6$ (since values 1, 2, 3 and 6 respectively correspond to the values of the first, second, third and fourth item of the `VALUES` collection). The `stretch_path` constraint holds since the corresponding sequence of visited states, $s$ $s_{41}$ $s_{42}$ $s_{31}$ $s_{11}$ $s_{12}$ $s_{13}$ $s_{41}$ $s_{42}$, ends up in an accepting state (i.e., accepting states are denoted graphically by a double circle in the figure).

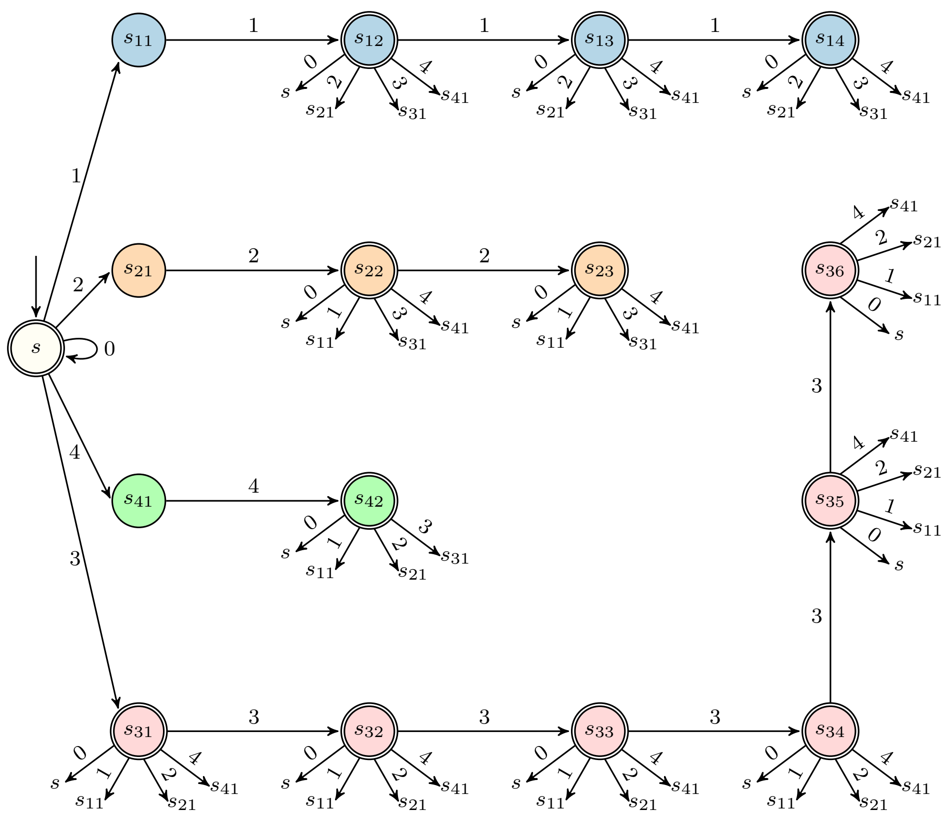



Figure 5.754: Automaton of the `stretch_path` constraint of the **Example** slot (states related to a same stretch have the same colour)

## 5.378 stretch_path_partition

DESCRIPTION LINKS

**Origin** Derived from stretch_path.

**Constraint** stretch_path_partition(VARIABLES, PARTLIMITS)

**Synonym** stretch.

**Type**

VALUES : collection(val−int)

**Arguments**

VARIABLES : collection(var−dvar)
PARTLIMITS : collection(p − VALUES, lmin−int, lmax−int)

**Restrictions**

$|\text{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)
$|\text{VARIABLES}| > 0$
required(VARIABLES, var)
$|\text{PARTLIMITS}| > 0$
required(PARTLIMITS, [p, lmin, lmax])
$\text{PARTLIMITS.lmin} \geq 0$
$\text{PARTLIMITS.lmin} \leq \text{PARTLIMITS.lmax}$
$\text{PARTLIMITS.lmin} \leq |\text{VARIABLES}|$

In order to define the meaning of the `stretch_path_partition` constraint, we first introduce the notions of *stretch* and *span*. Let $n$ be the number of variables of the collection `VARIABLES`. Let $X_i, \dots, X_j$ $(1 \leq i \leq j \leq n)$ be consecutive variables of the collection of variables `VARIABLES` such that the following conditions apply:

- All variables $X_i, \dots, X_j$ take their values in the same partition of the `PARTLIMITS` collection (i.e., $\exists l \in [1, |\texttt{PARTLIMITS}|]$ such that $\forall k \in [i, j]$ : $X_k \in \texttt{PARTLIMITS}[l].p$),
- $i = 1$ or $X_{i-1}$ is different from $X_i$,
- $j = n$ or $X_{j+1}$ is different from $X_j$.

We call such a set of variables a *stretch*. The *span* of the stretch is equal to $j - i + 1$, while the *value* of the stretch is $l$. We now define the condition enforced by the `stretch_path_partition` constraint.

**Purpose**

Each item $\texttt{PARTLIMITS}[l] = (\texttt{p} - values, \texttt{lmin} - s, \texttt{lmax} - t)$ of the `PARTLIMITS` collection enforces the minimum value $s$ as well as the maximum value $t$ for the span of a stretch of value $l$ over consecutive variables of the `VARIABLES` collection.

Note that:

1. Having an item $\texttt{PARTLIMITS}[l] = (\texttt{p} - values, \texttt{lmin} - s, \texttt{lmax} - t)$ with $s$ strictly greater than 0 does not mean that values of $values$ should be assigned to one of the variables of collection `VARIABLES`. It rather means that, when a value of $values$ is used, all stretches of value $l$ must have a span that belong to interval $[s, t]$.
2. A variable of the collection `VARIABLES` may be assigned a value that is not defined in the attribute `p` of the `PARTLIMITS` collection.

**Example**

$$\left(\begin{array}{l}\langle 1,2,0,0,2,2,2,0\rangle, \\ \left\langle\begin{array}{lll}\texttt{p}-\langle 1,2\rangle & \texttt{lmin}-2 & \texttt{lmax}-4, \\ \texttt{p}-\langle 3\rangle & \texttt{lmin}-0 & \texttt{lmax}-2\end{array}\right\rangle\end{array}\right)$$

The `stretch_path_partition` constraint holds since the sequence 1 2 0 0 2 2 2 0 contains two stretches 1 2, and 2 2 2 respectively verifying the following conditions:

- The span of the first stretch 1 2 is located within interval $[2, 4]$ (i.e., the limit associated with item $\texttt{PARTLIMITS}[1]$).
- The span of the second stretch 2 2 2 is located within interval $[2, 4]$ (i.e., the limit associated with item $\texttt{PARTLIMITS}[1]$).

**Typical**

$|\texttt{VARIABLES}| > 1$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$|\texttt{VARIABLES}| > |\texttt{PARTLIMITS}|$
$|\texttt{PARTLIMITS}| > 1$
$\texttt{sum}(\texttt{PARTLIMITS.lmin}) \leq |\texttt{VARIABLES}|$
$\texttt{PARTLIMITS.lmax} \leq |\texttt{VARIABLES}|$

**Symmetries**

- Items of VARIABLES can be reversed.
- Items of PARTLIMITS are permutable.
- Items of PARTLIMITS.p are permutable.
- All occurrences of two distinct tuples of values in VARIABLES.var or PARLIMITS.p.val can be swapped; all occurrences of a tuple of values in VARIABLES.var or PARLIMITS.p.val can be renamed to any unused tuple of values.

**See also**

**common keyword:** pattern *(sliding sequence constraint)*.

**specialisation:** stretch_path *(*variable $\in$ partition *replaced by* variable*)*.

**Keywords**

**characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint, partition.

**combinatorial object:** sequence.

**constraint network structure:** Berge-acyclic constraint network.

**constraint type:** timetabling constraint, sliding sequence constraint.

**filtering:** arc-consistency.

**final graph structure:** consecutive loops are connected.

## 5.379 strict_lex2

DESCRIPTION LINKS

**Origin** [168]

**Constraint** strict_lex2(MATRIX)

**Type** VECTOR : collection(var−dvar)

**Argument** MATRIX : collection(vec − VECTOR)

**Restrictions**
$|\texttt{VECTOR}| \geq 1$
required(VECTOR, var)
required(MATRIX, vec)
same_size(MATRIX, vec)

**Purpose** Given a matrix of domain variables, enforces that both adjacent rows, and adjacent columns are lexicographically ordered (adjacent rows and adjacent columns cannot be equal).

**Example** $(\langle \texttt{vec} - \langle 2,2,3\rangle, \texttt{vec} - \langle 2,3,1\rangle\rangle)$

The strict_lex2 constraint holds since:

- The first row $\langle 2,2,3\rangle$ is lexicographically strictly less than the second row $\langle 2,3,1\rangle$.
- The first column $\langle 2,2\rangle$ is lexicographically strictly less than the second column $\langle 2,3\rangle$.
- The second column $\langle 2,3\rangle$ is lexicographically strictly less than the third column $\langle 3,1\rangle$.

**Typical**
$|\texttt{VECTOR}| > 1$
$|\texttt{MATRIX}| > 1$

**Symmetry** One and the same constant can be added to the var attribute of all items of MATRIX.vec.

**Usage** A *symmetry-breaking* constraint.

**Reformulation** The strict_lex2 constraint can be expressed as a conjunction of two lex_chain_less constraints: A first lex_chain_less constraint on the MATRIX argument and a second lex_chain_less constraint on the transpose of the MATRIX argument.

**Systems** strict_lex2 in **MiniZinc**.

**See also** **common keyword:** allperm, lex_lesseq *(lexicographic order)*.

**implies:** lex2, lex_chain_less.

**part of system of constraints:** lex_chain_less.

**Keywords** **constraint type:** predefined constraint, system of constraints, order constraint.

**modelling:** matrix, matrix model.

**symmetry:** symmetry, matrix symmetry, lexicographic order.

## 5.380 strictly_decreasing

DESCRIPTION LINKS GRAPH AUTOMATON

**Origin** Derived from strictly_increasing.

**Constraint** strictly_decreasing(VARIABLES)

**Argument** VARIABLES : collection(var−dvar)

**Restriction** required(VARIABLES, var)

**Purpose** The variables of the collection VARIABLES are strictly decreasing.

**Example** $(\langle 8, 4, 3, 1\rangle)$

The strictly_decreasing constraint holds since $8 > 4 > 3 > 1$.

**Typical** $|\text{VARIABLES}| > 2$

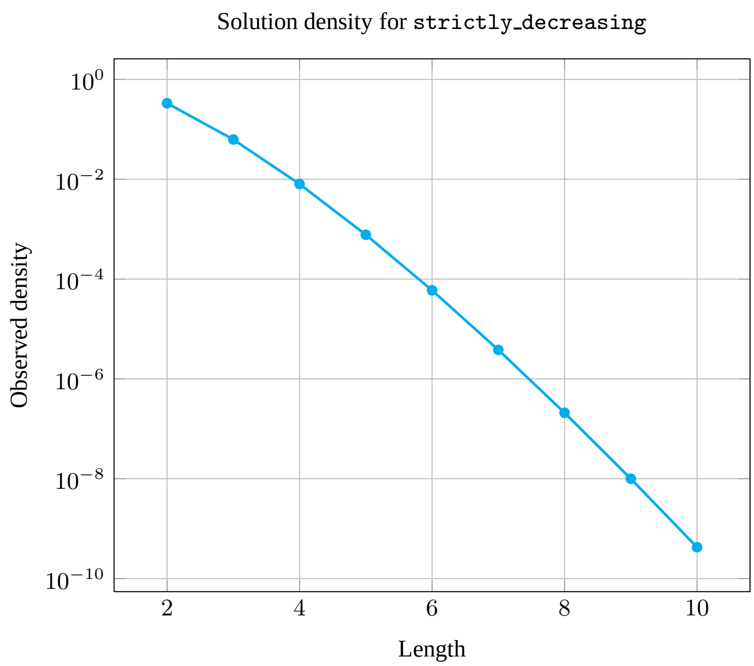
**Symmetry** One and the same constant can be added to the var attribute of all items of VARIABLES.

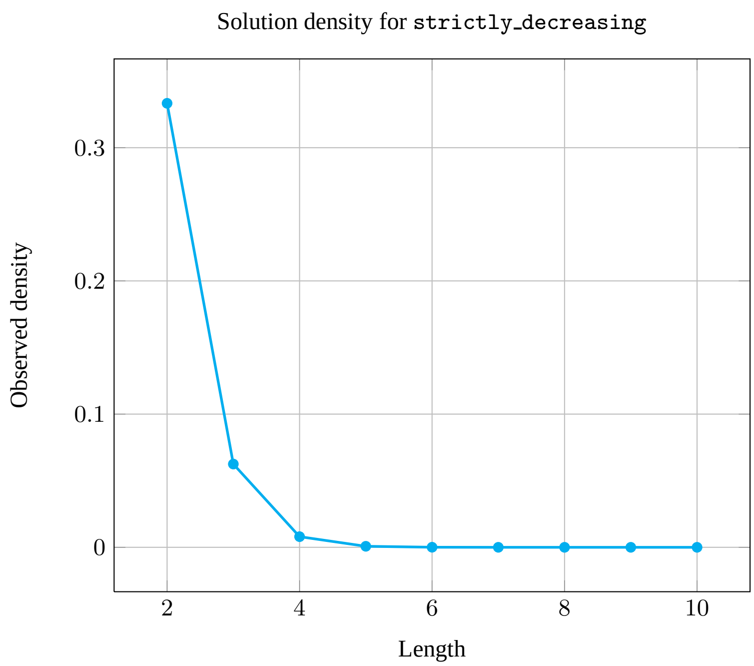
**Arg. properties** Contractible wrt. VARIABLES.

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Solutions | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Number of solutions for strictly_decreasing: domains $0..n$




**Systems** `increasingNValue` in **Choco**, `rel` in **Gecode**.

**See also** **common keyword:** `increasing` *(order constraint)*.

**comparison swapped:** strictly_increasing.

**implies:** alldifferent, decreasing.

**Keywords**

**characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

**constraint network structure:** sliding cyclic(1) constraint network(1).

**constraint type:** decomposition, order constraint.

**filtering:** arc-consistency.

| | |
|---|---|
| **Arc input(s)** | VARIABLES |
| **Arc generator** | $PATH \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var $>$ variables2.var |
| **Graph property(ies)** | $\mathbf{NARC} = \lvert\mathtt{VARIABLES}\rvert - 1$ |

**Graph model** Parts (A) and (B) of Figure 5.755 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.

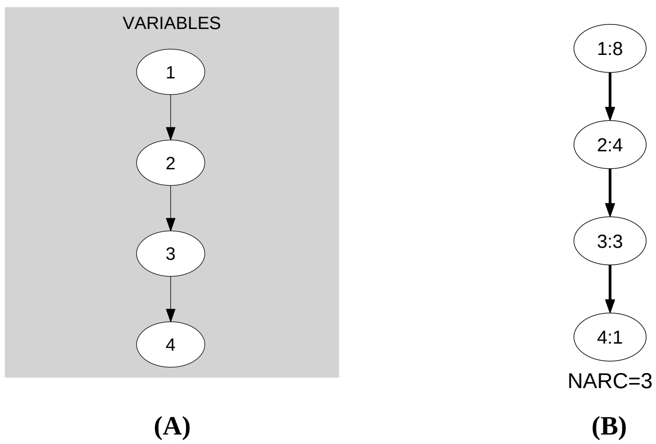



Figure 5.755: Initial and final graph of the `strictly_decreasing` constraint

**Automaton** Figure 5.756 depicts the automaton associated with the strictly_decreasing constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable $S_i$. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and $S_i$: $\mathtt{VAR}_i \leq \mathtt{VAR}_{i+1} \Leftrightarrow S_i$.

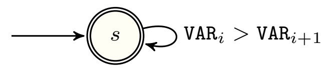



Figure 5.756: Automaton of the strictly_decreasing constraint

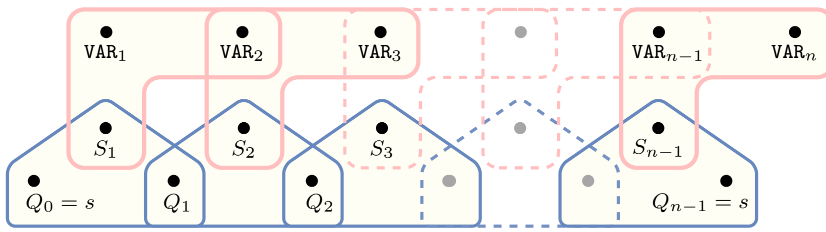



Figure 5.757: Hypergraph of the reformulation corresponding to the automaton of the strictly_decreasing constraint

## 5.381 strictly_increasing

DESCRIPTION LINKS GRAPH AUTOMATON

**Origin** KOALOG

**Constraint** strictly_increasing(VARIABLES)

**Argument** VARIABLES : collection(var−dvar)

**Restriction** required(VARIABLES, var)

**Purpose** The variables of the collection VARIABLES are strictly increasing.

**Example** $(\langle 1, 3, 6, 8\rangle)$

The strictly_increasing constraint holds since $1 < 3 < 6 < 8$.

**Typical** $|\texttt{VARIABLES}| > 2$

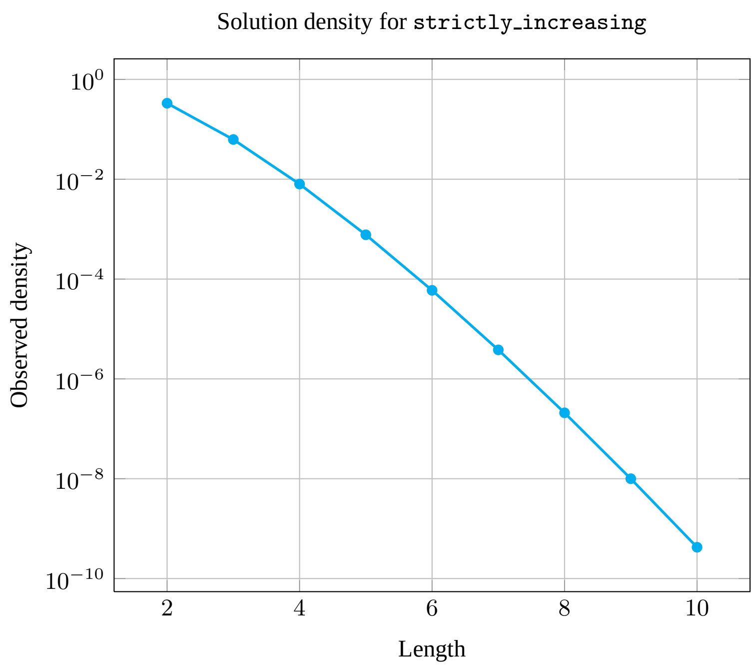
**Symmetry** One and the same constant can be added to the var attribute of all items of VARIABLES.

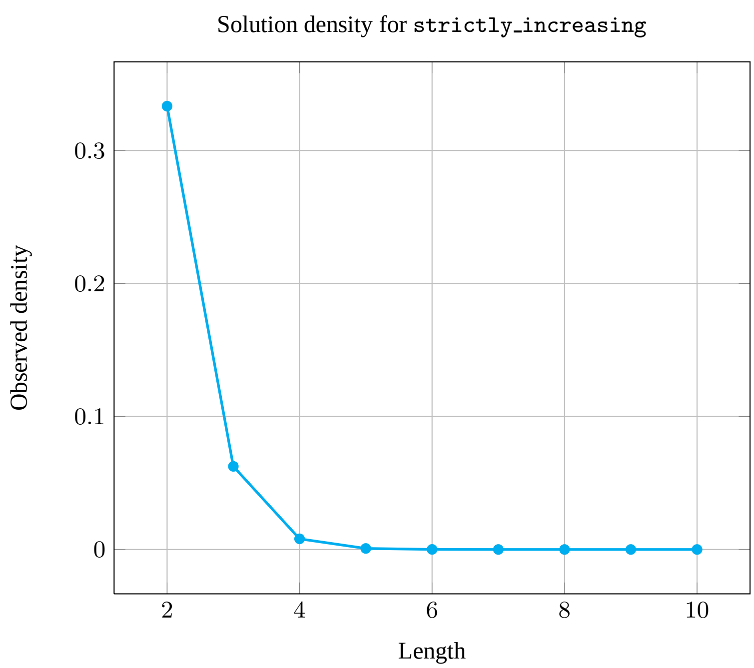
**Arg. properties** Contractible wrt. VARIABLES.

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Solutions | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Number of solutions for strictly_increasing: domains $0..n$




**Systems** increasingNValue in **Choco**, rel in **Gecode**.

**Used in** golomb, int_value_precede_chain, max_occ_of_tuples_of_values.

**See also**

**common keyword:** `decreasing` *(order constraint)*.

**comparison swapped:** `strictly_decreasing`.

**implied by:** `golomb`.

**implies:** `alldifferent`, `increasing`.

**uses in its reformulation:** `alldifferent`.

**Keywords**

**characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

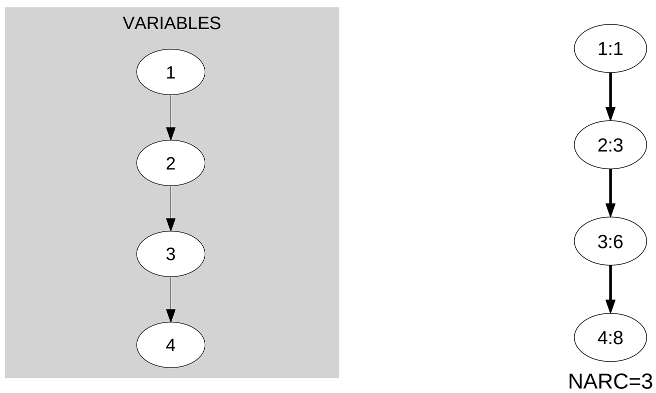
**constraint network structure:** sliding cyclic(1) constraint network(1).

**constraint type:** decomposition, order constraint.

**filtering:** arc-consistency.

| | |
|---|---|
| **Arc input(s)** | VARIABLES |
| **Arc generator** | $PATH \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var $<$ variables2.var |
| **Graph property(ies)** | $\mathbf{NARC} = \vert\texttt{VARIABLES}\vert - 1$ |

**Graph model** Parts (A) and (B) of Figure 5.758 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.




**(A)** **(B)**

Figure 5.758: Initial and final graph of the `strictly_increasing` constraint

**Automaton** Figure 5.759 depicts the automaton associated with the `strictly_increasing` constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection `VARIABLES` corresponds a 0-1 signature variable $S_i$. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and $S_i$: $\texttt{VAR}_i \geq \texttt{VAR}_{i+1} \Leftrightarrow S_i$.

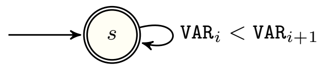



Figure 5.759: Automaton of the `strictly_increasing` constraint

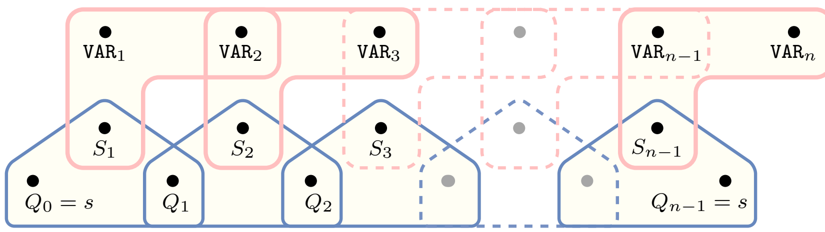



Figure 5.760: Hypergraph of the reformulation corresponding to the automaton of the `strictly_increasing` constraint

## 5.382 strongly_connected

DESCRIPTION LINKS GRAPH

**Origin** [5]

**Constraint** strongly_connected(NODES)

**Argument** NODES : collection(index−int, succ−svar)

**Restrictions**
required(NODES, [index, succ])
NODES.index $\geq 1$
NODES.index $\leq$ |NODES|
distinct(NODES, index)

**Purpose** Consider a digraph $G$ described by the NODES collection. Select a subset of arcs of $G$ so that we have a single strongly connected component involving all vertices of $G$.

**Example**

$$\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\{2\}, \\ \texttt{index}-2 & \texttt{succ}-\{3\}, \\ \texttt{index}-3 & \texttt{succ}-\{2,5\}, \\ \texttt{index}-4 & \texttt{succ}-\{1\}, \\ \texttt{index}-5 & \texttt{succ}-\{4\} \end{array}\right\rangle\right)$$

The strongly_connected constraint holds since the NODES collection depicts a graph involving a single strongly connected component (i.e., since we have a circuit visiting successively the vertices 1, 2, 3, 5, and 4).

**Typical** |NODES| $> 2$

**Symmetry** Items of NODES are permutable.

**Algorithm** The sketch of a filtering algorithm for the strongly_connected constraint is given in [142, page 89].

**See also**
**common keyword:** link_set_to_booleans *(constraint involving set variables)*.
**implied by:** connected.
**related:** circuit *(one single strongly connected component in the final solution)*.

**Keywords**
**constraint arguments:** constraint involving set variables.
**constraint type:** graph constraint.
**filtering:** linear programming.
**final graph structure:** strongly connected component.

| | |
|---|---|
| **Arc input(s)** | NODES |
| **Arc generator** | $CLIQUE \mapsto$ collection(nodes1, nodes2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | in_set(nodes2.index, nodes1.succ) |
| **Graph property(ies)** | $\underline{\mathbf{MIN\_NSCC}}$= \|NODES\| |

**Graph model** Part (A) of Figure 5.761 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.761 gives the final graph associated with the **Example** slot. The strongly_connected constraint holds since the final graph contains a single strongly connected component mentioning every vertex of the initial graph.

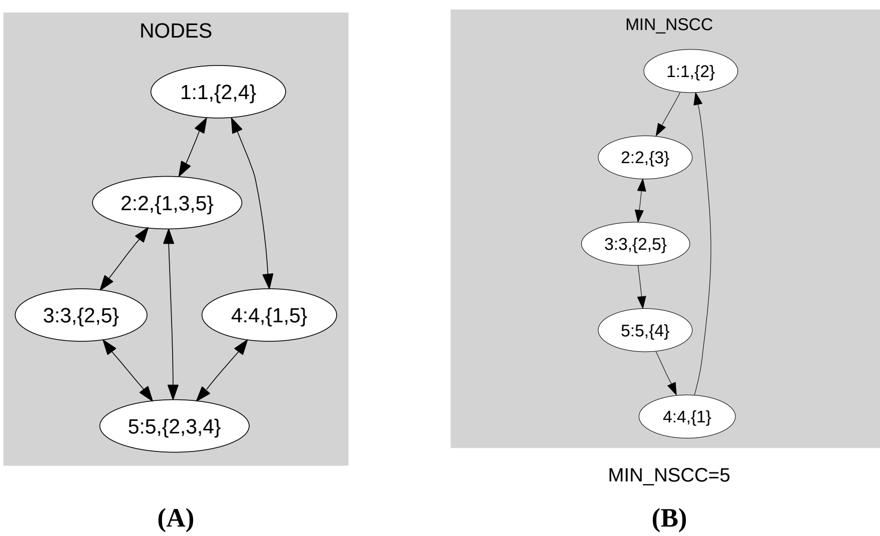



Figure 5.761: Initial and final graph of the strongly_connected set constraint

**Signature** Since the maximum number of vertices of the final graph is equal to |NODES| we can rewrite the graph property $\mathbf{MIN\_NSCC} = |\texttt{NODES}|$ to $\mathbf{MIN\_NSCC} \geq |\texttt{NODES}|$ and simplify $\underline{\overline{\mathbf{MIN\_NSCC}}}$ to $\overline{\mathbf{MIN\_NSCC}}$.

## 5.383 subgraph_isomorphism

DESCRIPTION LINKS

**Origin** [277]

**Constraint** subgraph_isomorphism(NODES_PATTERN, NODES_TARGET, FUNCTION)

**Arguments**

| | | |
|---|---|---|
| NODES_PATTERN | : | collection(index−int, succ−sint) |
| NODES_TARGET | : | collection(index−int, succ−svar) |
| FUNCTION | : | collection(image−dvar) |

**Restrictions**

required(NODES_PATTERN, [index, succ])
NODES_PATTERN.index $\geq 1$
NODES_PATTERN.index $\leq$ |NODES_PATTERN|
distinct(NODES_PATTERN, index)
NODES_PATTERN.succ $\geq 1$
NODES_PATTERN.succ $\leq$ |NODES_PATTERN|
required(NODES_TARGET, [index, succ])
NODES_TARGET.index $\geq 1$
NODES_TARGET.index $\leq$ |NODES_TARGET|
distinct(NODES_TARGET, index)
NODES_TARGET.succ $\geq 1$
NODES_TARGET.succ $\leq$ |NODES_TARGET|
required(FUNCTION, [image])
FUNCTION.image $\geq 1$
FUNCTION.image $\leq$ |NODES_TARGET|
distinct(FUNCTION, image)
|FUNCTION| = |NODES_PATTERN|

**Purpose**

Given two directed graphs PATTERN and TARGET enforce a one to one correspondence, defined by the function FUNCTION, between the vertices of the graph PATTERN and the vertices of an induced subgraph of TARGET so that, if there is an arc from $u$ to $v$ in the graph PATTERN, then there is also an arc from the image of $u$ to the image of $v$ in the induced subgraph of TARGET. The vertices of both graphs are respectively defined by the two collections of vertices NODES_PATTERN and NODES_TARGET. Within collection NODES_PATTERN the set of successors of each node is fixed, while this is not the case for the collection NODES_TARGET. This stems from the fact that the TARGET graph is not fixed (i.e., the lower and upper bounds of the target graph are specified when we post the subgraph_isomorphism constraint, while the induced subgraph of a solution to the subgraph_isomorphism constraint corresponds to a graph for which the upper and lower bounds are identical).

**Example**

$$\left(\begin{array}{l}\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\{2,4\},\\ \texttt{index}-2 & \texttt{succ}-\{1,3,4\},\\ \texttt{index}-3 & \texttt{succ}-\emptyset,\\ \texttt{index}-4 & \texttt{succ}-\emptyset\end{array}\right\rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\emptyset,\\ \texttt{index}-2 & \texttt{succ}-\{3,4,5\},\\ \texttt{index}-3 & \texttt{succ}-\emptyset,\\ \texttt{index}-4 & \texttt{succ}-\{2,5\},\\ \texttt{index}-5 & \texttt{succ}-\emptyset\end{array}\right\rangle,\\ \langle 4,2,3,5\rangle\end{array}\right)$$

Figure 5.762 gives the pattern (see Part (A)) and target graph (see Part (B)) of the **Example** slot as well as the one to one correspondence (see Part (C)) between the pattern graph and the induced subgraph of the target graph. The `subgraph_isomorphism` constraint since:

- To the arc from vertex $1$ to vertex $4$ in the pattern graph corresponds the arc from vertex $4$ to $5$ in the induced subgraph of the target graph.
- To the arc from vertex $1$ to vertex $2$ in the pattern graph corresponds the arc from vertex $4$ to $2$ in the induced subgraph of the target graph.
- To the arc from vertex $2$ to vertex $1$ in the pattern graph corresponds the arc from vertex $2$ to $4$ in the induced subgraph of the target graph.
- To the arc from vertex $2$ to vertex $4$ in the pattern graph corresponds the arc from vertex $2$ to $5$ in the induced subgraph of the target graph.
- To the arc from vertex $2$ to vertex $3$ in the pattern graph corresponds the arc from vertex $2$ to $3$ in the induced subgraph of the target graph.

**Typical**

$|\mathtt{NODES\_PATTERN}| > 1$
$|\mathtt{NODES\_TARGET}| > 1$

**Symmetries**

- Items of `NODES_PATTERN` are permutable.
- Items of `NODES_TARGET` are permutable.

**Usage**

Within the context of constraint programming the constraint was used for finding symmetries [325, 327, 326].

**Algorithm**

[412, 341, 254, 445].

**See also**

**related:** `graph_isomorphism`.

**Keywords**

**constraint arguments:** constraint involving set variables.

**constraint type:** predefined constraint, graph constraint.

**symmetry:** symmetry.

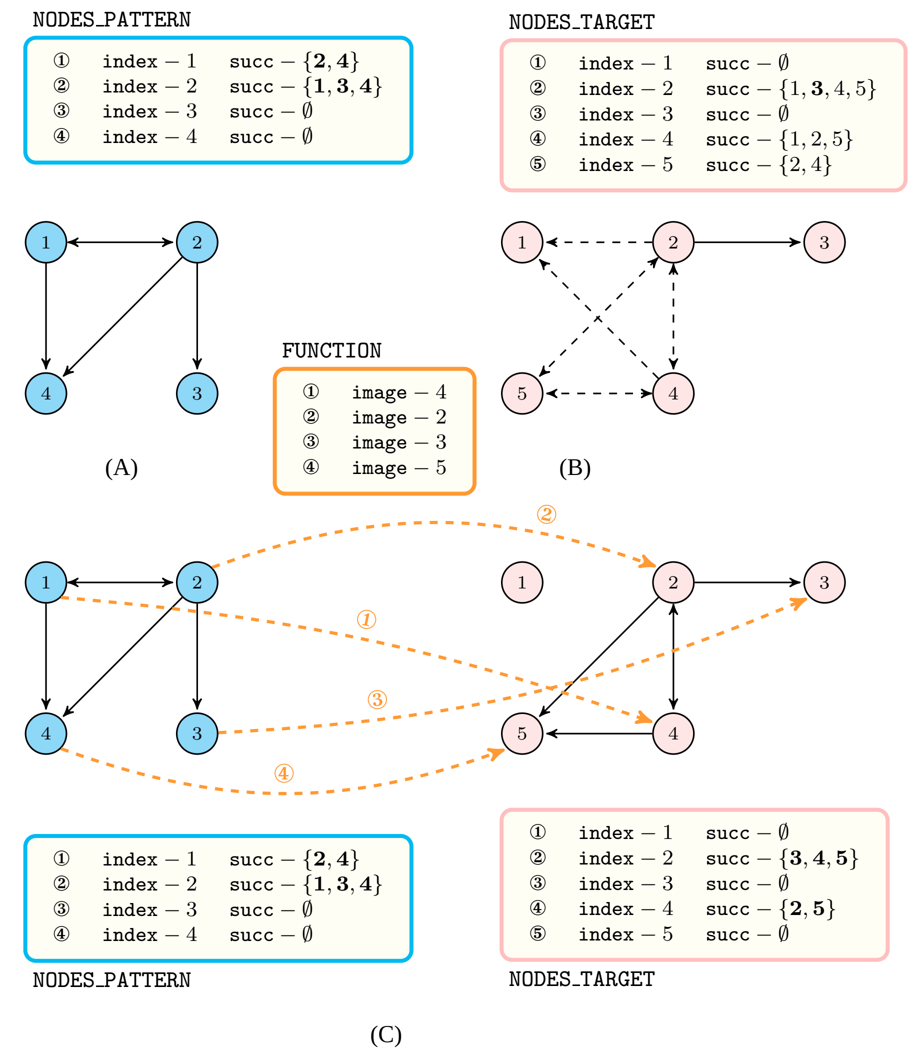



Figure 5.762: Illustration of the **Example** slot: (A) The pattern graph, (B) a possible initial target graph – plain arcs must belong to the induced subgraph, while dotted arcs may or may not belong to the induced subgraph – and (C) the correspondence, denoted by thick dashed arcs, between the vertices of the pattern graph and the vertices of the induced subgraph of the target graph. Within a set variable a bold value (respectively a plain value) represents a value that for sure belong (respectively that may belong) to the set.

# 5.384 sum

DESCRIPTION LINKS GRAPH

**Origin** [444].

**Constraint** sum(INDEX, SETS, CONSTANTS, S)

**Synonym** sum_pred.

**Arguments**

| | | |
|---|---|---|
| INDEX | : | dvar |
| SETS | : | collection(ind−int, set−sint) |
| CONSTANTS | : | collection(cst−int) |
| S | : | dvar |

**Restrictions**

$|\mathtt{SETS}| \geq 1$
required(SETS, [ind, set])
distinct(SETS, ind)
$|\mathtt{CONSTANTS}| \geq 1$
required(CONSTANTS, cst)

**Purpose** S is equal to the sum of the constants of CONSTANTS corresponding to the INDEX$^{th}$ set of the SETS collection.

**Example**

$$\left(\begin{array}{l} 8, \left\langle \begin{array}{ll} \mathtt{ind}-8 & \mathtt{set}-\{2,3\}, \\ \mathtt{ind}-1 & \mathtt{set}-\{3\}, \\ \mathtt{ind}-3 & \mathtt{set}-\{1,4,5\}, \\ \mathtt{ind}-6 & \mathtt{set}-\{2,4\} \end{array} \right\rangle, \\ \langle 4,9,1,3,1\rangle, 10 \end{array}\right)$$

The sum constraint holds since its last argument $\mathtt{S} = 10$ is equal to the sum of the $2^{th}$ and $3^{th}$ items of the collection $\langle 4,9,1,3,1\rangle$. As illustrated by Figure 5.763, this stems from the fact that its first argument $\mathtt{INDEX} = 8$ corresponds to the value of the ind attribute of the first item of the SETS collection. Consequently the corresponding set $\{2,3\}$ is used for summing the $2^{th}$ and $3^{th}$ items of the CONSTANTS collection.

**Typical**

$|\mathtt{SETS}| > 1$
$|\mathtt{CONSTANTS}| > |\mathtt{SETS}|$
$\mathtt{range}(\mathtt{CONSTANTS.cst}) > 1$

**Symmetry** Items of SETS are permutable.

**Arg. properties** Functional dependency: S determined by INDEX, SETS and CONSTANTS.

**Usage** In his article introducing the sum constraint, Tallys H. Yunes mentions the *Sequence Dependent Cumulative Cost Problem* as the subproblem that originally motivates this constraint.

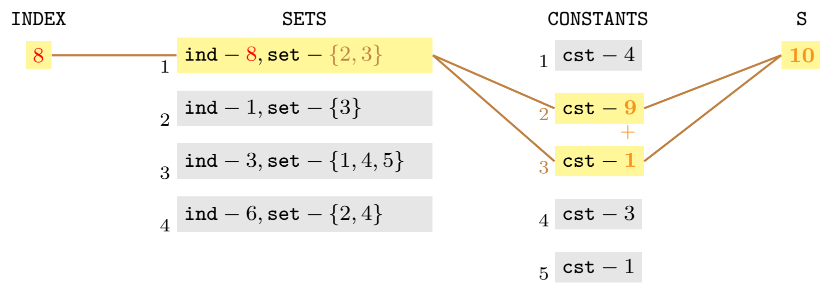



Figure 5.763: Illustration of the correspondence between the arguments of the `sum`(`INDEX`, `SETS`, `CONSTANTS`, `S`) constraint in the context of the **Example** slot (from right to left, $\mathtt{S} = 10$ is equal to the sum of the constants $9$ and $1$ corresponding to the indices $2$ and $3$ of the set for which the `ind` attribute is equal to $\mathtt{INDEX} = 8$)

**Remark** The `sum` constraint is called `sum_pred` in **MiniZinc** (`http://www.minizinc.org/`).

**Algorithm** The article [444] gives the convex hull relaxation of the `sum` constraint.

**Systems** `sum_pred` in **MiniZinc**.

**See also** **common keyword:** `element` (*data constraint*), `sum_ctr`, `sum_set` (*sum*).

**used in graph description:** `in_set`.

**Keywords** **characteristic of a constraint:** convex hull relaxation, sum.

**constraint type:** data constraint.

**filtering:** linear programming.

**modelling:** functional dependency.

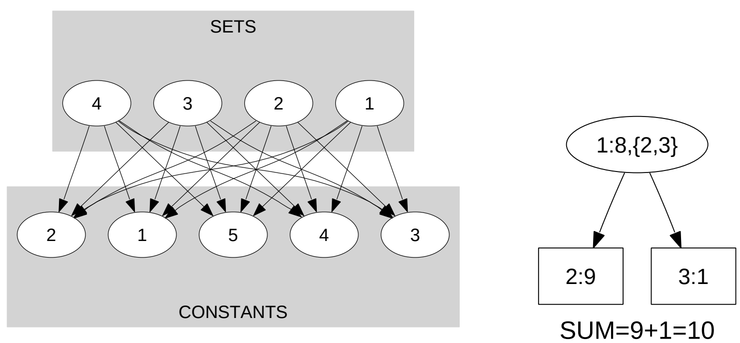
| | |
|---|---|
| **Arc input(s)** | SETS CONSTANTS |
| **Arc generator** | *PRODUCT* $\mapsto$ collection(sets, constants) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • INDEX = sets.ind<br>• in_set(constants.key, sets.set) |
| **Graph property(ies)** | **SUM**(CONSTANTS, cst) = S |

**Graph model** According to the value assigned to INDEX the arc constraint selects for the final graph:

- The INDEX$^{th}$ item of the SETS collection,
- The items of the CONSTANTS collection for which the key correspond to the indices of the INDEX$^{th}$ set of the SETS collection.

Finally, since we use the **SUM** graph property on the cst attribute of the CONSTANTS collection, the last argument S of the sum constraint is equal to the sum of the constants associated with the vertices of the final graph.

Parts (A) and (B) of Figure 5.764 respectively show the initial and final graph associated with the **Example** slot. Since we use the **SUM** graph property we show the vertices from which we compute S in a box.




**(A)** **(B)**

Figure 5.764: Initial and final graph of the sum constraint

## 5.385 sum_ctr

DESCRIPTION LINKS GRAPH

**Origin** Arithmetic constraint.

**Constraint** sum_ctr(VARIABLES, CTR, VAR)

**Synonyms** constant_sum, sum, linear, scalar_product.

**Arguments**

| | | |
|---|---|---|
| VARIABLES | : | collection(var−dvar) |
| CTR | : | atom |
| VAR | : | dvar |

**Restrictions**

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

**Purpose** Constraint the sum of a set of domain variables. More precisely, let S denote the sum of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

**Example** $(\langle 1, 1, 4\rangle, =, 6)$

The sum_ctr constraint holds since the condition $1 + 1 + 4 = 6$ is satisfied.

**Typical**

|VARIABLES| $> 1$
range(VARIABLES.var) $> 1$
CTR $\in [=, <, \geq, >, \leq]$

**Symmetry** Items of VARIABLES are permutable.

**Arg. properties**

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$ and minval(VARIABLES.var) $\geq 0$.
- Contractible wrt. VARIABLES when CTR $\in [\geq, >]$ and maxval(VARIABLES.var) $\leq 0$.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$ and minval(VARIABLES.var) $\geq 0$.
- Extensible wrt. VARIABLES when CTR $\in [<, \leq]$ and maxval(VARIABLES.var) $\leq 0$.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

**Remark** When CTR corresponds to $=$ this constraint is referenced under the names constant_sum in KOALOG (http://www.koalog.com/php/index.php) and sum in **JaCoP** (http://www.jacop.eu/).

**Systems** equation in **Choco**, linear in **Gecode**, scalar_product in **SICStus**.

**Used in** bin_packing, cumulative, cumulative_convex, cumulative_with_level_of_priority, cumulatives, indexed_sum, interval_and_sum, relaxed_sliding_sum, sliding_sum, sliding_time_window_sum.

**See also**

**assignment dimension added:** interval_and_sum *(assignment dimension corresponding to intervals is added)*.

**common keyword:** arith_sliding *(arithmetic constraint)*, increasing_sum *(sum)*, product_ctr, range_ctr *(arithmetic constraint)*, sum, sum_cubes_ctr, sum_powers4_ctr, sum_powers5_ctr, sum_powers6_ctr *(sum)*, sum_set *(arithmetic constraint)*, sum_squares_ctr *(sum)*.

**generalisation:** scalar_product *(arithmetic constraint where all coefficients are not necessarly equal to* 1*)*.

**implied by:** arith_sliding.

**system of constraints:** sliding_sum.

**Keywords**

**characteristic of a constraint:** sum.

**constraint type:** arithmetic constraint.

**heuristics:** regret based heuristics, regret based heuristics in matrix problems.

**Cond. implications**

- sum_ctr(VARIABLES, CTR, VAR)
  with VARIABLES.var $\geq 0$
  and VARIABLES.var $\leq 1$
  **implies** sum_squares_ctr(VARIABLES, CTR, VAR)
  when VARIABLES.var $\geq 0$
  and VARIABLES.var $\leq 1$.

- sum_ctr(VARIABLES, CTR, VAR)
  with VARIABLES.var $\geq -1$
  and VARIABLES.var $\leq 1$
  **implies** sum_cubes_ctr(VARIABLES, CTR, VAR)
  when VARIABLES.var $\geq -1$
  and VARIABLES.var $\leq 1$.

- sum_ctr(VARIABLES, CTR, VAR)
  with VARIABLES.var $\geq -1$
  and VARIABLES.var $\leq 1$
  **implies** sum_powers5_ctr(VARIABLES, CTR, VAR)
  when VARIABLES.var $\geq -1$
  and VARIABLES.var $\leq 1$.

- sum_ctr(VARIABLES, CTR, VAR)
  with CTR $\in [=]$
  and increasing(VARIABLES)
  **implies** increasing_sum(VARIABLES, VAR).

| | |
|---|---|
| **Arc input(s)** | VARIABLES |
| **Arc generator** | $SELF \mapsto$ collection(variables) |
| **Arc arity** | 1 |
| **Arc constraint(s)** | TRUE |
| **Graph property(ies)** | **SUM**(VARIABLES, var) CTR VAR |

**Graph model** Since we want to keep all the vertices of the initial graph we use the $SELF$ arc generator together with the TRUE arc constraint. This predefined arc constraint always holds.

Parts (A) and (B) of Figure 5.765 respectively show the initial and final graph associated with the **Example** slot. Since we use the TRUE arc constraint both graphs are identical.

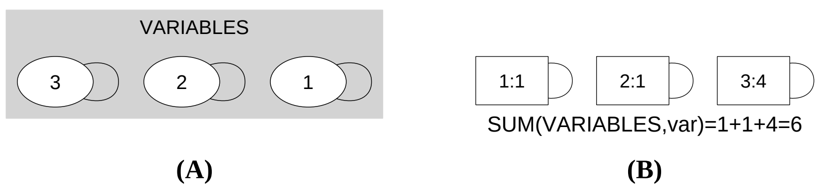



Figure 5.765: Initial and final graph of the sum_ctr constraint

## 5.386 sum_cubes_ctr

DESCRIPTION LINKS

**Origin** Arithmetic constraint.

**Constraint** sum_cubes_ctr(VARIABLES, CTR, VAR)

**Synonyms** sum_cubes, sum_of_cubes, sum_of_cubes_ctr.

**Arguments**

| | | |
|---|---|---|
| VARIABLES | : | collection(var−dvar) |
| CTR | : | atom |
| VAR | : | dvar |

**Restrictions**

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

**Purpose** Constraint the sum of the cubes of a set of domain variables. More precisely, let S denote the sum of the cubes of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

**Example** $(\langle 1, 2, 2\rangle, =, 17)$

The sum_cubes_ctr constraint holds since the condition $1^3 + 2^3 + 2^3 = 17$ is satisfied.

**Typical**

|VARIABLES| $> 1$
range(VARIABLES.var) $> 1$
CTR $\in [=, <, \geq, >, \leq]$

**Symmetry** Items of VARIABLES are permutable.

**Arg. properties**

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$ and minval(VARIABLES.var) $\geq 0$.
- Contractible wrt. VARIABLES when CTR $\in [\geq, >]$ and maxval(VARIABLES.var) $\leq 0$.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$ and minval(VARIABLES.var) $\geq 0$.
- Extensible wrt. VARIABLES when CTR $\in [<, \leq]$ and maxval(VARIABLES.var) $\leq 0$.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

**See also** **common keyword:** sum_ctr, sum_powers4_ctr, sum_powers5_ctr, sum_powers6_ctr, sum_squares_ctr *(sum)*.

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

**constraint type:** predefined constraint, arithmetic constraint.

## 5.387 sum_free

DESCRIPTION LINKS

**Origin** [428]

**Constraint** sum_free(S)

**Argument** S : svar

**Purpose** Impose for all pairs of values (not necessarily distinct) $i$, $j$ of the set S the fact that the sum $i + j$ is not an element of S.

**Example** $(\{1, 3, 5, 9\})$

The sum_free$(\{1, 3, 5, 9\})$ constraint holds since:

- $1 + 1 = 2 \notin$ S, $1 + 3 = 4 \notin$ S, $1 + 5 = 6 \notin$ S, $1 + 9 = 10 \notin$ S.
- $3 + 3 = 6 \notin$ S, $3 + 5 = 8 \notin$ S, $3 + 9 = 12 \notin$ S.
- $5 + 5 = 10 \notin$ S, $5 + 9 = 14 \notin$ S.

**Usage** The sum_free constraint was introduced by W.-J. van Hoeve and A. Sabharwal in order to model in a concise way Schur problems.

- On one hand, the first model has $n$ domain variables $x_i$ $(1 \leq i \leq n)$, where $x_i$ corresponds to the subset in which element $i$ occurs. The constraints $x_i = s \wedge x_j = s \Rightarrow x_{i+j} \neq s$ $(s \in [1, k], i, j \in [1, n], i \leq j, i + j \leq n)$ enforce that the $k$ subsets are sum-free. We have $O(k \cdot n^2)$ such constraints.
- On the other hand, the model proposed by W.-J. van Hoeve and A. Sabharwal represents in an explicit way with a set variable $S_i$ $(1 \leq i \leq n)$ each subset of the partition we are looking for. Now, to express the fact that these $k$ subsets are sum-free they simply use $k$ sum_free constraints of the form sum_free$(S_i)$.

While the two models have the same behaviour when we focus on the number of backtracks the second model is much more efficient from a memory point of view.

**Algorithm** W.-J. van Hoeve and A. Sabharwal have proposed an algorithm that enforces bound-consistency for the sum_free constraint in [428].

**Keywords** **constraint arguments:** unary constraint, constraint involving set variables.

**constraint type:** predefined constraint.

**filtering:** bound-consistency.

**problems:** Schur number.

## 5.388 sum_of_increments

DESCRIPTION LINKS

**Origin** [86]

**Constraint** sum_of_increments(VARIABLES, LIMIT)

**Synonyms** increments_sum, incr_sum, sum_incr, sum_increments.

**Arguments**

| | | |
|---|---|---|
| VARIABLES | : | collection(var−dvar) |
| LIMIT | : | dvar |

**Restrictions**

required(VARIABLES, var)
VARIABLES.var $\geq 0$
LIMIT $\geq 0$

**Purpose**

Given a collection of variables VARIABLES which can only be assigned non negative values, and a variable LIMIT, enforce the condition VARIABLES[1].var $+ \sum_{i=2}^{|\texttt{VARIABLES}|} \max(\texttt{VARIABLES}[i].\texttt{var} - \texttt{VARIABLES}[i-1].\texttt{var}, 0) \leq \texttt{LIMIT}$. VARIABLES[1].var stands from the fact that we assume an additional implicit 0 before the first variable (i.e., VARIABLES[1].var $= \max(\texttt{VARIABLES}[1].\texttt{var} - 0, 0)$).

**Example**

$(\langle 4, 4, 3, 4, 6\rangle, 7)$

The sum_of_increments constraint holds since we have that $4 + \max(4-4,0) + \max(3-4,0) + \max(4-3,0) + \max(6-4,0) \leq 7$.

**Typical**

$|\texttt{VARIABLES}| > 2$
range(VARIABLES.var) $> 1$
maxval(VARIABLES.var) $> 0$
LIMIT $> 0$
LIMIT $\leq$ |VARIABLES|∗range(VARIABLES.var)/2

**Symmetries**

- One and the same constant can be added to VARIABLES.var and to LIMIT.
- Items of VARIABLES can be reversed.
- LIMIT can be increased.

**Arg. properties**

- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

**Usage** The sum_of_increments was initially motivated by the problem of decomposing a matrix of non-negative integers into a positive linear combination of matrices consisting of only zeros and ones, where the ones occur consecutively in each row.

**Algorithm** A $O(|\mathtt{VARIABLES}|)$ bound-consistency filtering algorithm for the sum_of_increments constraint is described in [86].

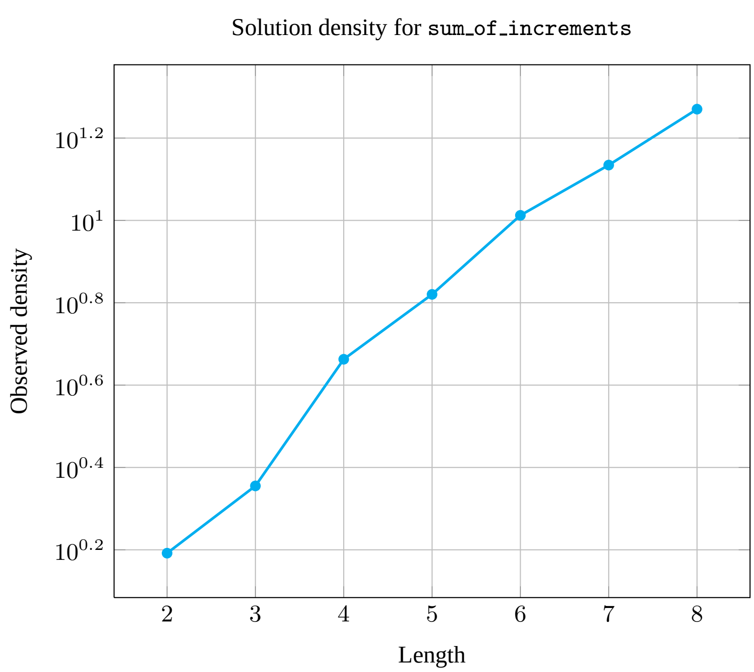
**Reformulation** The following reformulations are provided in [86]. Assuming $\mathtt{VARIABLES}[0].\mathtt{var}$ is defined as 0 (i.e., a zero is added before the first variable of the VARIABLES collection) we have:

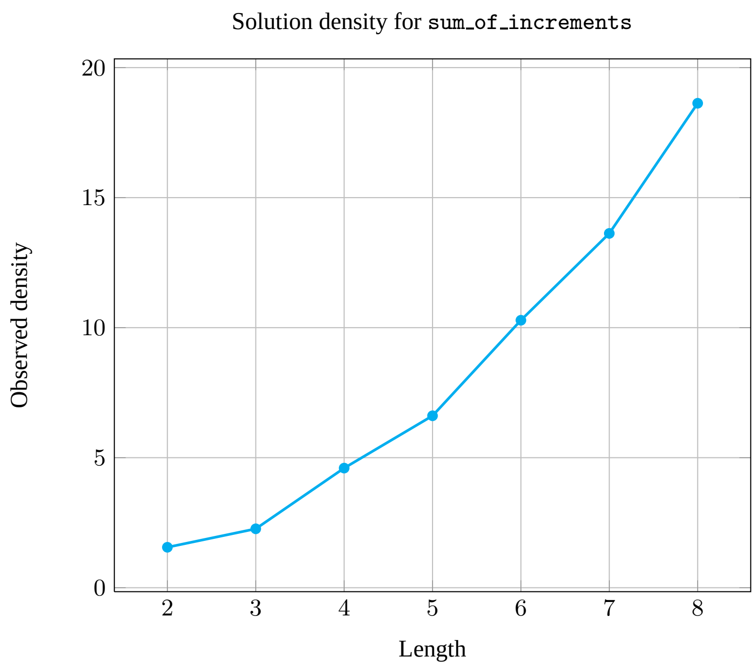
- $\sum_{i=1}^{|\mathtt{VARIABLES}|} S_i \leq \mathtt{LIMIT}$ with $D_i = \mathtt{VARIABLES}[i].\mathtt{var} - \mathtt{VARIABLES}[i-1].\mathtt{var}$ and $S_i = \max(D_i, 0)$ $(1 \leq i \leq |\mathtt{VARIABLES}|)$.
- $\sum_{i=1}^{|\mathtt{VARIABLES}|} S_i \leq \mathtt{LIMIT}$ with $\mathtt{VARIABLES}[i].\mathtt{var} - \mathtt{VARIABLES}[i-1].\mathtt{var} \leq S_i$ and $S_i \in [0, \overline{\mathtt{LIMIT}}]$ $(1 \leq i \leq |\mathtt{VARIABLES}|)$.

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 14 | 145 | 2875 | 51415 | 1210104 | 28573741 | 801944469 |

Number of solutions for sum_of_increments: domains $0..n$

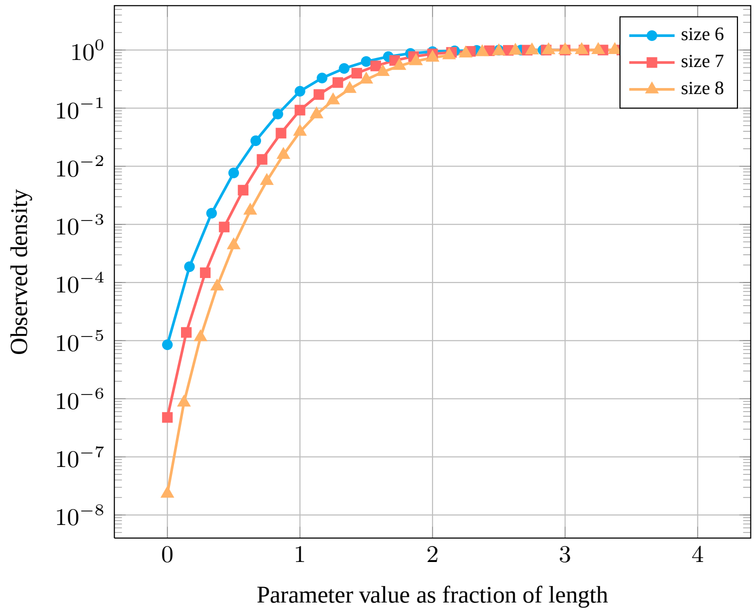



Solution density for sum_of_increments
Observed density
20
15
10
5
0
2
3
4
5
6
7
8
Length

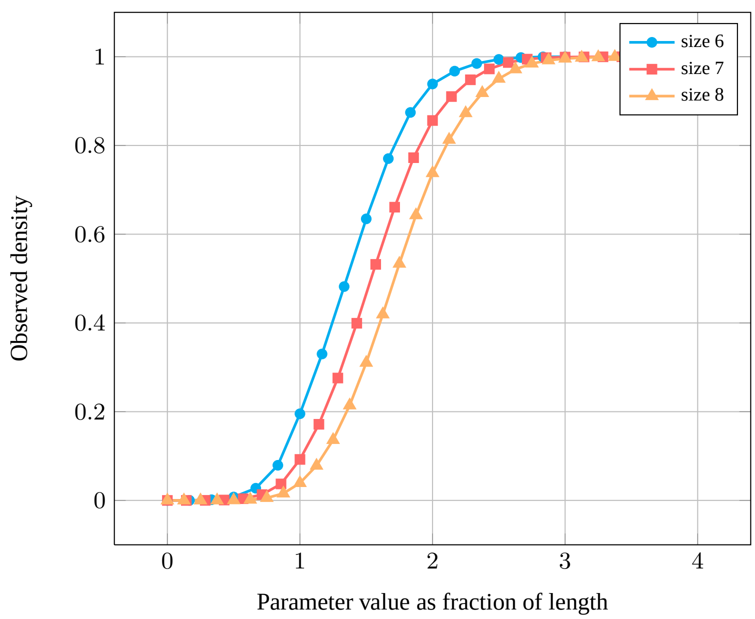
| Length ($n$) | | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Total | | 14 | 145 | 2875 | 51415 | 1210104 | 28573741 | 801944469 |
| Parameter value | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 1 | 4 | 7 | 11 | 16 | 22 | 29 | 37 |
| | 2 | 9 | 23 | 51 | 101 | 183 | 309 | 493 |
| | 3 | - | 54 | 156 | 396 | 904 | 1891 | 3679 |
| | 4 | - | 60 | 375 | 1167 | 3235 | 8135 | 18835 |
| | 5 | - | - | 485 | 2848 | 9318 | 27483 | 74143 |
| | 6 | - | - | 563 | 4263 | 22981 | 77947 | 240751 |
| | 7 | - | - | 608 | 5568 | 38836 | 193742 | 675244 |
| | 8 | - | - | 625 | 6616 | 56703 | 359880 | 1688427 |
| | 9 | - | - | - | 7314 | 74658 | 578511 | 3369015 |
| | 10 | - | - | - | 7650 | 90639 | 837441 | 5865915 |
| | 11 | - | - | - | 7720 | 102875 | 1115687 | 9220695 |
| | 12 | - | - | - | 7755 | 110425 | 1386029 | 13354545 |
| | 13 | - | - | - | - | 113827 | 1619993 | 18051195 |
| | 14 | - | - | - | - | 115857 | 1795694 | 22965651 |
| | 15 | - | - | - | - | 116942 | 1908968 | 27670800 |
| | 16 | - | - | - | - | 117437 | 1988222 | 31755573 |
| | 17 | - | - | - | - | 117612 | 2039616 | 34989993 |
| | 18 | - | - | - | - | 117649 | 2069933 | 37574073 |
| | 19 | - | - | - | - | - | 2085763 | 39526569 |
| | 20 | - | - | - | - | - | 2092817 | 40912205 |
| | 21 | - | - | - | - | - | 2095436 | 41827847 |
| | 22 | - | - | - | - | - | 2096360 | 42386387 |
| | 23 | - | - | - | - | - | 2096822 | 42700112 |
| | 24 | - | - | - | - | - | 2097032 | 42865683 |
| | 25 | - | - | - | - | - | - | 42953199 |
| | 26 | - | - | - | - | - | - | 43002171 |
| | 27 | - | - | - | - | - | - | 43027581 |
| | 28 | - | - | - | - | - | - | 43039551 |
| | 29 | - | - | - | - | - | - | 43044507 |
| | 30 | - | - | - | - | - | - | 43046215 |
| | 31 | - | - | - | - | - | - | 43046656 |
| | 32 | - | - | - | - | - | - | 43046721 |

Solution count for `sum_of_increments`: domains $0..n$

Solution density for `sum_of_increments`




Solution density for `sum_of_increments`




**Keywords** **characteristic of a constraint:** difference, sum.
**constraint type:** predefined constraint.
**filtering:** bound-consistency.

## 5.389 sum_of_weights_of_distinct_values

DESCRIPTION LINKS GRAPH

**Origin** [40]

**Constraint** sum_of_weights_of_distinct_values(VARIABLES, VALUES, COST)

**Synonym** swdv.

**Arguments**

VARIABLES : collection(var−dvar)
VALUES : collection(val−int, weight−int)
COST : dvar

**Restrictions**

required(VARIABLES, var)
$|\mathtt{VALUES}| > 0$
required(VALUES, [val, weight])
VALUES.weight $\geq 0$
distinct(VALUES, val)
in_attr(VARIABLES, var, VALUES, val)
COST $\geq 0$

**Purpose** All variables of the VARIABLES collection take a value in the VALUES collection. In addition COST is the sum of the weight attributes associated with the distinct values taken by the variables of VARIABLES.

**Example**

$$\left(\begin{array}{l}\langle 1,6,1\rangle, \\ \left\langle\begin{array}{ll}\mathtt{val}-1 & \mathtt{weight}-5, \\ \mathtt{val}-2 & \mathtt{weight}-3, \\ \mathtt{val}-6 & \mathtt{weight}-7\end{array}\right\rangle, 12\end{array}\right)$$

The sum_of_weights_of_distinct_values constraint holds since its last argument COST $= 12$ is equal to the sum $5+7$ of the weights of the values 1 and 6 that occur within the $\langle 1,6,1\rangle$ collection.

**Typical**

$|\mathtt{VARIABLES}| > 1$
range(VARIABLES.var) $> 1$
$|\mathtt{VALUES}| > 1$
range(VALUES.weight) $> 1$
maxval(VALUES.weight) $> 0$

**Symmetries**

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- Items of VALUES are permutable.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

**Arg. properties**

Functional dependency: COST determined by VARIABLES and VALUES.

**See also**

**attached to cost variant:** nvalue *(all values have a weight of* 1*)*.

**common keyword:** global_cardinality_with_costs, minimum_weight_alldifferent, weighted_partial_alldiff *(weighted assignment)*.

**Keywords**

**application area:** assignment.

**constraint arguments:** pure functional dependency.

**constraint type:** relaxation.

**filtering:** cost filtering constraint.

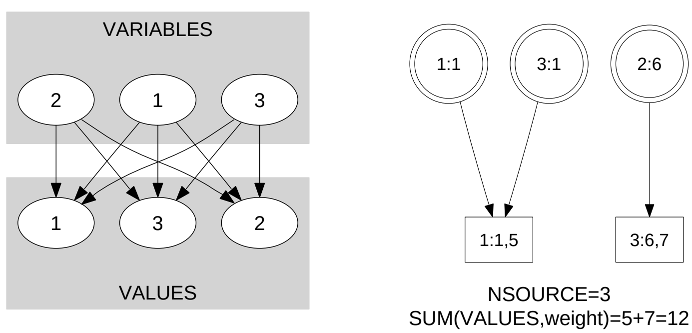
**modelling:** functional dependency.

**problems:** domination, weighted assignment, facilities location problem.

| | |
|---|---|
| **Arc input(s)** | VARIABLES VALUES |
| **Arc generator** | $PRODUCT \mapsto$ collection(variables, values) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables.var = values.val |
| **Graph property(ies)** | • **NSOURCE**= \|VARIABLES\|<br>• **SUM**(VALUES, weight) = COST |

**Signature** Since we use the $PRODUCT$ arc generator, the number of sources of the final graph cannot exceed the number of sources of the initial graph. Since the initial graph contains |VARIABLES| sources, this number is an upper bound of the number of sources of the final graph. Therefore we can rewrite **NSOURCE** = |VARIABLES| to **NSOURCE** $\geq$ |VARIABLES| and simplify $\overline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$.

Parts (A) and (B) of Figure 5.766 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** graph property, the source vertices of the final graph are shown in a double circle. Since we also use the **SUM** graph property we show the vertices from which we compute the total cost in a box.




**(A)** **(B)**

Figure 5.766: Initial and final graph of the sum_of_weights_of_distinct_values constraint

## 5.390 sum_powers4_ctr

DESCRIPTION LINKS

**Origin** Arithmetic constraint.

**Constraint** sum_powers4_ctr(VARIABLES, CTR, VAR)

**Synonyms** sum_powers4, sum_of_powers4, sum_of_powers4_ctr.

**Arguments**

| | | |
|---|---|---|
| VARIABLES | : | collection(var−dvar) |
| CTR | : | atom |
| VAR | : | dvar |

**Restrictions**

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

**Purpose** Constraint the sum of the power of four of a set of domain variables. More precisely, let S denote the sum of the power of four of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

**Example** $(\langle 1, 1, 2\rangle, =, 18)$

The sum_powers4_ctr constraint holds since the condition $1^4 + 1^4 + 2^4 = 18$ is satisfied.

**Typical**

$|\mathtt{VARIABLES}| > 1$
range(VARIABLES.var) $> 1$
CTR $\in [=, <, \geq, >, \leq]$

**Symmetry** Items of VARIABLES are permutable.

**Arg. properties**

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

**See also** **common keyword:** sum_ctr, sum_cubes_ctr, sum_powers5_ctr, sum_powers6_ctr, sum_squares_ctr *(sum)*.

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

**constraint type:** predefined constraint, arithmetic constraint.

## 5.391 sum_powers5_ctr

DESCRIPTION LINKS

**Origin** Arithmetic constraint.

**Constraint** sum_powers5_ctr(VARIABLES, CTR, VAR)

**Synonyms** sum_powers5, sum_of_powers5, sum_of_powers5_ctr.

**Arguments**

| | | |
|---|---|---|
| VARIABLES | : | collection(var−dvar) |
| CTR | : | atom |
| VAR | : | dvar |

**Restrictions**

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

**Purpose** Constraint the sum of the power of five of a set of domain variables. More precisely, let S denote the sum of the power of five of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

**Example** $(\langle 1, 1, 2\rangle, =, 34)$

The sum_powers5_ctr constraint holds since the condition $1^5 + 1^5 + 2^5 = 34$ is satisfied.

**Typical**

|VARIABLES| $> 1$
range(VARIABLES.var) $> 1$
CTR $\in [=, <, \geq, >, \leq]$

**Symmetry** Items of VARIABLES are permutable.

**Arg. properties**

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$ and minval(VARIABLES.var) $\geq 0$.
- Contractible wrt. VARIABLES when CTR $\in [\geq, >]$ and maxval(VARIABLES.var) $\leq 0$.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$ and minval(VARIABLES.var) $\geq 0$.
- Extensible wrt. VARIABLES when CTR $\in [<, \leq]$ and maxval(VARIABLES.var) $\leq 0$.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

**See also** **common keyword:** sum_ctr, sum_cubes_ctr, sum_powers4_ctr, sum_powers6_ctr, sum_squares_ctr *(sum)*.

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

**constraint type:** predefined constraint, arithmetic constraint.

## 5.392 sum_powers6_ctr

DESCRIPTION LINKS

**Origin** Arithmetic constraint.

**Constraint** sum_powers6_ctr(VARIABLES, CTR, VAR)

**Synonyms** sum_powers6, sum_of_powers6, sum_of_powers6_ctr.

**Arguments**

| | | |
|---|---|---|
| VARIABLES | : | collection(var−dvar) |
| CTR | : | atom |
| VAR | : | dvar |

**Restrictions**

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

**Purpose** Constraint the sum of the power of six of a set of domain variables. More precisely, let S denote the sum of the power of six of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

**Example** $(\langle 1, 1, 2\rangle, =, 66)$

The sum_powers6_ctr constraint holds since the condition $1^6 + 1^6 + 2^6 = 66$ is satisfied.

**Typical**

$|$VARIABLES$| > 1$
range(VARIABLES.var) $> 1$
CTR $\in [=, <, \geq, >, \leq]$

**Symmetry** Items of VARIABLES are permutable.

**Arg. properties**

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

**See also** **common keyword:** sum_ctr, sum_cubes_ctr, sum_powers4_ctr, sum_powers5_ctr, sum_squares_ctr *(sum)*.

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

**constraint type:** predefined constraint, arithmetic constraint.

## 5.393 sum_set

DESCRIPTION LINKS GRAPH

**Origin** H. Cambazard

**Constraint** sum_set(SV, VALUES, CTR, VAR)

**Arguments**

| | | |
|---|---|---|
| SV | : | svar |
| VALUES | : | collection(val−int, coef−int) |
| CTR | : | atom |
| VAR | : | dvar |

**Restrictions**

required(VALUES, [val, coef])
distinct(VALUES, val)
VALUES.coef $\geq 0$
CTR $\in [=, \neq, <, \geq, >, \leq]$

**Purpose** Let SUM denote the sum of the coef attributes of the VALUES collection for which the corresponding values val occur in the set SV. Enforce the following constraint to hold: SUM CTR VAR.

**Example**

$$\left(\begin{array}{l}\{2,3,6\},\\ \left\langle\begin{array}{ll}\texttt{val}-2 & \texttt{coef}-7,\\ \texttt{val}-9 & \texttt{coef}-1,\\ \texttt{val}-5 & \texttt{coef}-7,\\ \texttt{val}-6 & \texttt{coef}-2\end{array}\right\rangle, =, 9\end{array}\right)$$

The sum_set constraint holds since the sum of the coef attributes $7 + 2$ for which the corresponding val attribute belongs to the first argument SV $= \{2, 3, 6\}$ is equal (i.e., since CTR is set to $=$) to its last argument VAR $= 9$.

**Typical**

$|\text{VALUES}| > 1$
VALUES.coef $> 0$
CTR $\in [=, <, \geq, >, \leq]$

**Symmetry** Items of VALUES are permutable.

**Systems** weights in **Gecode**.

**See also** **common keyword:** sum, sum_ctr *(sum)*.

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

**constraint arguments:** binary constraint, constraint involving set variables.

**constraint type:** arithmetic constraint.

| | |
|---|---|
| **Arc input(s)** | VALUES |
| **Arc generator** | $SELF \mapsto$ collection(values) |
| **Arc arity** | 1 |
| **Arc constraint(s)** | in_set(values.val, SV) |
| **Graph property(ies)** | **SUM**(VALUES, coef) CTR VAR |

**Graph model** Parts (A) and (B) of Figure 5.767 respectively show the initial and final graph associated with the **Example** slot.

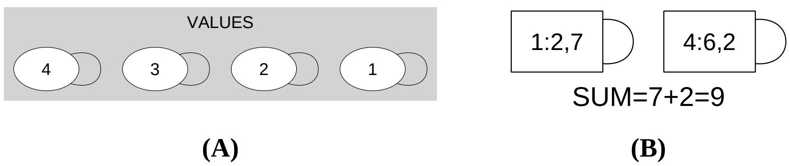



Figure 5.767: Initial and final graph of the sum_set constraint

## 5.394 sum_squares_ctr

DESCRIPTION LINKS

**Origin** Arithmetic constraint.

**Constraint** sum_squares_ctr(VARIABLES, CTR, VAR)

**Synonyms** sum_squares, sum_of_squares, sum_of_squares_ctr.

**Arguments**

| | | |
|---|---|---|
| VARIABLES | : | collection(var−dvar) |
| CTR | : | atom |
| VAR | : | dvar |

**Restrictions**

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

**Purpose** Constraint the sum of the squares of a set of domain variables. More precisely, let S denote the sum of the squares of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

**Example** $(\langle 1, 1, 4\rangle, =, 18)$

The sum_squares_ctr constraint holds since the condition $1^2 + 1^2 + 4^2 = 18$ is satisfied.

**Typical**

$|$VARIABLES$| > 1$
range(VARIABLES.var) $> 1$
CTR $\in [=, <, \geq, >, \leq]$

**Symmetry** Items of VARIABLES are permutable.

**Arg. properties**

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

**See also** **common keyword:** sum_ctr, sum_cubes_ctr, sum_powers4_ctr, sum_powers5_ctr, sum_powers6_ctr *(sum)*.

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

**constraint type:** predefined constraint, arithmetic constraint.

**Cond. implications**

- sum_squares_ctr(VARIABLES, CTR, VAR)
  with VARIABLES.var $\geq -1$
  and VARIABLES.var $\leq 1$
  **implies** sum_powers4_ctr(VARIABLES, CTR, VAR)
  when VARIABLES.var $\geq -1$
  and VARIABLES.var $\leq 1$.

- sum_squares_ctr(VARIABLES, CTR, VAR)
  with VARIABLES.var $\geq -1$
  and VARIABLES.var $\leq 1$
  **implies** sum_powers6_ctr(VARIABLES, CTR, VAR)
  when VARIABLES.var $\geq -1$
  and VARIABLES.var $\leq 1$.

## 5.395 symmetric

DESCRIPTION LINKS GRAPH

**Origin** [142]

**Constraint** symmetric(NODES)

**Argument** NODES : collection(index−int, succ−svar)

**Restrictions**
required(NODES, [index, succ])
NODES.index $\geq 1$
NODES.index $\leq$ |NODES|
distinct(NODES, index)

**Purpose** Consider a digraph $G$ described by the NODES collection. Select a subset of arcs of $G$ so that the corresponding graph is symmetric (i.e., if there is an arc from $i$ to $j$, there is also an arc from $j$ to $i$).

**Example**

$$\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\{1,2,3\}, \\ \texttt{index}-2 & \texttt{succ}-\{1,3\}, \\ \texttt{index}-3 & \texttt{succ}-\{1,2\}, \\ \texttt{index}-4 & \texttt{succ}-\{5,6\}, \\ \texttt{index}-5 & \texttt{succ}-\{4\}, \\ \texttt{index}-6 & \texttt{succ}-\{4\} \end{array}\right\rangle\right)$$

The symmetric constraint holds since the NODES collection depicts a symmetric graph.

**Typical** |NODES| $> 2$

**Symmetry** Items of NODES are permutable.

**Algorithm** The filtering algorithm for the symmetric constraint is given in [142, page 87]. It removes (respectively imposes) the arcs $(i, j)$ for which the arc $(j, i)$ is not present (respectively is present). It has an overall complexity of $O(n+m)$ where $n$ and $m$ respectively denote the number of vertices and the number of arcs of the initial graph.

**See also** **common keyword:** connected *(symmetric)*.

**used in graph description:** in_set.

**Keywords** **constraint arguments:** constraint involving set variables.

**constraint type:** graph constraint.

**final graph structure:** symmetric.

| | |
|---|---|
| **Arc input(s)** | `NODES` |
| **Arc generator** | $CLIQUE \mapsto$ `collection`(`nodes1`, `nodes2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `in_set`(`nodes2.index`, `nodes1.succ`) |
| **Graph class** | `SYMMETRIC` |

**Graph model** Part (A) of Figure 5.768 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the `succ` attribute of a given vertex. Part (B) of Figure 5.768 gives the final graph associated with the **Example** slot.

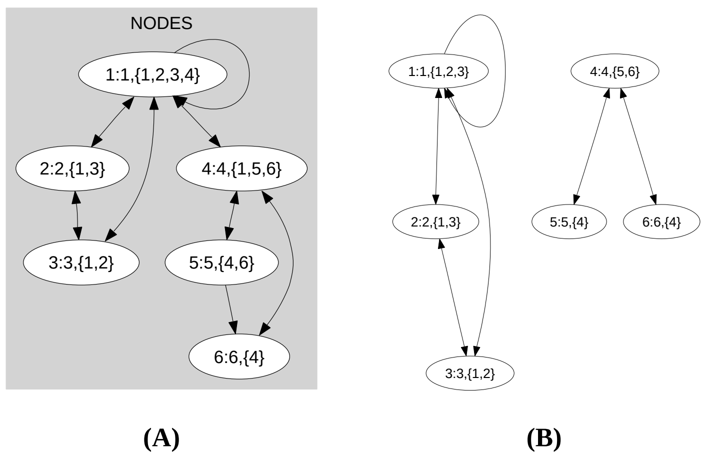



Figure 5.768: Initial and final graph of the `symmetric` set constraint

## 5.396 symmetric_alldifferent

DESCRIPTION LINKS GRAPH

**Origin** [345]

**Constraint** `symmetric_alldifferent(NODES)`

**Synonyms** `symmetric_alldiff`, `symmetric_alldistinct`, `symm_alldifferent`, `symm_alldiff`, `symm_alldistinct`, `one_factor`, `two_cycle`.

**Argument** NODES : collection(index−int, succ−dvar)

**Restrictions**

$|\mathtt{NODES}| \bmod 2 = 0$
required(NODES, [index, succ])
NODES.index $\geq 1$
NODES.index $\leq |\mathtt{NODES}|$
distinct(NODES, index)
NODES.succ $\geq 1$
NODES.succ $\leq |\mathtt{NODES}|$

**Purpose** All variables associated with the `succ` attribute of the `NODES` collection should be pairwise distinct. In addition enforce the following condition: if variable $\mathtt{NODES}[i]$.succ takes value $j$ with $j \neq i$ then variable $\mathtt{NODES}[j]$.succ takes value $i$. This can be interpreted as a graph-covering problem where one has to cover a digraph $G$ with circuits of length two in such a way that each vertex of $G$ belongs to a single circuit.

**Example**

$$\left(\left\langle\begin{array}{ll} \mathtt{index}-1 & \mathtt{succ}-3, \\ \mathtt{index}-2 & \mathtt{succ}-4, \\ \mathtt{index}-3 & \mathtt{succ}-1, \\ \mathtt{index}-4 & \mathtt{succ}-2 \end{array}\right\rangle\right)$$

The `symmetric_alldifferent` constraint holds since:

- $\mathtt{NODES}[1].\mathtt{succ} = 3 \Leftrightarrow \mathtt{NODES}[3].\mathtt{succ} = 1$,
- $\mathtt{NODES}[2].\mathtt{succ} = 4 \Leftrightarrow \mathtt{NODES}[4].\mathtt{succ} = 2$.

**All solutions** Figure 5.769 gives all solutions to the following non ground instance of the `symmetric_alldifferent` constraint: $\mathtt{S}_1 \in [1,4]$, $\mathtt{S}_2 \in [1,3]$, $\mathtt{S}_3 \in [1,4]$, $\mathtt{S}_4 \in [1,3]$, `symmetric_alldifferent`$(\langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4\rangle)$.

**Typical** $|\mathtt{NODES}| \geq 4$

**Symmetry** Items of `NODES` are permutable.

**Usage** As it was reported in [345, page 420], this constraint is useful to express matches between persons or between teams. The `symmetric_alldifferent` constraint also appears implicitly in the *cycle cover problem* and corresponds to the four conditions given in section 1 *Modeling the Cycle Cover Problem* of [308].

① $(\langle 2_1, 1_2, 4_3, 3_4 \rangle)$
② $(\langle 4_1, 3_2, 2_3, 1_4 \rangle)$

Figure 5.769: All solutions corresponding to the non ground example of the `symmetric_alldifferent` constraint of the **All solutions** slot (the `index` attribute is displayed as indices of the `succ` attribute)

**Remark** This constraint is referenced under the name `one_factor` in [211] as well as in [409]. From a modelling point of view this constraint can be expressed with the `cycle` constraint [41] where one imposes the additional condition that each cycle has only two nodes.

**Algorithm** A filtering algorithm for the `symmetric_alldifferent` constraint was proposed by J.-C. Régin in [345]. It achieves arc-consistency and its running time is dominated by the complexity of finding all edges that do not belong to any maximum cardinality matching in an undirected $n$-vertex, $m$-edge graph, i.e., $O(m \cdot n)$.

For the soft case of the `symmetric_alldifferent` constraint where the cost is the minimum number of variables to assign differently in order to get back to a solution, a filtering algorithm achieving arc-consistency is described in [131, 130]. It has a complexity of $O(p \cdot m)$, where $p$ is the number of maximal extreme sets in the value graph associated with the constraint and $m$ is the number of edges. It iterates over extreme sets and not over vertices as in the algorithm due to J.-C. Régin.

**Reformulation** The `symmetric_alldifferent`(NODES) constraint can be expressed in term of a conjunction of $|\texttt{NODES}|^2$ reified constraints of the form $\texttt{NODES}[i].\texttt{succ} = j \Leftrightarrow \texttt{NODES}[j].\texttt{succ} = i$ $(1 \leq i, j \leq |\texttt{NODES}|)$. The `symmetric_alldifferent` constraint can also be reformulated as an `inverse` constraint as shown below:

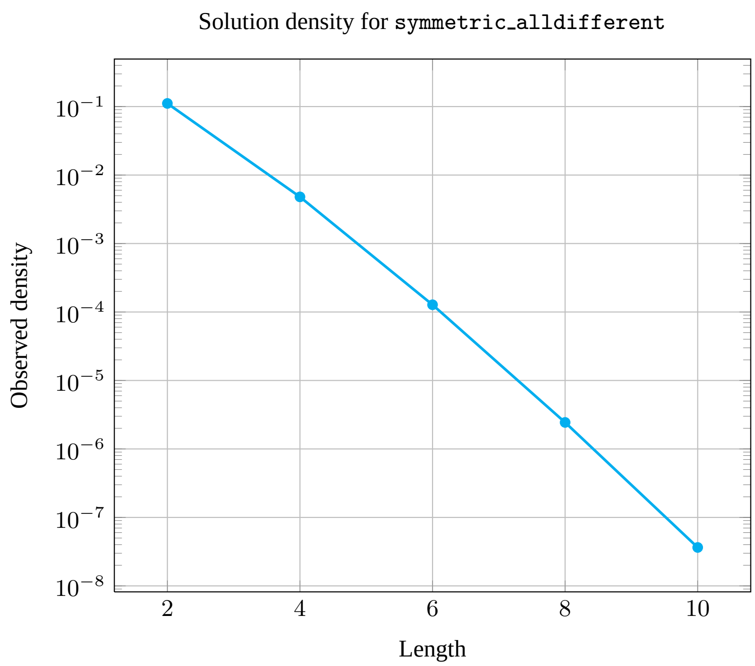
$$\texttt{symmetric\_alldifferent}\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-s_1, \\ \texttt{index}-2 & \texttt{succ}-s_2, \\ \vdots & \vdots \\ \texttt{index}-n & \texttt{succ}-s_n \end{array}\right\rangle\right)$$

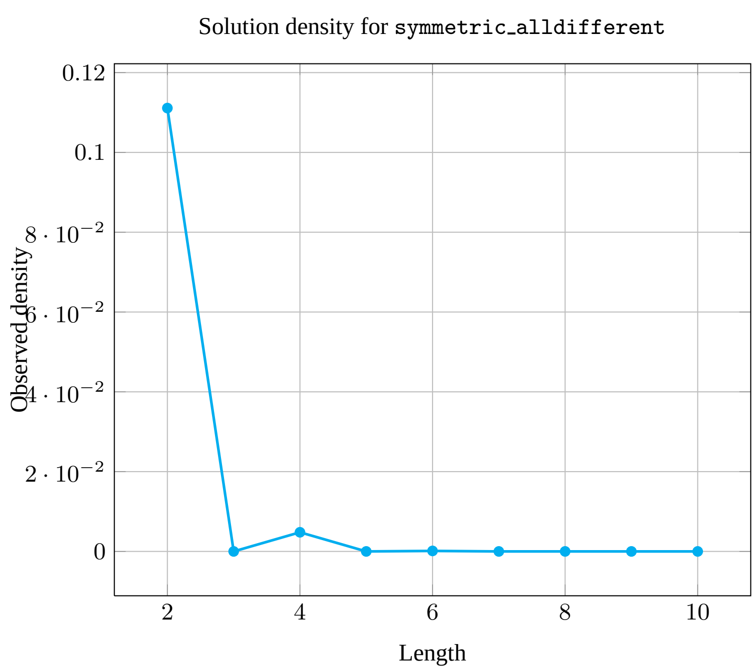
$$\texttt{inverse}\left(\left\langle\begin{array}{lll} \texttt{index}-1 & \texttt{succ}-s_1 & \texttt{pred}-s_1, \\ \texttt{index}-2 & \texttt{succ}-s_2 & \texttt{pred}-s_2, \\ \vdots & \vdots & \vdots \\ \texttt{index}-n & \texttt{succ}-s_n & \texttt{pred}-s_n \end{array}\right\rangle\right)$$

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Solutions | 1 | 0 | 3 | 0 | 15 | 0 | 105 | 0 | 945 |

Number of solutions for `symmetric_alldifferent`: domains $0..n$




**See also** **common keyword:** `alldifferent`, `cycle`, `inverse` *(permutation)*.

**implies:** `derangement`, `symmetric_alldifferent_except_0`, `symmetric_alldifferent_loop`.

**implies (items to collection):** k_alldifferent, lex_alldifferent.

**related:** roots.

**Keywords**

**application area:** sport timetabling.

**characteristic of a constraint:** all different, disequality.

**combinatorial object:** permutation, involution, matching.

**constraint type:** graph constraint, timetabling constraint, graph partitioning constraint.

**filtering:** arc-consistency.

**final graph structure:** circuit.

**modelling:** cycle.

**Cond. implications**

- symmetric_alldifferent(NODES)
  **implies** balance_cycle(BALANCE, NODES)
  when $\mathtt{BALANCE} = 0$.
- symmetric_alldifferent(NODES)
  **implies** cycle(NCYCLE, NODES)
  when $2 * \mathtt{NCYCLE} = |\mathtt{NODES}|$.
- symmetric_alldifferent(NODES)
  **implies** permutation(VARIABLES : NODES).

| | |
|---|---|
| **Arc input(s)** | `NODES` |
| **Arc generator** | $CLIQUE(\neq) \mapsto$`collection`(`nodes1`, `nodes2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • `nodes1.succ` = `nodes2.index`<br>• `nodes2.succ` = `nodes1.index` |
| **Graph property(ies)** | **NARC**= \|`NODES`\| |

**Graph model** In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices.

Parts (A) and (B) of Figure 5.770 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

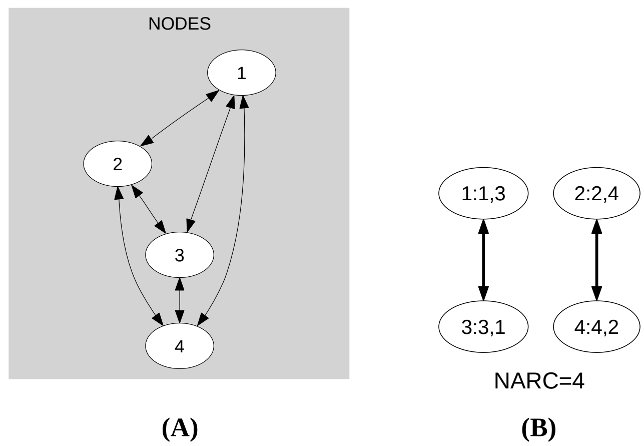



Figure 5.770: Initial and final graph of the `symmetric_alldifferent` constraint

**Signature** Since all the `index` attributes of the `NODES` collection are distinct, and because of the first condition `nodes1.succ` = `nodes2.index` of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to the maximum number of vertices |`NODES`| of the final graph. So we can rewrite **NARC** = |`NODES`| to **NARC** $\geq$ |`NODES`| and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

## 5.397 symmetric_alldifferent_except_0

DESCRIPTION LINKS

**Origin** Derived from symmetric_alldifferent

**Constraint** symmetric_alldifferent_except_0(NODES)

**Synonyms** symmetric_alldiff_except_0, symmetric_alldistinct_except_0, symm_alldifferent_except_0, symm_alldiff_except_0, symm_alldistinct_except_0.

**Argument** NODES : collection(index−int, succ−dvar)

**Restrictions**
required(NODES, [index, succ])
NODES.index $\geq 1$
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ $\geq 0$
NODES.succ $\leq$ |NODES|

**Purpose**

Enforce the following three conditions:

1. $\forall i \in [1, |\texttt{NODES}|], \forall j \in [1, |\texttt{NODES}|], (j \neq i)$: $\texttt{NODES}[i].\texttt{succ} = 0 \vee \texttt{NODES}[j].\texttt{succ} = 0 \vee \texttt{NODES}[i].\texttt{succ} \neq \texttt{NODES}[j].\texttt{succ}$.
2. $\forall i \in [1, |\texttt{NODES}|] : \texttt{NODES}[i].\texttt{succ} \neq i$.
3. $\texttt{NODES}[i].\texttt{succ} = j \wedge j \neq i \wedge j \neq 0 \Leftrightarrow \texttt{NODES}[j].\texttt{succ} = i \wedge i \neq j \wedge i \neq 0$.

**Example**

$$\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-3, \\ \texttt{index}-2 & \texttt{succ}-0, \\ \texttt{index}-3 & \texttt{succ}-1, \\ \texttt{index}-4 & \texttt{succ}-0 \end{array}\right\rangle\right)$$

The symmetric_alldifferent_except_0 constraint holds since:

- $\texttt{NODES}[1].\texttt{succ} = 3 \Leftrightarrow \texttt{NODES}[3].\texttt{succ} = 1$,
- $\texttt{NODES}[2].\texttt{succ} = 0$ and value 2 is not assigned to any variable.
- $\texttt{NODES}[4].\texttt{succ} = 0$ and value 4 is not assigned to any variable.

Given 3 successor variables that have to be assigned a value in interval $[0, 3]$, the solutions to the symmetric_alldifferent_except_0 ($\langle \texttt{index}-1\ \texttt{succ}-s_1, \texttt{index}-2\ \texttt{succ}-s_2, \texttt{index}-3\ \texttt{succ}-s_3\rangle$) constraint are $\langle 1\ 0, 2\ 0, 3\ 0\rangle$, $\langle 1\ 0, 2\ 3, 3\ 2\rangle$, $\langle 1\ 2, 2\ 1, 3\ 0\rangle$, and $\langle 1\ 3, 2\ 0, 3\ 1\rangle$.

Given 4 successor variables that have to be assigned a value in interval $[0, 3]$, the solutions to the symmetric_alldifferent_except_0 ($\langle \texttt{index}-1\ \texttt{succ}-s_1, \texttt{index}-2\ \texttt{succ}-s_2, \texttt{index}-3\ \texttt{succ}-s_3, \texttt{index}-4\ \texttt{succ}-s_4\rangle$) constraint are $\langle 1\ 0, 2\ 0, 3\ 0, 4\ 0\rangle$, $\langle 1\ 0, 2\ 0, 3\ 4, 4\ 3\rangle$, $\langle 1\ 0, 2\ 3, 3\ 2, 4\ 0\rangle$, $\langle 1\ 0, 2\ 4, 3\ 0, 4\ 2\rangle$, $\langle 1\ 2, 2\ 1, 3\ 0, 4\ 0\rangle$, $\langle 1\ 2, 2\ 1, 3\ 4, 4\ 3\rangle$, $\langle 1\ 3, 2\ 0, 3\ 1, 4\ 0\rangle$, $\langle 1\ 3, 2\ 4, 3\ 1, 4\ 2\rangle$, $\langle 1\ 4, 2\ 0, 3\ 0, 4\ 1\rangle$, $\langle 1\ 4, 2\ 3, 3\ 2, 4\ 1\rangle$.

**All solutions** Figure 5.771 gives all solutions to the following non ground instance of the `symmetric_alldifferent_except_0` constraint: $\mathtt{S}_1 \in [0,5]$, $\mathtt{S}_2 \in [1,3]$, $\mathtt{S}_3 \in [1,4]$, $\mathtt{S}_4 \in [0,3]$, $\mathtt{S}_5 \in [0,2]$, `symmetric_alldifferent_except_0`$(\langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4, 5\ \mathtt{S}_5\rangle)$.

① $(\langle 0_1, 3_2, 2_3, 0_4, 0_5\rangle)$
② $(\langle 2_1, 1_2, 4_3, 3_4, 0_5\rangle)$
③ $(\langle 4_1, 3_2, 2_3, 1_4, 0_5\rangle)$
④ $(\langle 5_1, 3_2, 2_3, 0_4, 1_5\rangle)$

Figure 5.771: All solutions corresponding to the non ground example of the `symmetric_alldifferent_except_0` constraint of the **All solutions** slot (the `index` attribute is displayed as indices of the `succ` attribute)

**Typical**

$|\mathtt{NODES}| \geq 4$
$\mathtt{minval}(\mathtt{NODES.succ}) = 0$
$\mathtt{maxval}(\mathtt{NODES.succ}) > 0$

**Symmetry** Items of `NODES` are permutable.

**Usage** Within the context of sport scheduling, $\mathtt{NODES}[i].\mathtt{succ} = j$ ($i \neq 0, j \neq 0, i \neq j$) is interpreted as the fact that team $i$ plays against team $j$, while $\mathtt{NODES}[i].\mathtt{succ} = 0$ ($i \neq 0$) is interpreted as the fact that team $i$ does not play at all.

**Algorithm** An arc-consistency filtering algorithm for the `symmetric_alldifferent_except_0` constraint is described in [131, 130]. The algorithm is based on the following facts:

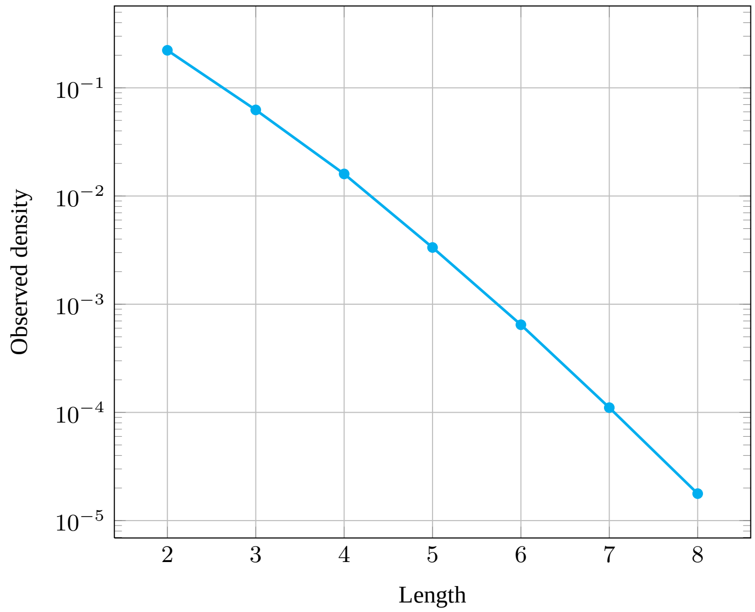
- First, one can map solutions to the `symmetric_alldifferent_except_0` constraint to perfect $(g, f)$-matchings in a non-bipartite graph derived from the domain of the variables of the constraint where $g(x) = 0$, $f(x) = 1$ for vertices $x$ which have 0 in their domain, and $g(x) = f(x) = 1$ for all the remaining vertices. A *perfect* $(g, f)$-*matching* $\mathcal{M}$ of a graph is a subset of edges such that every vertex $x$ is incident with the number of edges in $\mathcal{M}$ between $g(x)$ and $f(x)$.
- Second, Gallai-Edmonds decomposition [179, 150] allows to find out all edges that do not belong to any perfect $(g, f)$-matchings, and therefore prune the corresponding variables.

**Counting**

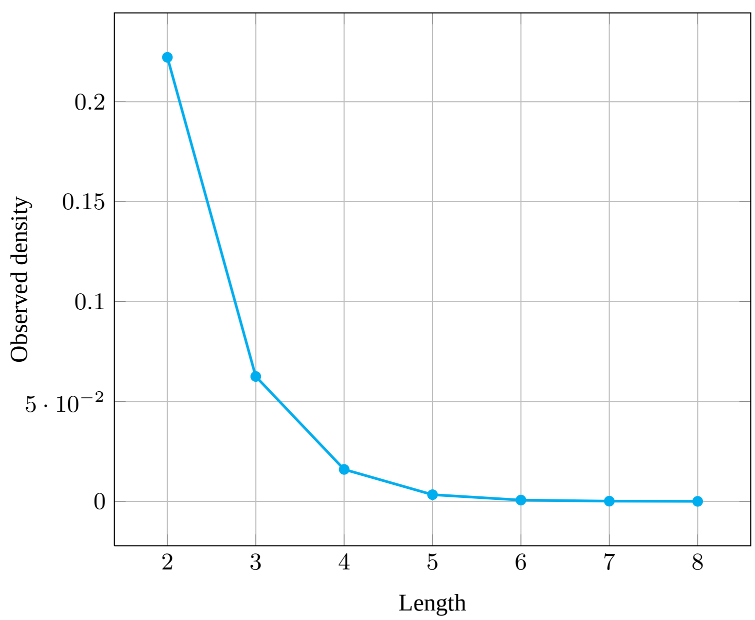
| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 2 | 4 | 10 | 26 | 76 | 232 | 764 |

Number of solutions for `symmetric_alldifferent_except_0`: domains $0..n$

Solution density for `symmetric_alldifferent_except_0`




Solution density for `symmetric_alldifferent_except_0`




**See also** **implied by:** `symmetric_alldifferent`.

**implies (items to collection):** `k_alldifferent`, `lex_alldifferent`.

**Keywords** **application area:** sport timetabling.

**characteristic of a constraint:** joker value.

**combinatorial object:** matching.

**constraint type:** predefined constraint, timetabling constraint.

**Cond. implications** symmetric_alldifferent_except_0(NODES)
**implies** alldifferent_except_0(VARIABLES : NODES).

# 5.398 symmetric_alldifferent_loop

DESCRIPTION LINKS GRAPH

**Origin** Derived from `symmetric_alldifferent`

**Constraint** `symmetric_alldifferent_loop(NODES)`

**Synonyms** `symmetric_alldiff_loop`, `symmetric_alldistinct_loop`, `symm_alldifferent_loop`, `symm_alldiff_loop`, `symm_alldistinct_loop`.

**Argument** `NODES` : `collection(index-int, succ-dvar)`

**Restrictions**
`required(NODES, [index, succ])`
`NODES.index` $\geq 1$
`NODES.index` $\leq$ `|NODES|`
`distinct(NODES, index)`
`NODES.succ` $\geq 1$
`NODES.succ` $\leq$ `|NODES|`

**Purpose** All variables associated with the `succ` attribute of the `NODES` collection should be pairwise distinct. In addition enforce the following condition: if variable `NODES`$[i]$`.succ` is assigned value $j$ then variable `NODES`$[j]$`.succ` is assigned value $i$. Note that $i$ and $j$ are not necessarily distinct. This can be interpreted as a graph-covering problem where one has to cover a digraph $G$ with circuits of length two or one in such a way that each vertex of $G$ belongs to a single circuit.

**Example**

$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{succ}-4,\\ \texttt{index}-3 & \texttt{succ}-3,\\ \texttt{index}-4 & \texttt{succ}-2\end{array}\right\rangle\right)$$

The `symmetric_alldifferent_loop` constraint holds since:

- We have two loops respectively corresponding to `NODES`$[1]$`.succ` $=1$ and `NODES`$[3]$`.succ` $=3$.
- We have one circuit of length 2 corresponding to `NODES`$[2]$`.succ` $=4 \Leftrightarrow$ `NODES`$[4]$`.succ` $=2$.

Figure 5.772 provides a second example involving a `symmetric_alldifferent_loop` constraint.

**All solutions** Figure 5.773 gives all solutions to the following non ground instance of the `symmetric_alldifferent_loop` constraint: $\mathtt{S}_1 \in [2,5]$, $\mathtt{S}_2 \in [1,3]$, $\mathtt{S}_3 \in [1,4]$, $\mathtt{S}_4 \in [2,4]$, $\mathtt{S}_5 \in [1,5]$, `symmetric_alldifferent_loop`$(\langle 1\,\mathtt{S}_1, 2\,\mathtt{S}_2, 3\,\mathtt{S}_3, 4\,\mathtt{S}_4, 5\,\mathtt{S}_5\rangle)$.

**Typical** `|NODES|` $\geq 4$

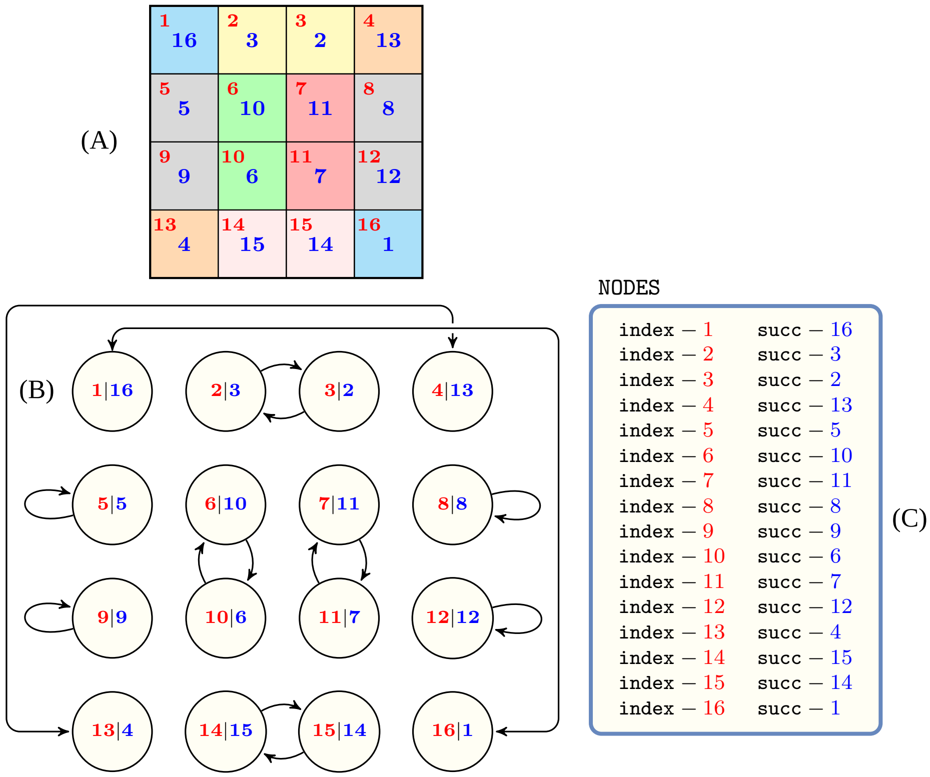



Figure 5.772: (A) Magic square Duerer where cells that belong to a same cycle are coloured identically by a colour different from grey; each cell has an *index* in its upper left corner (in red) and a *value* (in blue). (B) Corresponding graph where there is an arc from node $i$ to node $j$ if and only if the value of cell $i$ is equal to the index of cell $j$. (C) Collection of nodes passed to the `symmetric_alldifferent_loop` constraint: the four self-loops of the graph correspond to the four grey cells of the magic square such that the value of the cell (in blue) is equal to the index of the cell (in red).

**Symmetry**

Items of `NODES` are permutable.

**Algorithm**

An arc-consistency filtering algorithm for the `symmetric_alldifferent_loop` constraint is described in [131, 130]. The algorithm is based on the following ideas:

- First, one can map solutions of the `symmetric_alldifferent_loop` constraint to perfect $(g, f)$-matchings in a non-bipartite graph derived from the domain of the variables of the constraint where $g(x) = 0$, $f(x) = 1$ for vertices $x$ which have a self-loop, and $g(x) = f(x) = 1$ for all the remaining vertices. A *perfect* $(g, f)$-*matching* $\mathcal{M}$ of a graph is a subset of edges such that every vertex $x$ is incident with the number of edges in $\mathcal{M}$ between $g(x)$ and $f(x)$.
- Second, Gallai-Edmonds decomposition [179, 150] allows to find out all edges that do not belong any perfect $(g, f)$-matchings, and therefore prune the corresponding

① $(\langle \mathbf{2}_1, \mathbf{1}_2, \mathbf{3}_3, \mathbf{4}_4, \mathbf{5}_5 \rangle)$
② $(\langle \mathbf{2}_1, \mathbf{1}_2, \mathbf{4}_3, \mathbf{3}_4, \mathbf{5}_5 \rangle)$
③ $(\langle \mathbf{3}_1, \mathbf{2}_2, \mathbf{1}_3, \mathbf{4}_4, \mathbf{5}_5 \rangle)$
④ $(\langle \mathbf{5}_1, \mathbf{2}_2, \mathbf{3}_3, \mathbf{4}_4, \mathbf{1}_5 \rangle)$
⑤ $(\langle \mathbf{5}_1, \mathbf{2}_2, \mathbf{4}_3, \mathbf{3}_4, \mathbf{1}_5 \rangle)$
⑥ $(\langle \mathbf{5}_1, \mathbf{3}_2, \mathbf{2}_3, \mathbf{4}_4, \mathbf{1}_5 \rangle)$

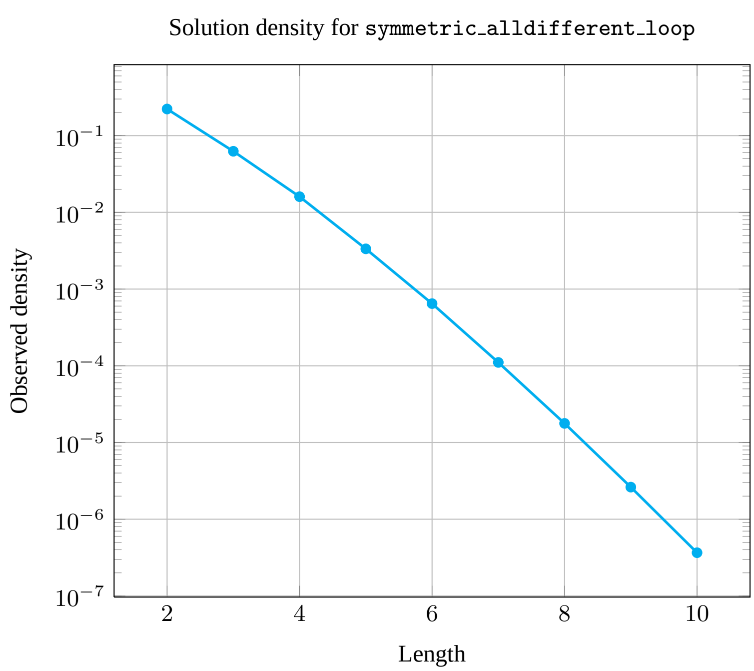
Figure 5.773: All solutions corresponding to the non ground example of the `symmetric_alldifferent_loop` constraint of the **All solutions** slot; the `index` attribute is displayed as indices of the `succ` attribute and self loops are coloured in red.

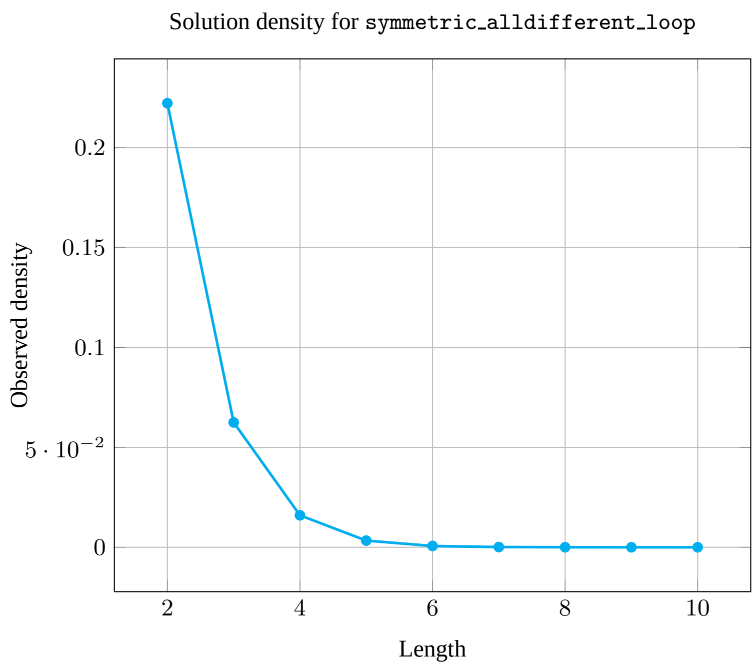
variables.

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Solutions | 2 | 4 | 10 | 26 | 76 | 232 | 764 | 2620 | 9496 |

Number of solutions for `symmetric_alldifferent_loop`: domains $0..n$






**See also**

**implied by:** `symmetric_alldifferent`.

**implies:** `twin`.

**implies (items to collection):** `lex_alldifferent`.

**Keywords**

**characteristic of a constraint:** all different, disequality.

**combinatorial object:** permutation, involution, matching.

**constraint type:** graph constraint, graph partitioning constraint.

**final graph structure:** circuit.

**modelling:** cycle.

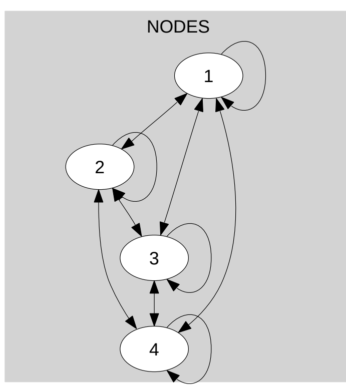
**Cond. implications**

`symmetric_alldifferent_loop`(`NODES`)
**implies** `permutation`(`VARIABLES` : `NODES`).

| | |
|---|---|
| **Arc input(s)** | NODES |
| **Arc generator** | $CLIQUE \mapsto$ collection(nodes1, nodes2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • nodes1.succ = nodes2.index<br>• nodes2.succ = nodes1.index |
| **Graph property(ies)** | $\mathbf{NARC}=$ \|NODES\| |

**Graph model** In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices.

Parts (A) and (B) of Figure 5.774 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.



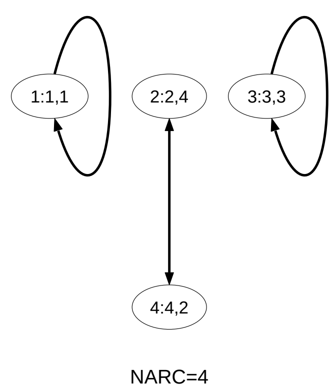




Figure 5.774: Initial and final graph of the symmetric_alldifferent_loop constraint

**Signature** Since all the index attributes of the NODES collection are distinct, and because of the first condition nodes1.succ = nodes2.index of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to the maximum number of vertices |NODES| of the final graph. So we can rewrite $\mathbf{NARC}=$ |NODES| to $\mathbf{NARC}\geq$ |NODES| and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

## 5.399 symmetric_cardinality

DESCRIPTION LINKS GRAPH

**Origin** Derived from global_cardinality by W. Kocjan.

**Constraint** symmetric_cardinality(VARS, VALS)

**Arguments**

$$\begin{array}{lll} \mathtt{VARS} & : & \mathtt{collection}(\mathtt{idvar}-\mathtt{int}, \mathtt{var}-\mathtt{svar}, \mathtt{l}-\mathtt{int}, \mathtt{u}-\mathtt{int}) \\ \mathtt{VALS} & : & \mathtt{collection}(\mathtt{idval}-\mathtt{int}, \mathtt{val}-\mathtt{svar}, \mathtt{l}-\mathtt{int}, \mathtt{u}-\mathtt{int}) \end{array}$$

**Restrictions**

$\mathtt{required}(\mathtt{VARS}, [\mathtt{idvar}, \mathtt{var}, \mathtt{l}, \mathtt{u}])$
$|\mathtt{VARS}| \geq 1$
$\mathtt{VARS.idvar} \geq 1$
$\mathtt{VARS.idvar} \leq |\mathtt{VARS}|$
$\mathtt{distinct}(\mathtt{VARS}, \mathtt{idvar})$
$\mathtt{VARS.l} \geq 0$
$\mathtt{VARS.l} \leq \mathtt{VARS.u}$
$\mathtt{VARS.u} \leq |\mathtt{VALS}|$
$\mathtt{required}(\mathtt{VALS}, [\mathtt{idval}, \mathtt{val}, \mathtt{l}, \mathtt{u}])$
$|\mathtt{VALS}| \geq 1$
$\mathtt{VALS.idval} \geq 1$
$\mathtt{VALS.idval} \leq |\mathtt{VALS}|$
$\mathtt{distinct}(\mathtt{VALS}, \mathtt{idval})$
$\mathtt{VALS.l} \geq 0$
$\mathtt{VALS.l} \leq \mathtt{VALS.u}$
$\mathtt{VALS.u} \leq |\mathtt{VARS}|$

**Purpose** Put in relation two sets: for each element of one set gives the corresponding elements of the other set to which it is associated. In addition, it constraints the number of elements associated with each element to be in a given interval.

**Example**

$$\left(\begin{array}{l} \left\langle\begin{array}{llll} \mathtt{idvar}-1 & \mathtt{var}-\{3\} & \mathtt{l}-0 & \mathtt{u}-1, \\ \mathtt{idvar}-2 & \mathtt{var}-\{1\} & \mathtt{l}-1 & \mathtt{u}-2, \\ \mathtt{idvar}-3 & \mathtt{var}-\{1,2\} & \mathtt{l}-1 & \mathtt{u}-2, \\ \mathtt{idvar}-4 & \mathtt{var}-\{1,3\} & \mathtt{l}-2 & \mathtt{u}-3 \end{array}\right\rangle, \\ \left\langle\begin{array}{llll} \mathtt{idval}-1 & \mathtt{val}-\{2,3,4\} & \mathtt{l}-3 & \mathtt{u}-4, \\ \mathtt{idval}-2 & \mathtt{val}-\{3\} & \mathtt{l}-1 & \mathtt{u}-1, \\ \mathtt{idval}-3 & \mathtt{val}-\{1,4\} & \mathtt{l}-1 & \mathtt{u}-2, \\ \mathtt{idval}-4 & \mathtt{val}-\emptyset & \mathtt{l}-0 & \mathtt{u}-1 \end{array}\right\rangle \end{array}\right)$$

The symmetric_cardinality constraint holds since:

- $3 \in \mathtt{VARS}[1].\mathtt{var} \Leftrightarrow 1 \in \mathtt{VALS}[3].\mathtt{val}$,
- $1 \in \mathtt{VARS}[2].\mathtt{var} \Leftrightarrow 2 \in \mathtt{VALS}[1].\mathtt{val}$,
- $1 \in \mathtt{VARS}[3].\mathtt{var} \Leftrightarrow 3 \in \mathtt{VALS}[1].\mathtt{val}$,
- $2 \in \mathtt{VARS}[3].\mathtt{var} \Leftrightarrow 3 \in \mathtt{VALS}[2].\mathtt{val}$,

- $1 \in$ VARS$[4]$.var $\Leftrightarrow 4 \in$ VALS$[1]$.val,
- $3 \in$ VARS$[4]$.var $\Leftrightarrow 4 \in$ VALS$[3]$.val,
- The number of elements of VARS$[1]$.var $= \{3\}$ belongs to interval $[0, 1]$,
- The number of elements of VARS$[2]$.var $= \{1\}$ belongs to interval $[1, 2]$,
- The number of elements of VARS$[3]$.var $= \{1, 2\}$ belongs to interval $[1, 2]$,
- The number of elements of VARS$[4]$.var $= \{1, 3\}$ belongs to interval $[2, 3]$,
- The number of elements of VALS$[1]$.val $= \{2, 3, 4\}$ belongs to interval $[3, 4]$,
- The number of elements of VALS$[2]$.val $= \{3\}$ belongs to interval $[1, 1]$,
- The number of elements of VALS$[3]$.val $= \{1, 4\}$ belongs to interval $[1, 2]$,
- The number of elements of VALS$[4]$.val $= \emptyset$ belongs to interval $[0, 1]$.

**Typical**

$|\text{VARS}| > 1$
$|\text{VALS}| > 1$

**Symmetries**

- Items of VARS are permutable.
- Items of VALS are permutable.

**Usage**

The most simple example of applying symmetric_gcc is a variant of personnel assignment problem, where one person can be assigned to perform between $n$ and $m$ $(n \leq m)$ jobs, and every job requires between $p$ and $q$ $(p \leq q)$ persons. In addition every job requires different kind of skills. The previous problem can be modelled as follows:

- For each person we create an item of the VARS collection,
- For each job we create an item of the VALS collection,
- There is an arc between a person and the particular job if this person is qualified to perform it.

**Remark**

The symmetric_gcc constraint generalises the global_cardinality constraint by allowing a variable to take more than one value.

**Algorithm**

A first flow-based arc-consistency algorithm for the symmetric_cardinality constraint is described in [241]. A second arc-consistency filtering algorithm exploiting matching theory [148] is described in [129, 130].

**See also**

**common keyword:** link_set_to_booleans *(constraint involving set variables)*.

**generalisation:** symmetric_gcc *(*fixed interval *replaced by* variable*)*.

**root concept:** global_cardinality.

**used in graph description:** in_set.

**Keywords**

**application area:** assignment.

**combinatorial object:** relation.

**constraint arguments:** constraint involving set variables.

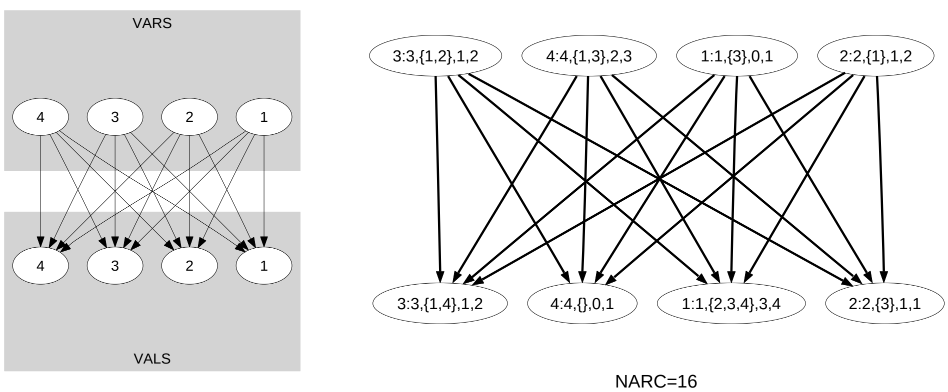
**constraint type:** decomposition, timetabling constraint.

**filtering:** flow, bipartite matching.

| | |
|---|---|
| **Arc input(s)** | `VARS VALS` |
| **Arc generator** | $PRODUCT \mapsto$ `collection(vars, vals)` |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • `in_set(vars.idvar, vals.val)` $\Leftrightarrow$ `in_set(vals.idval, vars.var)`<br>• `vars.l` $\leq$ `card_set(vars.var)`<br>• `vars.u` $\geq$ `card_set(vars.var)`<br>• `vals.l` $\leq$ `card_set(vals.val)`<br>• `vals.u` $\geq$ `card_set(vals.val)` |
| **Graph property(ies)** | $\mathbf{NARC}$ = \|`VARS`\| $*$ \|`VALS`\| |

**Graph model** The graph model used for the `symmetric_cardinality` is similar to the one used in the `domain_constraint` or in the `link_set_to_booleans` constraints: we use an equivalence in the arc constraint and ask all arc constraints to hold.

Parts (A) and (B) of Figure 5.775 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, all the arcs of the final graph are stressed in bold.




**(A)** **(B)**

Figure 5.775: Initial and final graph of the `symmetric_cardinality` constraint

**Signature** Since we use the $PRODUCT$ arc generator on the collections `VARS` and `VALS`, the number of arcs of the initial graph is equal to |`VARS`| $\cdot$ |`VALS`|. Therefore the maximum number of arcs of the final graph is also equal to |`VARS`| $\cdot$ |`VALS`| and we can rewrite $\mathbf{NARC}$ = |`VARS`| $\cdot$ |`VALS`| to $\mathbf{NARC} \geq$ |`VARS`| $\cdot$ |`VALS`|. So we can simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

# 5.400 symmetric_gcc

DESCRIPTION LINKS GRAPH

**Origin** Derived from global_cardinality by W. Kocjan.

**Constraint** symmetric_gcc(VARS, VALS)

**Synonym** sgcc.

**Arguments**

VARS : collection(idvar−int, var−svar, nocc−dvar)
VALS : collection(idval−int, val−svar, nocc−dvar)

**Restrictions**

required(VARS, [idvar, var, nocc])
$|$VARS$| \geq 1$
VARS.idvar $\geq 1$
VARS.idvar $\leq |$VARS$|$
distinct(VARS, idvar)
VARS.nocc $\geq 0$
VARS.nocc $\leq |$VALS$|$
required(VALS, [idval, val, nocc])
$|$VALS$| \geq 1$
VALS.idval $\geq 1$
VALS.idval $\leq |$VALS$|$
distinct(VALS, idval)
VALS.nocc $\geq 0$
VALS.nocc $\leq |$VARS$|$

**Purpose** Put in relation two sets: for each element of one set gives the corresponding elements of the other set to which it is associated. In addition, enforce a cardinality constraint on the number of occurrences of each value.

**Example**

$$\left(\begin{array}{l}\left\langle\begin{array}{lll}\texttt{idvar}-1 & \texttt{var}-\{3\} & \texttt{nocc}-1,\\ \texttt{idvar}-2 & \texttt{var}-\{1\} & \texttt{nocc}-1,\\ \texttt{idvar}-3 & \texttt{var}-\{1,2\} & \texttt{nocc}-2,\\ \texttt{idvar}-4 & \texttt{var}-\{1,3\} & \texttt{nocc}-2\end{array}\right\rangle,\\ \left\langle\begin{array}{lll}\texttt{idval}-1 & \texttt{val}-\{2,3,4\} & \texttt{nocc}-3,\\ \texttt{idval}-2 & \texttt{val}-\{3\} & \texttt{nocc}-1,\\ \texttt{idval}-3 & \texttt{val}-\{1,4\} & \texttt{nocc}-2,\\ \texttt{idval}-4 & \texttt{val}-\emptyset & \texttt{nocc}-0\end{array}\right\rangle\end{array}\right)$$

The symmetric_gcc constraint holds since:

- $3 \in$ VARS[1].var $\Leftrightarrow 1 \in$ VALS[3].val,
- $1 \in$ VARS[2].var $\Leftrightarrow 2 \in$ VALS[1].val,
- $1 \in$ VARS[3].var $\Leftrightarrow 3 \in$ VALS[1].val,
- $2 \in$ VARS[3].var $\Leftrightarrow 3 \in$ VALS[2].val,

- $1 \in$ VARS$[4]$.var $\Leftrightarrow 4 \in$ VALS$[1]$.val,
- $3 \in$ VARS$[4]$.var $\Leftrightarrow 4 \in$ VALS$[3]$.val,
- The number of elements of VARS$[1]$.var $= \{3\}$ is equal to $1$,
- The number of elements of VARS$[2]$.var $= \{1\}$ is equal to $1$,
- The number of elements of VARS$[3]$.var $= \{1, 2\}$ is equal to $2$,
- The number of elements of VARS$[4]$.var $= \{1, 3\}$ is equal to $2$,
- The number of elements of VALS$[1]$.val $= \{2, 3, 4\}$ is equal to $3$,
- The number of elements of VALS$[2]$.val $= \{3\}$ is equal to $1$,
- The number of elements of VALS$[3]$.val $= \{1, 4\}$ is equal to $2$,
- The number of elements of VALS$[4]$.val $= \emptyset$ is equal to $0$.

**Typical**

$|$VARS$| > 1$
$|$VALS$| > 1$

**Symmetries**

- Items of VARS are permutable.
- Items of VALS are permutable.

**Usage**

The most simple example of applying symmetric_gcc is a variant of personnel assignment problem, where one person can be assigned to perform between $n$ and $m$ $(n \leq m)$ jobs, and every job requires between $p$ and $q$ $(p \leq q)$ persons. In addition every job requires different kind of skills. The previous problem can be modelled as follows:

- For each person we create an item of the VARS collection,
- For each job we create an item of the VALS collection,
- There is an arc between a person and the particular job if this person is qualified to perform it.

**Remark**

The symmetric_gcc constraint generalises the global_cardinality constraint by allowing a variable to take more than one value. It corresponds to a variant of the symmetric_cardinality constraint described in [241] where the occurrence variables of the VARS and VALS collections are replaced by fixed intervals.

**See also**

**common keyword:** link_set_to_booleans *(constraint involving set variables)*.

**root concept:** global_cardinality.

**specialisation:** symmetric_cardinality *(*variable *replaced by* fixed interval*)*.

**used in graph description:** in_set.

**Keywords**

**application area:** assignment.

**combinatorial object:** relation.

**constraint arguments:** constraint involving set variables.

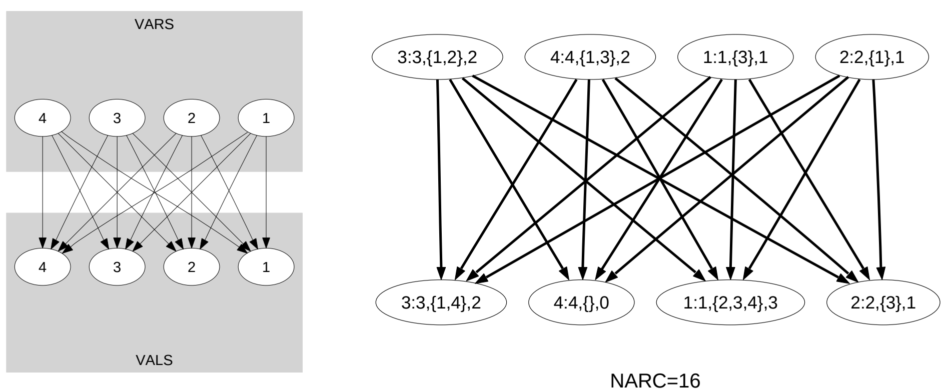
**constraint type:** decomposition, timetabling constraint.

**filtering:** flow.

| | |
|---|---|
| **Arc input(s)** | VARS VALS |
| **Arc generator** | $PRODUCT \mapsto$ collection(vars, vals) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • in_set(vars.idvar, vals.val) $\Leftrightarrow$ in_set(vals.idval, vars.var)<br>• vars.nocc = card_set(vars.var)<br>• vals.nocc = card_set(vals.val) |
| **Graph property(ies)** | $\mathbf{NARC} = \lvert\mathtt{VARS}\rvert * \lvert\mathtt{VALS}\rvert$ |

**Graph model** The graph model used for the symmetric_gcc is similar to the one used in the domain_constraint or in the link_set_to_booleans constraints: we use an equivalence in the arc constraint and ask all arc constraints to hold.

Parts (A) and (B) of Figure 5.776 respectively show the initial and final graph. Since we use the **NARC** graph property, all the arcs of the final graph are stressed in bold.




**(A)** **(B)**

Figure 5.776: Initial and final graph of the symmetric_gcc constraint

**Signature** Since we use the *PRODUCT* arc generator on the collections VARS and VALS, the number of arcs of the initial graph is equal to $|\mathtt{VARS}| \cdot |\mathtt{VALS}|$. Therefore the maximum number of arcs of the final graph is also equal to $|\mathtt{VARS}| \cdot |\mathtt{VALS}|$ and we can rewrite $\mathbf{NARC} = |\mathtt{VARS}| \cdot |\mathtt{VALS}|$ to $\mathbf{NARC} \geq |\mathtt{VARS}| \cdot |\mathtt{VALS}|$. So we can simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

# 5.401 temporal_path

DESCRIPTION LINKS GRAPH

**Origin** ILOG

**Constraint** temporal_path(NPATH, NODES)

**Arguments**

$$\begin{array}{lll} \texttt{NPATH} & : & \texttt{dvar} \\ \texttt{NODES} & : & \texttt{collection}\left(\begin{array}{l}\texttt{index}-\texttt{int},\\ \texttt{succ}-\texttt{dvar},\\ \texttt{start}-\texttt{dvar},\\ \texttt{end}-\texttt{dvar}\end{array}\right)\end{array}$$

**Restrictions**

$\texttt{NPATH} \geq 1$
$\texttt{NPATH} \leq |\texttt{NODES}|$
$\texttt{required}(\texttt{NODES}, [\texttt{index}, \texttt{succ}, \texttt{start}, \texttt{end}])$
$|\texttt{NODES}| > 0$
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
$\texttt{distinct}(\texttt{NODES}, \texttt{index})$
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$
$\texttt{NODES.start} \leq \texttt{NODES.end}$

**Purpose**

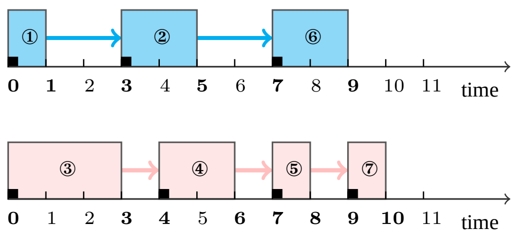
Let $G$ be the digraph described by the NODES collection. Partition $G$ with a set of disjoint paths such that each vertex of the graph belongs to a single path. In addition, for all pairs of consecutive vertices of a path we have a precedence constraint that enforces the end associated with the first vertex to be less than or equal to the start related to the second vertex.

**Example**

$$\left(2, \left\langle\begin{array}{llll} \texttt{index}-1 & \texttt{succ}-2 & \texttt{start}-0 & \texttt{end}-1,\\ \texttt{index}-2 & \texttt{succ}-6 & \texttt{start}-3 & \texttt{end}-5,\\ \texttt{index}-3 & \texttt{succ}-4 & \texttt{start}-0 & \texttt{end}-3,\\ \texttt{index}-4 & \texttt{succ}-5 & \texttt{start}-4 & \texttt{end}-6,\\ \texttt{index}-5 & \texttt{succ}-7 & \texttt{start}-7 & \texttt{end}-8,\\ \texttt{index}-6 & \texttt{succ}-6 & \texttt{start}-7 & \texttt{end}-9,\\ \texttt{index}-7 & \texttt{succ}-7 & \texttt{start}-9 & \texttt{end}-10 \end{array}\right\rangle\right)$$

The temporal_path constraint holds since:

- The items of the NODES collection represent the two ($\texttt{NPATH} = 2$) paths $1 \rightarrow 2 \rightarrow 6$ and $3 \rightarrow 4 \rightarrow 5 \rightarrow 7$.
- As illustrated by Figure 5.777, all precedences between adjacent vertices of a same path hold: each item $i$ ($1 \leq i \leq 7$) of the NODES collection is represented by a rectangle starting and ending at instants $\texttt{NODES}[i].\texttt{start}$ and $\texttt{NODES}[i].\texttt{end}$; the number within each rectangle designates the index of the corresponding item within the NODES collection.




| | | | | |
|---|---|---|---|---|
| ① | index − 1 | succ − 2 | start − 0 | end − 1 |
| ② | index − 2 | succ − 6 | start − 3 | end − 5 |
| ③ | index − 3 | succ − 4 | start − 0 | end − 3 |
| ④ | index − 4 | succ − 5 | start − 4 | end − 6 |
| ⑤ | index − 5 | succ − 7 | start − 7 | end − 8 |
| ⑥ | index − 6 | succ − 6 | start − 7 | end − 9 |
| ⑦ | index − 7 | succ − 7 | start − 9 | end − 10 |

NODES

Figure 5.777: The two paths of the **Example** slot represented as two sequences of tasks

**Typical**

NPATH $<$ |NODES|
|NODES| $> 1$
NODES.start $<$ NODES.end

**Symmetries**

- Items of NODES are permutable.
- One and the same constant can be added to the start and end attributes of all items of NODES.

**Arg. properties**

Functional dependency: NPATH determined by NODES.

**Remark**

This constraint is related to the path constraint of **Ilog Solver**. It can also be directly expressed with the cycle [41] constraint of **CHIP** by using the *diff nodes* and the origin parameters. A generic model based on linear programming that handles paths, trees and cycles is presented in [244].

**Reformulation**

The temporal_path(NPATH, NODES) constraint can be expressed in term of a conjunction of one path constraint, |NODES| element constraints, and |NODES| inequalities constraints:

- We pass to the path constraint the number of path variable NPATH as well as the items of the NODES collection form which we remove the start and end attributes.
- To the $i$-th ($1 \leq i \leq$ |NODES|) item of the NODES collection, we create a variable $Start_{succ_i}$ and an element(NODES[$i$].succ, $\langle T_{i,1}, T_{i,2}, \ldots, T_{i,\texttt{NODES}}\rangle$, $Start_{succ_i}$) constraint, where $T_{i,j}$ = NODES[$i$].start if $i \neq j$ and $T_{i,i}$ = NODES[$i$].end otherwise.
- Finaly to the $i$-th ($1 \leq i \leq$ |NODES|) item of the NODES collection, we also create an inequality constraint NODES[$i$].end $\leq Start_{succ_i}$. Note that, since $T_{i,i}$ was initialised to NODES[$i$].end, the inequality NODES[$i$].end $\leq T_{i,j}$ holds when $i = j$.

With respect to the **Example** slot we get the following conjunction of constraints:

$\texttt{path}(2, \langle \texttt{index}-1\ \texttt{succ}-2, \texttt{index}-2\ \texttt{succ}-6, \texttt{index}-3\ \texttt{succ}-4,$
$\texttt{index}-4\ \texttt{succ}-5, \texttt{index}-5\ \texttt{succ}-7, \texttt{index}-6\ \texttt{succ}-6,$
$\texttt{index}-7\ \texttt{succ}-7\rangle),$
$\texttt{element}(2, \langle 1, 3, 0, 4, 7, 7, 9\rangle, 3),$
$\texttt{element}(6, \langle 1, 5, 0, 4, 7, 7, 9\rangle, 7),$
$\texttt{element}(4, \langle 1, 5, 3, 4, 7, 7, 9\rangle, 4),$
$\texttt{element}(5, \langle 1, 5, 3, 6, 7, 7, 9\rangle, 7),$
$\texttt{element}(7, \langle 1, 5, 3, 6, 8, 7, 9\rangle, 9),$
$\texttt{element}(6, \langle 1, 5, 3, 6, 8, 9, 9\rangle, 9),$
$\texttt{element}(7, \langle 1, 5, 3, 6, 8, 9, 10\rangle, 10),$
$1 \leq 3, 5 \leq 7, 3 \leq 4, 6 \leq 7, 8 \leq 9, 9 \leq 9, 10 \leq 10.$

**See also**

**common keyword:** `path_from_to` *(path)*.

**implies (items to collection):** `atleast_nvector`.

**specialisation:** `path` *(time dimension removed)*.

**Keywords**

**combinatorial object:** path.

**constraint type:** graph constraint, graph partitioning constraint.

**final graph structure:** connected component.

**modelling:** sequence dependent set-up, functional dependency.

**modelling exercises:** sequence dependent set-up.

| | |
|---|---|
| **Arc input(s)** | NODES |
| **Arc generator** | $CLIQUE \mapsto$ collection(nodes1, nodes2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • nodes1.succ = nodes2.index<br>• nodes1.succ = nodes1.index ∨ nodes1.end ≤ nodes2.start<br>• nodes1.start ≤ nodes1.end<br>• nodes2.start ≤ nodes2.end |
| **Graph property(ies)** | • **MAX_ID** ≤ 1<br>• **NCC** = NPATH<br>• **NVERTEX** = \|NODES\| |

**Graph model** The arc constraint is a conjunction of four conditions that respectively correspond to:

- A constraint that links the successor variable of a first vertex to the index attribute of a second vertex,
- A precedence constraint that applies on one vertex and its distinct successor,
- One precedence constraint between the start and the end of the vertex that corresponds to the departure of an arc,
- One precedence constraint between the start and the end of the vertex that corresponds to the arrival of an arc.

We use the following three graph properties in order to enforce the partitioning of the graph in distinct paths:

- The first property **MAX_ID** ≤ 1 enforces that each vertex has no more than one predecessor (**MAX_ID** does not consider loops),
- The second property **NCC** = NPATH ensures that we have the required number of paths,
- The third property **NVERTEX** = |NODES| enforces that, for each vertex, the start is not located after the end.

Parts (A) and (B) of Figure 5.778 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_ID**, the **NCC** and the **NVERTEX** graph properties we display the following information on the final graph:

- We show with a double circle a vertex that has the maximum number of predecessors.
- We show the two connected components corresponding to the two paths.
- We put in bold the vertices.

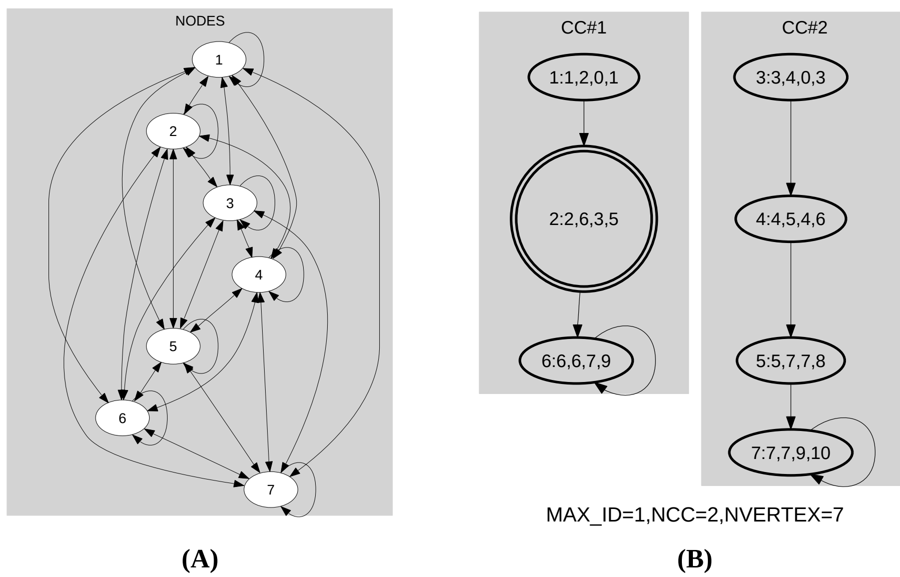



Figure 5.778: Initial and final graph of the `temporal_path` constraint

## 5.402 tour

**DESCRIPTION** **LINKS** **GRAPH**

**Origin** [5]

**Constraint** `tour(NODES)`

**Synonyms** `atour`, `cycle`.

**Argument** NODES : `collection(index-int, succ-svar)`

**Restrictions**

$|\texttt{NODES}| \geq 3$
`required(NODES, [index, succ])`
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
`distinct(NODES, index)`

**Purpose** Enforce to cover an undirected graph $G$ described by the `NODES` collection with a Hamiltonian cycle.

**Example**

$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\{2,4\},\\ \texttt{index}-2 & \texttt{succ}-\{1,3\},\\ \texttt{index}-3 & \texttt{succ}-\{2,4\},\\ \texttt{index}-4 & \texttt{succ}-\{1,3\}\end{array}\right\rangle\right)$$

The `tour` constraint holds since its `NODES` argument depicts the following Hamiltonian cycle visiting successively the vertices 1, 2, 3 and 4.

**Symmetry** Items of `NODES` are permutable.

**Algorithm** When the number of vertices is odd (i.e., $|\texttt{NODES}|$ is odd) a necessary condition is that the graph is not bipartite. Other necessary conditions for filtering the `tour` constraint are given in [131, 130].

**See also** **common keyword:** `circuit` *(graph partitioning constraint, Hamiltonian)*, `cycle` *(graph constraint)*, `link_set_to_booleans` *(constraint involving set variables)*.

**used in graph description:** `in_set`.

**Keywords** **characteristic of a constraint:** undirected graph.

**combinatorial object:** matching.

**constraint arguments:** constraint involving set variables.

**constraint type:** graph constraint.

**filtering:** DFS-bottleneck, linear programming.

**problems:** Hamiltonian.

| | |
|---|---|
| **Arc input(s)** | NODES |
| **Arc generator** | $CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | in_set(nodes2.index, nodes1.succ) $\Leftrightarrow$ in_set(nodes1.index, nodes2.succ) |
| **Graph property(ies)** | **NARC**$= \lvert$NODES$\rvert * \lvert$NODES$\rvert - \lvert$NODES$\rvert$ |

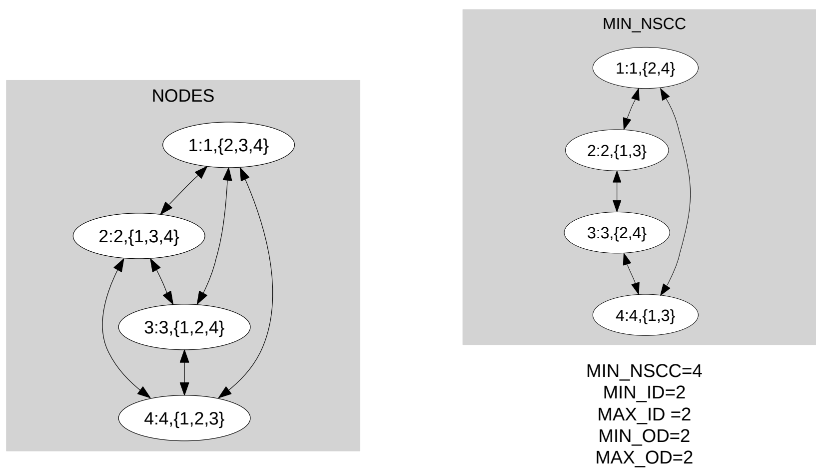
| | |
|---|---|
| **Arc input(s)** | NODES |
| **Arc generator** | $CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | in_set(nodes2.index, nodes1.succ) |
| **Graph property(ies)** | • **MIN_NSCC**$= \lvert$NODES$\rvert$<br>• **MIN_ID**$= 2$<br>• **MAX_ID**$= 2$<br>• **MIN_OD**$= 2$<br>• **MAX_OD**$= 2$ |

**Graph model** The first graph property enforces the subsequent condition: If we have an arc from the $i^{th}$ vertex to the $j^{th}$ vertex then we have also an arc from the $j^{th}$ vertex to the $i^{th}$ vertex. The second graph property enforces the following constraints:

- We have one strongly connected component containing |NODES| vertices,
- Each vertex has exactly two predecessors and two successors.

Part (A) of Figure 5.779 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.779 gives the final graph associated with the **Example** slot. The tour constraint holds since the final graph corresponds to a Hamiltonian cycle.

**Signature** Since the maximum number of vertices of the final graph is equal to |NODES|, we can rewrite the graph property **MIN_NSCC** $= \lvert$NODES$\rvert$ to **MIN_NSCC** $\geq \lvert$NODES$\rvert$ and simplify $\underline{\textbf{MIN\_NSCC}}$ to $\overline{\textbf{MIN\_NSCC}}$.




**(A)** **(B)**

Figure 5.779: Initial and final graph of the `tour` set constraint

## 5.403 track

DESCRIPTION LINKS GRAPH

**Origin** [274]

**Constraint** track(NTRAIL, TASKS)

**Arguments**

| | | |
|---|---|---|
| NTRAIL | : | int |
| TASKS | : | collection(trail−int, origin−dvar, end−dvar) |

**Restrictions**

NTRAIL $>$ 0
NTRAIL $\leq$ |TASKS|
|TASKS| $>$ 0
required(TASKS, [trail, origin, end])
TASKS.origin $\leq$ TASKS.end

**Purpose**

The track constraint forces that, at each point in time overlapped by at least one task, the number of distinct values of the trail attribute of the set of tasks that overlap that point, is equal to NTRAIL.

**Example**

$$\left(2, \left\langle \begin{array}{lll} \texttt{trail}-1 & \texttt{origin}-1 & \texttt{end}-2, \\ \texttt{trail}-2 & \texttt{origin}-1 & \texttt{end}-2, \\ \texttt{trail}-1 & \texttt{origin}-2 & \texttt{end}-4, \\ \texttt{trail}-2 & \texttt{origin}-2 & \texttt{end}-3, \\ \texttt{trail}-2 & \texttt{origin}-3 & \texttt{end}-4 \end{array} \right\rangle \right)$$

Figure 5.780 represents the tasks of the example: to the $i^{th}$ task of the TASKS collection corresponds a rectangle labelled by $i$. The track constraint holds since:

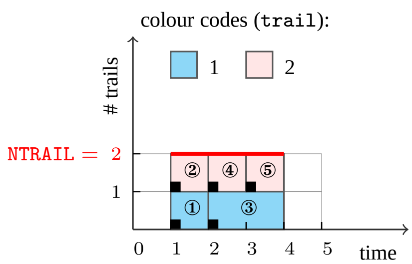
- The first and second tasks both overlap instant 1 and have a respective trail of 1 and 2. This makes two distinct values for the trail attribute at instant 1.
- The third and fourth tasks both overlap instant 2 and have a respective trail of 1 and 2. This makes two distinct values for the trail attribute at instant 2.
- The third and fifth tasks both overlap instant 3 and have a respective trail of 1 and 2. This makes two distinct values for the trail attribute at instant 3.

**Typical**

NTRAIL $<$ |TASKS|
|TASKS| $>$ 1
range(TASKS.trail) $>$ 1
TASKS.origin $<$ TASKS.end

**Symmetries**

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




TASKS

| | | | |
|---|---|---|---|
| ① | `trail` $-1$ | `origin` $-1$ | `end` $-2$ |
| ② | `trail` $-2$ | `origin` $-1$ | `end` $-2$ |
| ③ | `trail` $-1$ | `origin` $-2$ | `end` $-4$ |
| ④ | `trail` $-2$ | `origin` $-2$ | `end` $-3$ |
| ⑤ | `trail` $-2$ | `origin` $-3$ | `end` $-4$ |

Figure 5.780: The tasks associated with the example of the **Example** slot, at each instant we have two distinct values for the `trail` attribute ($\texttt{NTRAIL} = 2$)

**Reformulation** The `track` constraint can be expressed in term of a set of reified constraints and of $2 \cdot |\texttt{TASKS}|$ `nvalue` constraints:

1. For each pair of tasks $\texttt{TASKS}[i], \texttt{TASKS}[j]$ $(i, j \in [1, |\texttt{TASKS}|])$ of the TASKS collection we create a variable $T_{ij}^{\texttt{origin}}$ which is set to the `trail` attribute of task $\texttt{TASKS}[j]$ if task $\texttt{TASKS}[j]$ overlaps the origin attribute of task $\texttt{TASKS}[i]$, and to the `trail` attribute of task $\texttt{TASKS}[i]$ otherwise:
   - If $i = j$:
     - $T_{ij}^{\texttt{origin}} = \texttt{TASKS}[i].\texttt{trail}$.
   - If $i \neq j$:
     - $T_{ij}^{\texttt{origin}} = \texttt{TASKS}[i].\texttt{trail} \lor T_{ij}^{\texttt{origin}} = \texttt{TASKS}[j].\texttt{trail}$.
     - $((\texttt{TASKS}[j].\texttt{origin} \leq \texttt{TASKS}[i].\texttt{origin} \land$
       $\texttt{TASKS}[j].\texttt{end} > \texttt{TASKS}[i].\texttt{origin}) \land (T_{ij}^{\texttt{origin}} = \texttt{TASKS}[j].\texttt{trail})) \lor$
       $((\texttt{TASKS}[j].\texttt{origin} > \texttt{TASKS}[i].\texttt{origin} \lor$
       $\texttt{TASKS}[j].\texttt{end} \leq \texttt{TASKS}[i].\texttt{origin}) \land (T_{ij}^{\texttt{origin}} = \texttt{TASKS}[i].\texttt{trail}))$
2. For each task $\texttt{TASKS}[i]$ $(i \in [1, |\texttt{TASKS}|])$ we impose the number of distinct trails associated with the tasks that overlap the origin of task $\texttt{TASKS}[i]$ ($\texttt{TASKS}[i]$ overlaps its own origin) to be equal to NTRAIL:
   $\texttt{nvalue}(\texttt{NTRAIL}, \langle T_{i1}^{\texttt{origin}}, T_{i2}^{\texttt{origin}}, \ldots, T_{i|\texttt{TASKS}|}^{\texttt{origin}} \rangle)$.
3. For each pair of tasks $\texttt{TASKS}[i], \texttt{TASKS}[j]$ $(i, j \in [1, |\texttt{TASKS}|])$ of the TASKS collection we create a variable $T_{ij}^{\texttt{end}}$ which is set to the `trail` attribute of task $\texttt{TASKS}[j]$ if task $\texttt{TASKS}[j]$ overlaps the end attribute of task $\texttt{TASKS}[i]$, and to the `trail` attribute of task $\texttt{TASKS}[i]$ otherwise:
   - If $i = j$:
     - $T_{ij}^{\texttt{end}} = \texttt{TASKS}[i].\texttt{trail}$.
   - If $i \neq j$:
     - $T_{ij}^{\texttt{end}} = \texttt{TASKS}[i].\texttt{trail} \lor T_{ij}^{\texttt{end}} = \texttt{TASKS}[j].\texttt{trail}$.
     - $((\texttt{TASKS}[j].\texttt{origin} \leq \texttt{TASKS}[i].\texttt{end} - 1 \land$
       $\texttt{TASKS}[j].\texttt{end} > \texttt{TASKS}[i].\texttt{end} - 1) \land (T_{ij}^{\texttt{end}} = \texttt{TASKS}[j].\texttt{trail})) \lor$
       $((\texttt{TASKS}[j].\texttt{origin} > \texttt{TASKS}[i].\texttt{end} - 1 \lor$
       $\texttt{TASKS}[j].\texttt{end} \leq \texttt{TASKS}[i].\texttt{end} - 1) \land (T_{ij}^{\texttt{end}} = \texttt{TASKS}[i].\texttt{trail}))$
4. For each task $\texttt{TASKS}[i]$ $(i \in [1, |\texttt{TASKS}|])$ we impose the number of distinct trails associated with the tasks that overlap the end of task $\texttt{TASKS}[i]$ ($\texttt{TASKS}[i]$ overlaps its

own end) to be equal to NTRAIL:
nvalue(NTRAIL, $\langle T_{i1}^{\mathtt{end}}, T_{i2}^{\mathtt{end}}, \ldots, T_{i|\mathtt{TASKS}|}^{\mathtt{end}}\rangle$).

With respect to the **Example** slot we get the following conjunction of nvalue constraints:

- The nvalue($2, \langle 1, 2, 1, 1, 1\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the origin of the first task (i.e., instant $1$) that has a trail of $1$.
- The nvalue($2, \langle 1, 2, 2, 2, 2\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the origin of the second task (i.e., instant $1$) that has a trail of $2$.
- The nvalue($2, \langle 1, 1, 1, 2, 1\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the origin of the third task (i.e., instant $2$) that has a trail of $1$.
- The nvalue($2, \langle 2, 2, 1, 2, 2\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the origin of the fourth task (i.e., instant $2$) that has a trail of $2$.
- The nvalue($2, \langle 2, 2, 1, 2, 2\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the origin of the fifth task (i.e., instant $3$) that has a trail of $2$.
- The nvalue($2, \langle 1, 2, 1, 1, 1\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the last instant of the first task (i.e., instant $1$) that has a trail of $1$.
- The nvalue($2, \langle 1, 2, 2, 2, 2\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the last instant of the second task (i.e., instant $1$) that has a trail of $2$.
- The nvalue($2, \langle 1, 1, 1, 1, 2\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the last instant of the third task (i.e., instant $3$) that has a trail of $1$.
- The nvalue($2, \langle 2, 2, 1, 2, 2\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the last instant of the fourth task (i.e., instant $2$) that has a trail of $2$.
- The nvalue($2, \langle 2, 2, 1, 2, 2\rangle$) constraint corresponding to the trail attributes of the tasks that overlap the last instant of the fifth task (i.e., instant $3$) that has a trail of $2$.

**See also**

**common keyword:** coloured_cumulative *(resource constraint)*.

**implies (items to collection):** atleast_nvector.

**used in graph description:** nvalue.

**Keywords**

**characteristic of a constraint:** derived collection.

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

**Derived Collection**

$$\texttt{col}\left(\begin{array}{l}\texttt{TIME\_POINTS}-\texttt{collection}\left(\begin{array}{l}\texttt{origin}-\texttt{dvar},\\ \texttt{end}-\texttt{dvar},\\ \texttt{point}-\texttt{dvar}\end{array}\right),\\ \left[\begin{array}{l}\texttt{item}\left(\begin{array}{l}\texttt{origin}-\texttt{TASKS.origin},\\ \texttt{end}-\texttt{TASKS.end},\\ \texttt{point}-\texttt{TASKS.origin}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{origin}-\texttt{TASKS.origin},\\ \texttt{end}-\texttt{TASKS.end},\\ \texttt{point}-\texttt{TASKS.end}-1\end{array}\right)\end{array}\right]\end{array}\right)$$

**Arc input(s)** TASKS

**Arc generator** $SELF \mapsto \texttt{collection}(\texttt{tasks})$

**Arc arity** 1

**Arc constraint(s)** $\texttt{tasks.origin} \leq \texttt{tasks.end}$

**Graph property(ies)** $\mathbf{NARC} = |\texttt{TASKS}|$

**Arc input(s)** TIME_POINTS TASKS

**Arc generator** $PRODUCT \mapsto \texttt{collection}(\texttt{time\_points}, \texttt{tasks})$

**Arc arity** 2

**Arc constraint(s)**
- $\texttt{time\_points.end} > \texttt{time\_points.origin}$
- $\texttt{tasks.origin} \leq \texttt{time\_points.point}$
- $\texttt{time\_points.point} < \texttt{tasks.end}$

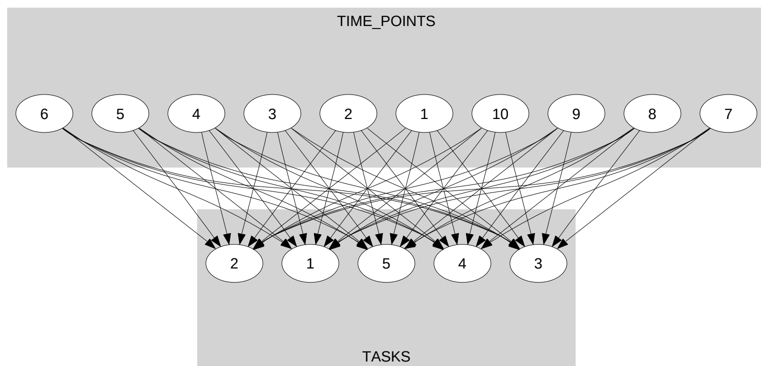
**Sets**

$$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables}-\texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ \left[\texttt{item}(\texttt{var}-\texttt{TASKS.trail})\right]\end{array}\right)\end{array}\right]$$

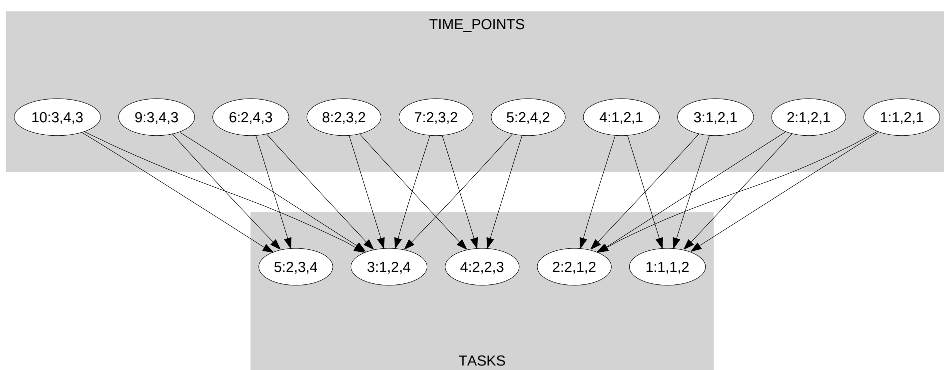
**Constraint(s) on sets** $\texttt{nvalue}(\texttt{NTRAIL}, \texttt{variables})$

**Graph model** Parts (A) and (B) of Figure 5.781 respectively show the initial and final graph of the second graph constraint of the **Example** slot.

**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 $|\texttt{TASKS}|$. Therefore we can rewrite $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$ and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.




**(A)**




**(B)**

Figure 5.781: Initial and final graph of the `track` constraint

## 5.404 tree

DESCRIPTION LINKS GRAPH

**Origin** N. Beldiceanu

**Constraint** tree(NTREES, NODES)

**Arguments**

NTREES : dvar
NODES : collection(index−int, succ−dvar)

**Restrictions**

$\texttt{NTREES} \geq 1$
$\texttt{NTREES} \leq |\texttt{NODES}|$
required(NODES, [index, succ])
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
distinct(NODES, index)
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$

**Purpose**

Given a digraph $G$ described by the NODES collection, cover $G$ by a set of NTREES trees in such a way that each vertex of $G$ belongs to one distinct tree. The edges of the trees are directed from their leaves to their respective roots.

**Example**

$$\left(\begin{array}{l}2,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{succ}-5,\\ \texttt{index}-3 & \texttt{succ}-5,\\ \texttt{index}-4 & \texttt{succ}-7,\\ \texttt{index}-5 & \texttt{succ}-1,\\ \texttt{index}-6 & \texttt{succ}-1,\\ \texttt{index}-7 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{succ}-5\end{array}\right\rangle\end{array}\right)$$

$$\left(\begin{array}{l}8,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{succ}-2,\\ \texttt{index}-3 & \texttt{succ}-3,\\ \texttt{index}-4 & \texttt{succ}-4,\\ \texttt{index}-5 & \texttt{succ}-5,\\ \texttt{index}-6 & \texttt{succ}-6,\\ \texttt{index}-7 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{succ}-8\end{array}\right\rangle\end{array}\right)$$

$$\left(\begin{array}{l}7,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-6,\\ \texttt{index}-2 & \texttt{succ}-2,\\ \texttt{index}-3 & \texttt{succ}-3,\\ \texttt{index}-4 & \texttt{succ}-4,\\ \texttt{index}-5 & \texttt{succ}-5,\\ \texttt{index}-6 & \texttt{succ}-6,\\ \texttt{index}-7 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{succ}-8\end{array}\right\rangle\end{array}\right)$$

The first `tree` constraint holds since the graph associated with the items of the `NODES` collection corresponds to two trees (i.e., `NTREES` $= 2$): each tree respectively involves the vertices $\{1, 2, 3, 5, 6, 8\}$ and $\{4, 7\}$. They are depicted by Figure 5.782.

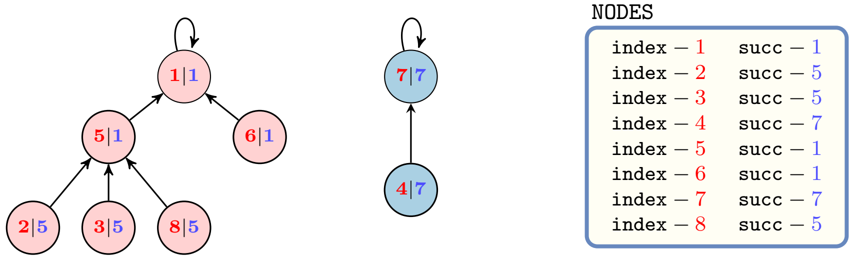



Figure 5.782: The two trees corresponding to the first example of the **Example** slot; each vertex contains the information `index`|`succ` where `succ` is the index of its father in the tree (by convention the father of the root is the root itself).

**All solutions**

Figure 5.783 gives all solutions to the following non ground instance of the `tree` constraint: `NTREES` $\in [3, 4]$, $\mathtt{S}_1 \in [1, 2]$, $\mathtt{S}_2 \in [1, 3]$, $\mathtt{S}_3 \in [1, 4]$, $\mathtt{S}_4 \in [2, 4]$, `tree`(`NTREES`, $\langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4\rangle$).

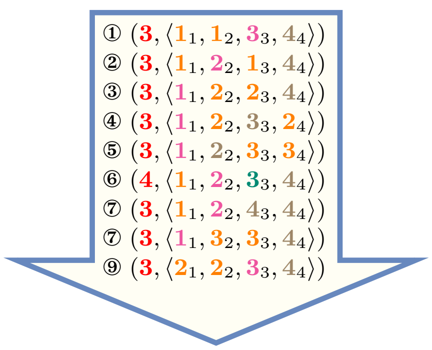



Figure 5.783: All solutions corresponding to the non ground example of the `tree` constraint of the **All solutions** slot (the `index` attribute is displayed as indices of the `succ` attribute)

**Typical**

`NTREES` $<$ |`NODES`|
|`NODES`| $> 2$

**Symmetry**

Items of `NODES` are permutable.

**Arg. properties**

Functional dependency: `NTREES` determined by `NODES`.

**Remark**

Given a complete digraph of $n$ vertices as well as an unrestricted number of trees `NTREES`, the total number of solutions to the corresponding `tree` constraint corresponds to the sequence A000272 of the On-Line Encyclopaedia of Integer Sequences [392].

Extension of the `tree` constraint to the *minimum spanning tree* constraint is described in [143, 349, 352].

**Algorithm**

An arc-consistency filtering algorithm for the `tree` constraint is described in [42]. This algorithm is based on a necessary and sufficient condition that we now depict.

To any `tree` constraint we associate the digraph $G = (V, E)$, where:

- To each item $\texttt{NODES}[i]$ of the `NODES` collection corresponds a vertex $v_i$ of $G$.
- For every pair of items $(\texttt{NODES}[i], \texttt{NODES}[j])$ of the `NODES` collection, where $i$ and $j$ are not necessarily distinct, there is an arc from $v_i$ to $v_j$ in $E$ if and only if $j$ is a potential value of $\texttt{NODES}[i].\texttt{succ}$.

A strongly connected component $C$ of $G$ is called a *sink component* if all the successors of all vertices of $C$ belong to $C$. Let `MINTREES` and `MAXTREES` respectively denote the number of sink components of $G$ and the number of vertices of $G$ with a loop.

The `tree` constraint has a solution if and only if:

- Each sink component of $G$ contains at least one vertex with a loop,
- The domain of `NTREES` has at least one value within interval $[\texttt{MINTREES}, \texttt{MAXTREES}]$.

Inspired by the idea of using dominators used in [223] for getting a linear time algorithm for computing strong articulation points of a digraph $G$, the worst case complexity of the algorithm proposed in [42] was also enhanced in a similar way by J.-G. Fages and X. Lorca [157].

**Reformulation**

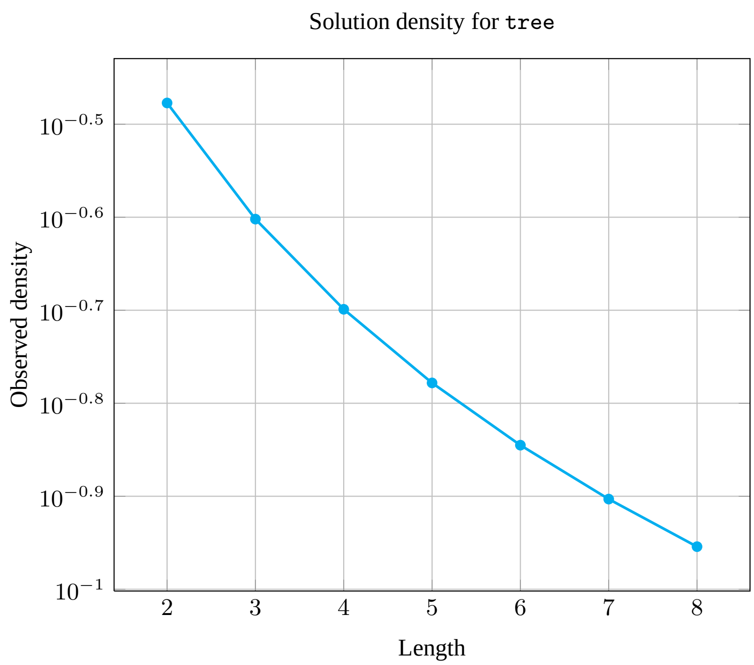
The `tree` constraint can be expressed in term of (1) a set of $|\texttt{NODES}|^2$ reified constraints for avoiding circuit between more than one node and of (2) $|\texttt{NODES}|$ reified constraints and of one sum constraint for counting the trees:

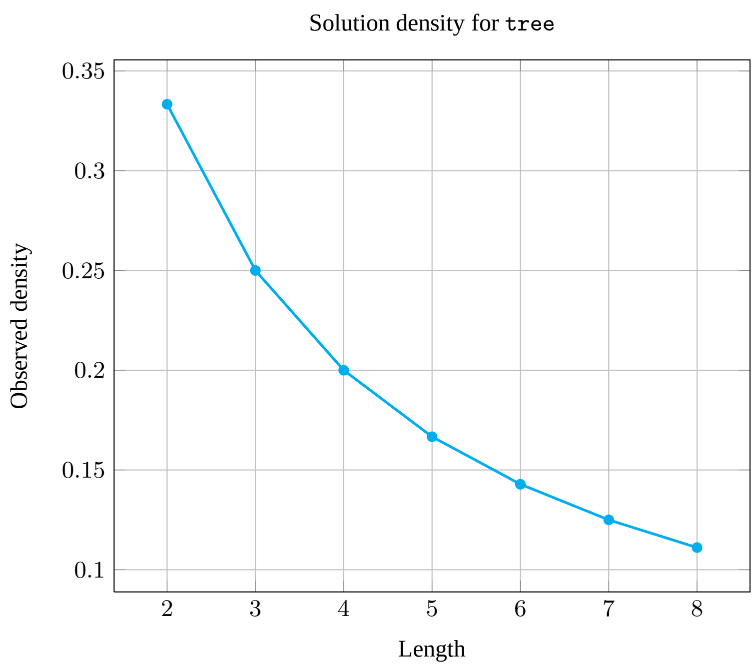
1. For each vertex $\texttt{NODES}[i]$ $(i \in [1, |\texttt{NODES}|])$ of the `NODES` collection we create a variable $R_i$ that takes its value within interval $[1, |\texttt{NODES}|]$. This variable represents the *rank* of vertex $\texttt{NODES}[i]$ within a solution. It is used to prevent the creation of circuit involving more than one vertex as explained now. For each pair of vertices $\texttt{NODES}[i], \texttt{NODES}[j]$ $(i, j \in [1, |\texttt{NODES}|])$ of the `NODES` collection we create a reified constraint of the form $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[j].\texttt{index} \wedge i \neq j \Rightarrow R_i < R_j$. The purpose of this constraint is to express the fact that, if there is an arc from vertex $\texttt{NODES}[i]$ to another vertex $\texttt{NODES}[j]$, then $R_i$ should be strictly less than $R_j$.
2. For each vertex $\texttt{NODES}[i]$ $(i \in [1, |\texttt{NODES}|])$ of the `NODES` collection we create a 0-1 variable $B_i$ and state the following reified constraint $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[i].\texttt{index} \Leftrightarrow B_i$ in order to force variable $B_i$ to be set to value 1 if and only if there is a loop on vertex $\texttt{NODES}[i]$. Finally we create a constraint $\texttt{NTREES} = B_1 + B_2 + \cdots + B_{|\texttt{NODES}|}$ for stating the fact that the number of trees is equal to the number of loops of the graph.

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 3 | 16 | 125 | 1296 | 16807 | 262144 | 4782969 |

Number of solutions for `tree`: domains $0..n$

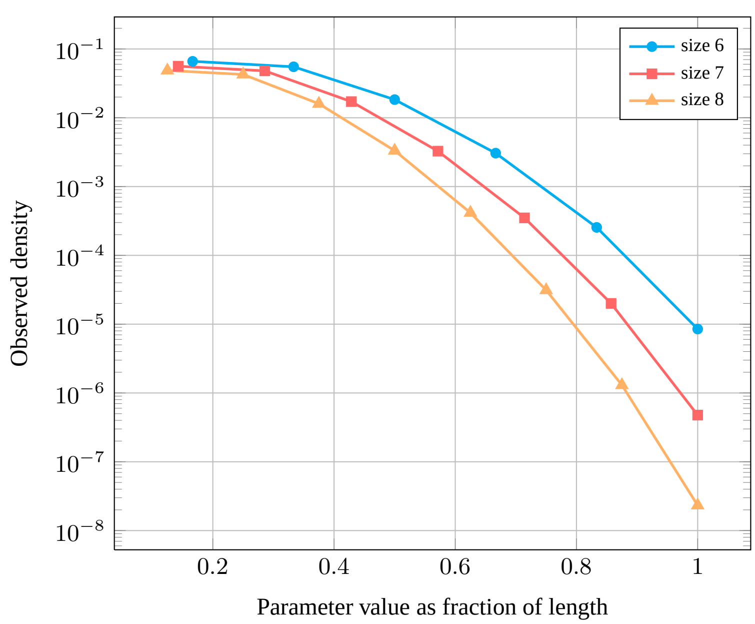

Solution density for tree
Observed density
$10^{-0.5}$
$10^{-0.6}$
$10^{-0.7}$
$10^{-0.8}$
$10^{-0.9}$
$10^{-1}$
2 3 4 5 6 7 8
Length

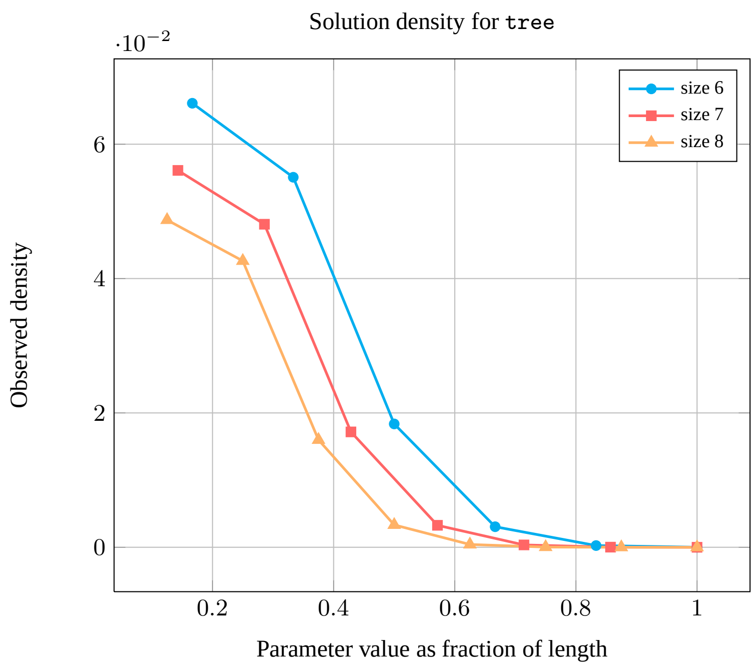

Solution density for tree
Observed density
0.35
0.3
0.25
0.2
0.15
0.1
2 3 4 5 6 7 8
Length

| Length ($n$) | | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Total | | 3 | 16 | 125 | 1296 | 16807 | 262144 | 4782969 |
| Parameter value | 1 | 2 | 9 | 64 | 625 | 7776 | 117649 | 2097152 |
| | 2 | 1 | 6 | 48 | 500 | 6480 | 100842 | 1835008 |
| | 3 | - | 1 | 12 | 150 | 2160 | 36015 | 688128 |
| | 4 | - | - | 1 | 20 | 360 | 6860 | 143360 |
| | 5 | - | - | - | 1 | 30 | 735 | 17920 |
| | 6 | - | - | - | - | 1 | 42 | 1344 |
| | 7 | - | - | - | - | - | 1 | 56 |
| | 8 | - | - | - | - | - | - | 1 |

Solution count for `tree`: domains $0..n$

Solution density for `tree`






**Systems** tree in **Choco**.

**See also** **common keyword:** cycle, graph_crossing, map *(graph partitioning constraint)*, proper_forest *(connected component,tree)*.

**implied by:** binary_tree.

**implies (items to collection):** atleast_nvector.

**related:** balance_tree *(counting number of trees versus controlling how balanced the trees are)*, global_cardinality_low_up_no_loop, global_cardinality_no_loop *(can be used for restricting number of children since discard loops associated with tree roots)*.

**shift of concept:** stable_compatibility, tree_range, tree_resource.

**specialisation:** binary_tree *(no limit on the number of children replaced by at most two children)*, path *(no limit on the number of children replaced by at most one child)*.

**uses in its reformulation:** tree_range, tree_resource.

**Keywords** **constraint type:** graph constraint, graph partitioning constraint.

**filtering:** DFS-bottleneck, strong articulation point, arc-consistency.

**final graph structure:** connected component, tree, one_succ.

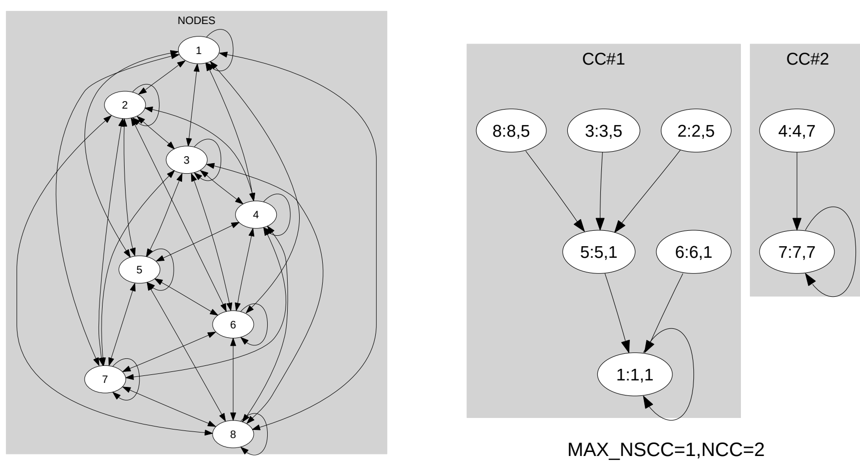
**modelling:** functional dependency.

| | |
|---|---|
| **Arc input(s)** | `NODES` |
| **Arc generator** | *CLIQUE* $\mapsto$ `collection(nodes1,nodes2)` |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `nodes1.succ` = `nodes2.index` |
| **Graph property(ies)** | • **MAX_NSCC** $\leq 1$<br>• **NCC** = `NTREES` |

**Graph model**

We use the graph property **MAX_NSCC** $\leq 1$ in order to specify the fact that the size of the largest strongly connected component should not exceed one. In fact each root of a tree is a strongly connected component with a single vertex. The second graph property **NCC** = `NTREES` enforces the number of trees to be equal to the number of connected components.

Parts (A) and (B) of Figure 5.784 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NCC** graph property, we display the two connected components of the final graph. Each of them corresponds to a tree. The `tree` constraint holds since all strongly connected components of the final graph have no more than one vertex and since `NTREES` = **NCC** = 2.




**(A)** **(B)**

Figure 5.784: Initial and final graph of the `tree` constraint

## 5.405 tree_range

DESCRIPTION LINKS GRAPH

**Origin** Derived from `tree`.

**Constraint** `tree_range(NTREES, R, NODES)`

**Arguments**

```
NTREES : dvar
R      : dvar
NODES  : collection(index-int, succ-dvar)
```

**Restrictions**

$\texttt{NTREES} \geq 0$
$\texttt{R} \geq 0$
$\texttt{R} < |\texttt{NODES}|$
$|\texttt{NODES}| > 0$
$\texttt{required}(\texttt{NODES}, [\texttt{index}, \texttt{succ}])$
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
$\texttt{distinct}(\texttt{NODES}, \texttt{index})$
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$

**Purpose** Cover the digraph $G$ described by the `NODES` collection with `NTREES` trees in such a way that each vertex of $G$ belongs to one distinct tree. `R` is the difference between the longest and the shortest paths (from a leaf to a root) of the final graph.

**Example**

$$\left(2, 1, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-5, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-7, \\ \texttt{index}-5 & \texttt{succ}-1, \\ \texttt{index}-6 & \texttt{succ}-1, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-5 \end{array} \right\rangle \right)$$

The `tree_range` constraint holds since the graph associated with the items of the `NODES` collection corresponds to two trees (i.e., $\texttt{NTREES} = 2$): each tree respectively involves the vertices $\{1, 2, 3, 5, 6, 8\}$ and $\{4, 7\}$. Furthermore $\texttt{R} = 1$ is set to the difference between the longest path (for instance $2 \rightarrow 5 \rightarrow 1$) and the shortest path (for instance $4 \rightarrow 7$) from a leaf to a root. Figure 5.785 provides the two trees associated with the example.

**Typical**

$\texttt{NTREES} < |\texttt{NODES}|$
$|\texttt{NODES}| > 2$

**Symmetry** Items of `NODES` are permutable.

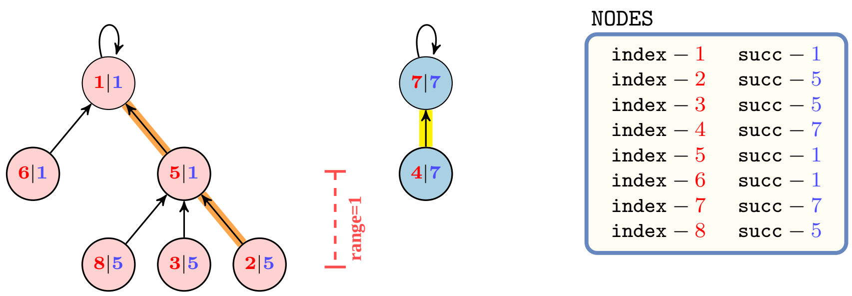



Figure 5.785: The two trees corresponding to the **Example** slot; each vertex contains the information index|succ where succ is the index of its father in the tree (by convention the father of the root is the root itself); the longest and shortest paths from a leaf to a root are respectively shown by thick orange and yellow line segments and have a length of 2 and 1; consequently the range is equal to 1.

**Arg. properties**

- Functional dependency: NTREES determined by NODES.
- Functional dependency: R determined by NODES.

**Reformulation**

By introducing a *distance variable* $D_i$, an *occurrence variable* $O_i$ and a *leave variable* $L_i$ $(1 \leq i \leq |\text{NODES}|)$ for each item $i$ of the NODES collection, where:

- $D_i$ represents the number of vertices from $i$ to the root of the corresponding tree,
- $O_i$ gives the number of occurrences of value $i$ within variables $\text{NODES}[1].\text{succ}, \text{NODES}[2].\text{succ}, \ldots, \text{NODES}[n].\text{succ}$,
- $L_i$ is set to 1 if item $i$ corresponds to a leave (i.e., $O_i > 0$) and 0 otherwise,

the tree_range(NTREES, R, NODES) constraint can be expressed in term of a conjunction of one tree constraint, |NODES| element constraints, |NODES| linear constraints, one global_cardinality constraint, |NODES| reified constraints, one open_minimum, one maximum and one linear constraint, where:

- The tree constraint models the fact that we have a forest of NTREES trees.
- Each element constraint provides the link between the attribute succ of the $i$-th item and the distance variable $D_{\text{NODES}[i].\text{succ}}$ associated with item $\text{NODES}[i].\text{succ}$.
- Each linear constraint associated with the $i$-th item states that the difference between the distance variable $D_i$ and the distance variable $D_{\text{NODES}[i].\text{succ}}$ is equal to 1.
- The global_cardinality constraint provides the number of occurrences $O_i$ of value $i$ $(1 \leq i \leq |\text{NODES}|)$ within variables $\text{NODES}[1].\text{succ}, \text{NODES}[2].\text{succ}, \ldots, \text{NODES}[|\text{NODES}|].\text{succ}$. Note that, when $O_i$ is equal to 0, the corresponding $i$-th item is a leave of one of the NTREES trees.
- Each reified constraint of the form $L_i \Leftrightarrow O_i > 0$ makes the link between the $i$-th occurrence variable $O_i$ and the $i$-th leave variable $L_i$.
- The open_minimum constraint computes the minimum distance MIN from the leaves to the corresponding roots. The leave variable $L_i$ is used in order to select only the distance variables corresponding to leaves.

- The `maximum` constraint computes the maximum distance `MAX` from the vertices to the roots. Since the maximum is achieved by a leave we do not need to focus just on the leaves as it was the case for the minimum distance `MIN`.
- The linear constraint $\mathtt{MAX} - \mathtt{MIN} = \mathtt{R}$ links together argument `R` to the minimum and maximum distances.

With respect to the **Example** slot we get the following conjunction of constraints:

$$
\begin{array}{l}
\texttt{tree}(2, \langle \texttt{index} - 1\ \texttt{succ} - 1,\ \texttt{index} - 2\ \texttt{succ} - 5,\\
\qquad\qquad \texttt{index} - 3\ \texttt{succ} - 5,\ \texttt{index} - 4\ \texttt{succ} - 7,\\
\qquad\qquad \texttt{index} - 5\ \texttt{succ} - 1,\ \texttt{index} - 6\ \texttt{succ} - 1,\\
\qquad\qquad \texttt{index} - 7\ \texttt{succ} - 7,\ \texttt{index} - 8\ \texttt{succ} - 5\rangle),\\
\texttt{domain}(\langle D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8\rangle, 0, 8),\\
DS_1 \in [0,8],\ \texttt{element}(1, \langle 0, D_2, D_3, D_4, D_5, D_6, D_7, D_8\rangle, DS_1),\ D_1 - 0 = 1,\\
DS_2 \in [0,8],\ \texttt{element}(5, \langle 1, 0, D_3, D_4, D_5, D_6, D_7, D_8\rangle, DS_2),\ D_2 - D_5 = 1,\\
DS_3 \in [0,8],\ \texttt{element}(5, \langle 1, D_2, 0, D_4, D_5, D_6, D_7, D_8\rangle, DS_3),\ D_3 - D_5 = 1,\\
DS_4 \in [0,8],\ \texttt{element}(7, \langle 1, D_2, D_3, 0, D_5, D_6, D_7, D_8\rangle, DS_4),\ D_4 - D_7 = 1,\\
DS_5 \in [0,8],\ \texttt{element}(1, \langle 1, D_2, D_3, D_4, 0, D_6, D_7, D_8\rangle, DS_5),\ D_5 - 1 = 1,\\
DS_6 \in [0,8],\ \texttt{element}(1, \langle 1, 3, 3, D_4, 2, 0, D_7, D_8\rangle, DS_6),\ D_6 - 1 = 1,\\
DS_7 \in [0,8],\ \texttt{element}(7, \langle 1, 3, 3, D_4, 2, 2, 0, D_8\rangle, DS_7),\ D_7 - 0 = 1,\\
DS_8 \in [0,8],\ \texttt{element}(5, \langle 1, 3, 3, 2, 2, 2, 1, 0\rangle, DS_8),\ D_8 - 2 = 1,\\
\texttt{global\_cardinality}(\langle 1, 5, 5, 7, 1, 1, 7, 5\rangle,\ \langle \texttt{val} - 1\ \texttt{noccurrence} - 3,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{val} - 2\ \texttt{noccurrence} - 0,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{val} - 3\ \texttt{noccurrence} - 0,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{val} - 4\ \texttt{noccurrence} - 0,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{val} - 5\ \texttt{noccurrence} - 3,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{val} - 6\ \texttt{noccurrence} - 0,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{val} - 7\ \texttt{noccurrence} - 2,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{val} - 8\ \texttt{noccurrence} - 0\rangle),\\
1 \Leftrightarrow 3 > 0, 0 \Leftrightarrow 0 > 0, 0 \Leftrightarrow 0 > 0, 0 \Leftrightarrow 0 > 0,\\
1 \Leftrightarrow 3 > 0, 0 \Leftrightarrow 0 > 0, 1 \Leftrightarrow 2 > 0, 0 \Leftrightarrow 0 > 0,\\
\texttt{open\_minimum}(\texttt{MIN}, \langle \texttt{var} - 3\ \texttt{bool} - 1, \texttt{var} - 0\ \texttt{bool} - 0,\\
\qquad\qquad\qquad\qquad \texttt{var} - 0\ \texttt{bool} - 0, \texttt{var} - 0\ \texttt{bool} - 0,\\
\qquad\qquad\qquad\qquad \texttt{var} - 3\ \texttt{bool} - 1, \texttt{var} - 0\ \texttt{bool} - 0,\\
\qquad\qquad\qquad\qquad \texttt{var} - 2\ \texttt{bool} - 1, \texttt{var} - 0\ \texttt{bool} - 0\rangle),\\
\texttt{maximum}(\texttt{MAX}, \langle 1, 3, 3, 2, 2, 2, 1, 3\rangle),\\
\texttt{MAX} - \texttt{MIN} = \texttt{R} = 1.
\end{array}
$$

**See also**

**related:** `balance` *(balanced tree versus balanced assignment)*.

**root concept:** `tree`.

**used in reformulation:** `domain`, `element`, `global_cardinality`, `maximum`, `open_minimum`, `tree`.

**Keywords**

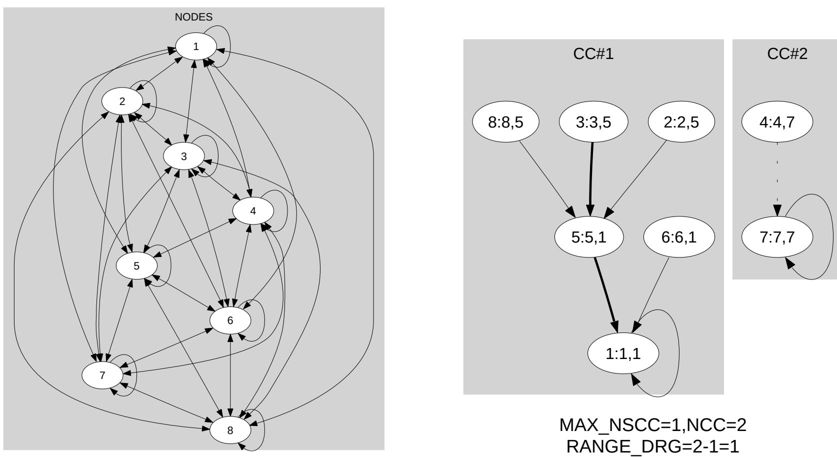
**constraint type:** graph constraint, graph partitioning constraint.

**final graph structure:** connected component, tree.

**modelling:** balanced tree, functional dependency.

| | |
|---|---|
| **Arc input(s)** | NODES |
| **Arc generator** | $CLIQUE \mapsto$ collection(nodes1, nodes2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | nodes1.succ = nodes2.index |
| **Graph property(ies)** | • **MAX_NSCC** $\leq 1$<br>• **NCC** = NTREES<br>• **RANGE_DRG** = R |

**Graph model** Parts (A) and (B) of Figure 5.786 respectively show the initial and final graph associated with the **Example** slot. Since we use the RANGE_DRG graph property, we respectively display the longest and shortest paths of the final graph with a bold and a dash line.




**(A)** **(B)**

Figure 5.786: Initial and final graph of the tree_range constraint

## 5.406 tree_resource

DESCRIPTION LINKS GRAPH

**Origin** Derived from tree.

**Constraint** tree_resource(RESOURCE, TASK)

**Arguments**

RESOURCE : collection(id−int, nb_task−dvar)
TASK : collection(id−int, father−dvar, resource−dvar)

**Restrictions**

$|\text{RESOURCE}| > 0$
required(RESOURCE, [id, nb_task])
RESOURCE.id $\geq 1$
RESOURCE.id $\leq |\text{RESOURCE}|$
distinct(RESOURCE, id)
RESOURCE.nb_task $\geq 0$
RESOURCE.nb_task $\leq |\text{TASK}|$
required(TASK, [id, father, resource])
TASK.id $> |\text{RESOURCE}|$
TASK.id $\leq |\text{RESOURCE}| + |\text{TASK}|$
distinct(TASK, id)
TASK.father $\geq 1$
TASK.father $\leq |\text{RESOURCE}| + |\text{TASK}|$
TASK.resource $\geq 1$
TASK.resource $\leq |\text{RESOURCE}|$

**Purpose**

Cover a digraph $G$ in such a way that each vertex belongs to one distinct tree. Each tree is made up from one *resource* vertex and several *task* vertices. The resource vertices correspond to the roots of the different trees. For each resource a domain variable nb_task indicates how many task-vertices belong to the corresponding tree. For each task a domain variable resource gives the identifier of the resource that can handle the task.

**Example**

$$\left(\begin{array}{l}\left\langle\begin{array}{ll}\text{id}-1 & \text{nb\_task}-4,\\ \text{id}-2 & \text{nb\_task}-0,\\ \text{id}-3 & \text{nb\_task}-1\end{array}\right\rangle,\\ \left\langle\begin{array}{lll}\text{id}-4 & \text{father}-8 & \text{resource}-1,\\ \text{id}-5 & \text{father}-3 & \text{resource}-3,\\ \text{id}-6 & \text{father}-8 & \text{resource}-1,\\ \text{id}-7 & \text{father}-1 & \text{resource}-1,\\ \text{id}-8 & \text{father}-1 & \text{resource}-1\end{array}\right\rangle\end{array}\right)$$

The tree_resource constraint holds since the graph associated with the items of the RESOURCE and the TASK collections corresponds to 3 trees (i.e., $|\text{RESOURCE}| = 3$): each tree respectively involves the vertices $\{1, 4, 6, 7, 8\}$, $\{2\}$ and $\{3, 5\}$. They are depicted by Figure 5.787, where *resource* and *task* vertices are respectively coloured in blue and pink.

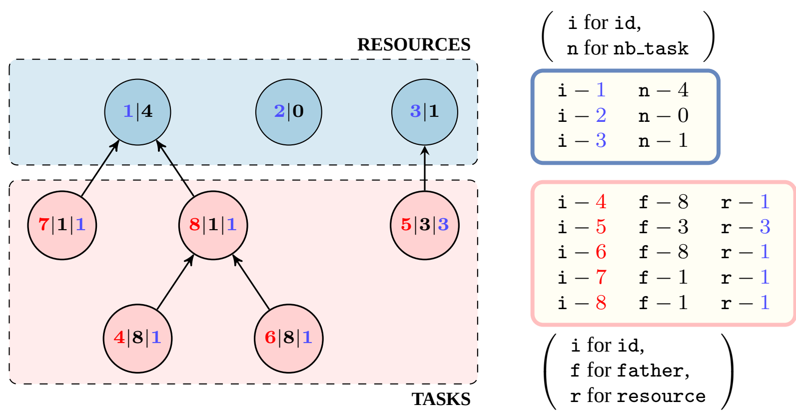



Figure 5.787: The three trees corresponding to the **Example** slot; each resource vertex (in blue) contains the information id|nb_task where nb_task is the number of tasks in the tree, while each task vertex (in pink) contains the information id|father|resource where father is the index of its father in the tree and resource is the index of the corresponding root task in the tree.

**Typical**

$|\mathtt{RESOURCE}| > 0$
$|\mathtt{TASK}| > |\mathtt{RESOURCE}|$

**Symmetries**

- Items of RESOURCE are permutable.
- Items of TASK are permutable.

**Reformulation**

The tree_resource(RESOURCE, TASK) constraint can be expressed in term of a conjunction of one tree constraint, |TASK| element constraints and one global_cardinality constraint:

- The tree constraint expresses the fact that we have a well formed tree.
- The element constraint is used for expressing the link between the father attribute of an item of the TASK collection and its corresponding resource attribute.
- The global_cardinality constraint is used to link the resource attribute of the items of the TASK collection with the nb_task attribute of the items of the RESOURCE collection.

With respect to the **Example** slot we get the following conjunction of constraints:

$$
\begin{array}{l}
\mathtt{tree}(3, \langle \mathtt{index}-1\ \mathtt{succ}-1, \\
\quad \mathtt{index}-2\ \mathtt{succ}-2, \\
\quad \mathtt{index}-3\ \mathtt{succ}-3, \\
\quad \mathtt{index}-4\ \mathtt{succ}-8, \\
\quad \mathtt{index}-5\ \mathtt{succ}-3, \\
\quad \mathtt{index}-6\ \mathtt{succ}-8, \\
\quad \mathtt{index}-7\ \mathtt{succ}-1, \\
\quad \mathtt{index}-8\ \mathtt{succ}-1\rangle),
\end{array}
$$

$\texttt{element}(8, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 1),$
$\texttt{element}(3, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 3),$
$\texttt{element}(8, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 1),$
$\texttt{element}(1, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 1),$
$\texttt{element}(1, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 1),$
$\texttt{global\_cardinality}(\langle 1, 3, 1, 1, 1\rangle,$
$\langle \texttt{val} - 1\ \texttt{noccurrence} - 4,$
$\texttt{val} - 2\ \texttt{noccurrence} - 0,$
$\texttt{val} - 3\ \texttt{noccurrence} - 1\rangle).$

**See also**

**root concept:** `tree`.

**used in reformulation:** `element`, `global_cardinality`, `tree`.

**Keywords**

**characteristic of a constraint:** derived collection.

**constraint type:** graph constraint, resource constraint, graph partitioning constraint.

**final graph structure:** tree, connected component.

**Derived Collection**

$$\texttt{col}\left(\begin{array}{l}\texttt{RESOURCE\_TASK}-\texttt{collection}\left(\begin{array}{l}\texttt{index}-\texttt{int},\\ \texttt{succ}-\texttt{dvar},\\ \texttt{name}-\texttt{dvar}\end{array}\right),\\ \left[\begin{array}{l}\texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{RESOURCE.id},\\ \texttt{succ}-\texttt{RESOURCE.id},\\ \texttt{name}-\texttt{RESOURCE.id}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{TASK.id},\\ \texttt{succ}-\texttt{TASK.father},\\ \texttt{name}-\texttt{TASK.resource}\end{array}\right)\end{array}\right]\end{array}\right)$$

**Arc input(s)** RESOURCE_TASK

**Arc generator** $CLIQUE \mapsto \texttt{collection}(\texttt{resource\_task1}, \texttt{resource\_task2})$

**Arc arity** 2

**Arc constraint(s)**
- $\texttt{resource\_task1.succ} = \texttt{resource\_task2.index}$
- $\texttt{resource\_task1.name} = \texttt{resource\_task2.name}$

**Graph property(ies)**
- $\mathbf{MAX\_NSCC} \leq 1$
- $\mathbf{NCC} = |\texttt{RESOURCE}|$
- $\mathbf{NVERTEX} = |\texttt{RESOURCE}| + |\texttt{TASK}|$

For all items of RESOURCE:

**Arc input(s)** RESOURCE_TASK

**Arc generator** $CLIQUE \mapsto \texttt{collection}(\texttt{resource\_task1}, \texttt{resource\_task2})$

**Arc arity** 2

**Arc constraint(s)**
- $\texttt{resource\_task1.succ} = \texttt{resource\_task2.index}$
- $\texttt{resource\_task1.name} = \texttt{resource\_task2.name}$
- $\texttt{resource\_task1.name} = \texttt{RESOURCE.id}$

**Graph property(ies)** $\mathbf{NVERTEX} = \texttt{RESOURCE.nb\_task} + 1$

**Graph model** For the second graph constraint, part (A) of Figure 5.788 shows the initial graphs associated with resources 1, 2 and 3 of the **Example** slot. For the second graph constraint, part (B) of Figure 5.788 shows the corresponding final graphs associated with resources 1, 2 and 3. Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold. To each resource corresponds a tree of respectively 4, 0 and 1 task-vertices.

**Signature** Since the initial graph of the first graph constraint contains $|\texttt{RESOURCE}| + |\texttt{TASK}|$ vertices, the corresponding final graph cannot have more than $|\texttt{RESOURCE}| + |\texttt{TASK}|$ vertices. Therefore we can rewrite the graph property $\mathbf{NVERTEX} = |\texttt{RESOURCE}| + |\texttt{TASK}|$ to $\mathbf{NVERTEX} \geq |\texttt{RESOURCE}| + |\texttt{TASK}|$ and simplify $\underline{\mathbf{NVERTEX}}$ to $\overline{\mathbf{NVERTEX}}$.

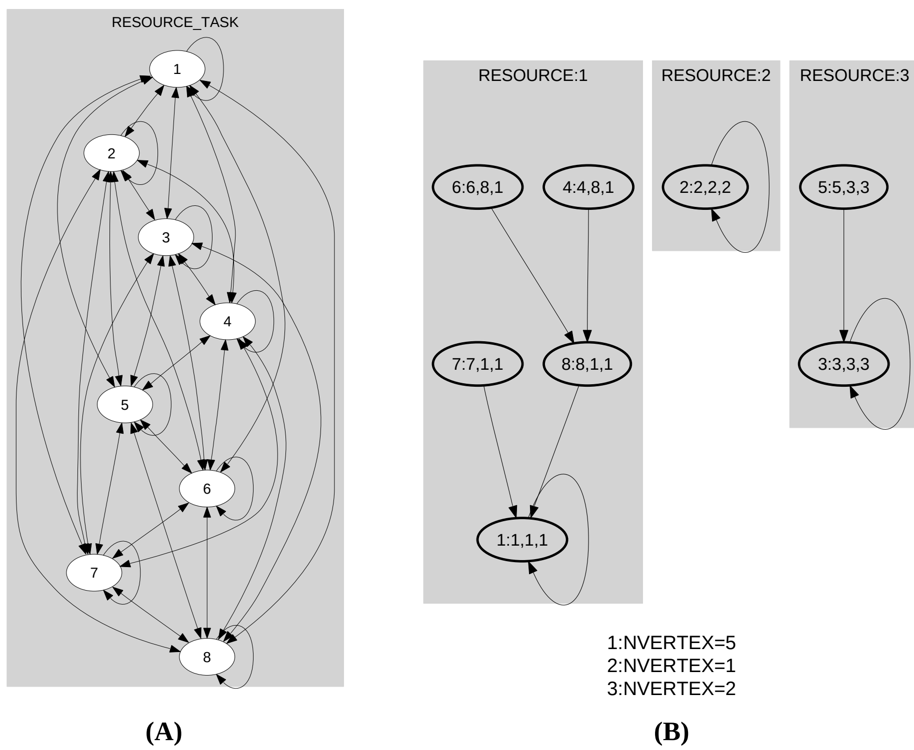



Figure 5.788: Initial and final graph of the `tree_resource` constraint

## 5.407 twin

DESCRIPTION LINKS

**Origin**

Pairs of variables related by hiden element constraints sharing the same table.

**Constraint**

twin(PAIRS)

**Argument**

PAIRS : collection(x−dvar, y−dvar)

**Restrictions**

required(PAIRS, x)
required(PAIRS, y)
$|\text{PAIRS}| > 0$

**Purpose**

Enforce the condition $\text{PAIRS}[i].\text{x} = u \wedge \text{PAIRS}[i].\text{y} = v \ (i \in [1, |\text{PAIRS}|]) \Rightarrow \forall j \in [1, |\text{PAIRS}|] : \text{PAIRS}[j].\text{x} = u \Leftrightarrow \text{PAIRS}[j].\text{y} = v$.

**Example**

$$\left(\left\langle\begin{array}{ll} \text{x}-1 & \text{y}-8, \\ \text{x}-9 & \text{y}-6, \\ \text{x}-1 & \text{y}-8, \\ \text{x}-5 & \text{y}-0, \\ \text{x}-6 & \text{y}-7, \\ \text{x}-9 & \text{y}-6 \end{array}\right\rangle\right)$$

The twin constraint holds since 1 is paired with 8, 9 is paired with 6, 5 is paired with 0, 6 is paired with 7.

**Typical**

$|\text{PAIRS}| > 1$
$|\text{PAIRS}| >$ nval(PAIRS.x)
$|\text{PAIRS}| >$ nval(PAIRS.y)
nval(PAIRS.x) $> 1$
nval(PAIRS.y) $> 1$
nval(PAIRS.x) $=$ nval(PAIRS.y)
nval(PAIRS.x) $< |\text{PAIRS}|$
nval(PAIRS.y) $< |\text{PAIRS}|$

**Arg. properties**

Contractible wrt. PAIRS.

**See also**

**implied by:** circuit, derangement, proper_circuit, symmetric_alldifferent_loop.

**related:** element *(pairs linked by an element with the same table)*.

**Keywords**

**characteristic of a constraint:** pair.

**constraint type:** predefined constraint.

## 5.408 two_layer_edge_crossing

DESCRIPTION LINKS GRAPH

**Origin** Inspired by [201].

**Constraint**

$$\texttt{two\_layer\_edge\_crossing}\left(\begin{array}{l}\texttt{NCROSS},\\ \texttt{VERTICES\_LAYER1},\\ \texttt{VERTICES\_LAYER2},\\ \texttt{EDGES}\end{array}\right)$$

**Arguments**

$$\begin{array}{lll}\texttt{NCROSS} & : & \texttt{dvar}\\ \texttt{VERTICES\_LAYER1} & : & \texttt{collection}(\texttt{id}-\texttt{int},\texttt{pos}-\texttt{dvar})\\ \texttt{VERTICES\_LAYER2} & : & \texttt{collection}(\texttt{id}-\texttt{int},\texttt{pos}-\texttt{dvar})\\ \texttt{EDGES} & : & \texttt{collection}(\texttt{id}-\texttt{int},\texttt{vertex1}-\texttt{int},\texttt{vertex2}-\texttt{int})\end{array}$$

**Restrictions**

$\texttt{NCROSS} \geq 0$
$\texttt{required}(\texttt{VERTICES\_LAYER1},[\texttt{id},\texttt{pos}])$
$\texttt{VERTICES\_LAYER1.id} \geq 1$
$\texttt{VERTICES\_LAYER1.id} \leq |\texttt{VERTICES\_LAYER1}|$
$\texttt{distinct}(\texttt{VERTICES\_LAYER1},\texttt{id})$
$\texttt{distinct}(\texttt{VERTICES\_LAYER1},\texttt{pos})$
$\texttt{required}(\texttt{VERTICES\_LAYER2},[\texttt{id},\texttt{pos}])$
$\texttt{VERTICES\_LAYER2.id} \geq 1$
$\texttt{VERTICES\_LAYER2.id} \leq |\texttt{VERTICES\_LAYER2}|$
$\texttt{distinct}(\texttt{VERTICES\_LAYER2},\texttt{id})$
$\texttt{distinct}(\texttt{VERTICES\_LAYER2},\texttt{pos})$
$\texttt{required}(\texttt{EDGES},[\texttt{id},\texttt{vertex1},\texttt{vertex2}])$
$\texttt{EDGES.id} \geq 1$
$\texttt{EDGES.id} \leq |\texttt{EDGES}|$
$\texttt{distinct}(\texttt{EDGES},\texttt{id})$
$\texttt{EDGES.vertex1} \geq 1$
$\texttt{EDGES.vertex1} \leq |\texttt{VERTICES\_LAYER1}|$
$\texttt{EDGES.vertex2} \geq 1$
$\texttt{EDGES.vertex2} \leq |\texttt{VERTICES\_LAYER2}|$

**Purpose** NCROSS is the number of line segments intersections.

**Example**

$$\left(\begin{array}{l}2,\langle\texttt{id}-1\ \texttt{pos}-1,\texttt{id}-2\ \texttt{pos}-2\rangle,\\ \langle\texttt{id}-1\ \texttt{pos}-3,\texttt{id}-2\ \texttt{pos}-1,\texttt{id}-3\ \texttt{pos}-2\rangle,\\ \left\langle\begin{array}{lll}\texttt{id}-1 & \texttt{vertex1}-2 & \texttt{vertex2}-2,\\ \texttt{id}-2 & \texttt{vertex1}-2 & \texttt{vertex2}-3,\\ \texttt{id}-3 & \texttt{vertex1}-1 & \texttt{vertex2}-1\end{array}\right\rangle\end{array}\right)$$

Figure 5.789 provides a picture of the example, where one can see the two line segments intersections. Each line segment of Figure 5.789 is labelled with its identifier and corresponds to an item of the EDGES collection. The two vertices on top of Figure 5.789

correspond to the items of the VERTICES_LAYER1 collection, while the three other vertices are associated with the items of VERTICES_LAYER2.

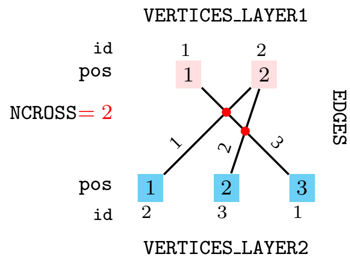



Figure 5.789: Intersection between line segments joining two layers of the **Example** slot for the constraint two_layer_edge_crossing(NCROSS, VERTICES_LAYER1, VERTICES_LAYER2, EDGES)

**Typical**

$|$VERTICES_LAYER1$| > 1$
$|$VERTICES_LAYER2$| > 1$
$|$EDGES$| \geq |$VERTICES_LAYER1$|$
$|$EDGES$| \geq |$VERTICES_LAYER2$|$

**Symmetries**

- Arguments are permutable w.r.t. permutation (NCROSS) (VERTICES_LAYER1, VERTICES_LAYER2) (EDGES).
- Items of VERTICES_LAYER1 are permutable.
- Items of VERTICES_LAYER2 are permutable.

**Arg. properties**

Functional dependency: NCROSS determined by VERTICES_LAYER1, VERTICES_LAYER2 and EDGES.

**Remark**

The two-layer edge crossing minimisation problem was proved to be NP-hard in [184].

**See also**

**common keyword:** crossing, graph_crossing *(line segments intersection)*.

**Keywords**

**characteristic of a constraint:** derived collection.

**constraint arguments:** pure functional dependency.

**geometry:** geometrical constraint, line segments intersection.

**miscellaneous:** obscure.

**modelling:** functional dependency.

**Derived Collection**

$$\text{col}\left(\begin{array}{l}\texttt{EDGES\_EXTREMITIES}-\texttt{collection}(\texttt{layer1}-\texttt{dvar},\texttt{layer2}-\texttt{dvar}),\\ \left[\ \texttt{item}\left(\begin{array}{l}\texttt{layer1}-\texttt{EDGES.vertex1}(\texttt{VERTICES\_LAYER1},\texttt{pos},\texttt{id}),\\ \texttt{layer2}-\texttt{EDGES.vertex2}(\texttt{VERTICES\_LAYER2},\texttt{pos},\texttt{id})\end{array}\right)\right]\end{array}\right)$$

**Arc input(s)** EDGES_EXTREMITIES

**Arc generator** $CLIQUE(<) \mapsto$ collection(edges_extremities1, edges_extremities2)

**Arc arity** 2

**Arc constraint(s)**

$$\bigvee\left(\begin{array}{l}\bigwedge\left(\begin{array}{l}\texttt{edges\_extremities1.layer1} < \texttt{edges\_extremities2.layer1},\\ \texttt{edges\_extremities1.layer2} > \texttt{edges\_extremities2.layer2}\end{array}\right),\\ \bigwedge\left(\begin{array}{l}\texttt{edges\_extremities1.layer1} > \texttt{edges\_extremities2.layer1},\\ \texttt{edges\_extremities1.layer2} < \texttt{edges\_extremities2.layer2}\end{array}\right)\end{array}\right)$$

**Graph property(ies)** $\mathbf{NARC}$= NCROSS

**Graph model**

As usual for the two-layer edge crossing problem [201], [22], positions of the vertices on each layer are represented as a permutation of the vertices. We generate a derived collection that, for each edge, contains the position of its extremities on both layers. In the arc generator we use the restriction $<$ in order to generate a single arc for each pair of segments. This is required, since otherwise we would count more than once a line segments intersection.

Parts (A) and (B) of Figure 5.790 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.

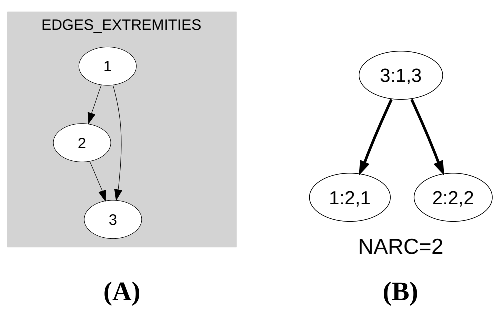



Figure 5.790: Initial and final graph of the two_layer_edge_crossing constraint

## 5.409 two_orth_are_in_contact

DESCRIPTION LINKS GRAPH AUTOMATON

**Origin** [358], used for defining orths_are_connected.

**Constraint** two_orth_are_in_contact(ORTHOTOPE1, ORTHOTOPE2)

**Type** ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

**Arguments**
ORTHOTOPE1 : ORTHOTOPE
ORTHOTOPE2 : ORTHOTOPE

**Restrictions**
|ORTHOTOPE| $>0$
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz $>0$
ORTHOTOPE.ori $\leq$ ORTHOTOPE.end
|ORTHOTOPE1| = |ORTHOTOPE2|
orth_link_ori_siz_end(ORTHOTOPE1)
orth_link_ori_siz_end(ORTHOTOPE2)

**Purpose**
Enforce the following conditions on two orthotopes $O_1$ and $O_2$:

- For all dimensions $i$, except one dimension, the projections of $O_1$ and $O_2$ onto $i$ have a non-empty intersection.
- For all dimensions $i$, the distance between the projections of $O_1$ and $O_2$ onto $i$ is equal to $0$.

**Example**
$$\left(\begin{array}{l}\langle \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4, \texttt{ori}-5\ \texttt{siz}-2\ \texttt{end}-7\rangle,\\ \langle \texttt{ori}-3\ \texttt{siz}-2\ \texttt{end}-5, \texttt{ori}-2\ \texttt{siz}-3\ \texttt{end}-5\rangle\end{array}\right)$$

Figure 5.791 shows the two rectangles of the example. The two_orth_are_in_contact constraint holds since the two rectangles are in contact: the contact is depicted by a pink line-segment.

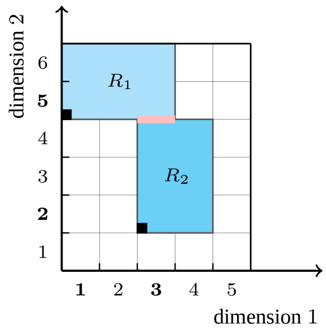
**Typical** |ORTHOTOPE| $>1$

**Symmetries**
- Arguments are permutable w.r.t. permutation (ORTHOTOPE1, ORTHOTOPE2).
- Items of ORTHOTOPE1 and ORTHOTOPE2 are permutable (*same permutation used*).

**Used in** orths_are_connected.

**See also** **implies:** two_orth_do_not_overlap.

**Keywords** **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.




$R_1$: $\langle \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4, \texttt{ori}-5\ \texttt{siz}-2\ \texttt{end}-7\rangle$
$R_2$: $\langle \texttt{ori}-3\ \texttt{siz}-2\ \texttt{end}-5, \texttt{ori}-2\ \texttt{siz}-3\ \texttt{end}-5\rangle$

`ORTHOTOPES` (rectangles)

Figure 5.791: The two rectangles that are in contact of the **Example** slot where the contact is shown in pink

**constraint network structure:** Berge-acyclic constraint network.

**constraint type:** logic.

**filtering:** arc-consistency.

**geometry:** geometrical constraint, touch, contact, non-overlapping, orthotope.

| | |
|---|---|
| **Arc input(s)** | ORTHOTOPE1 ORTHOTOPE2 |
| **Arc generator** | $PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • orthotope1.end > orthotope2.ori <br> • orthotope2.end > orthotope1.ori |
| **Graph property(ies)** | $\mathbf{NARC} = \|\text{ORTHOTOPE1}\| - 1$ |

| | |
|---|---|
| **Arc input(s)** | ORTHOTOPE1 ORTHOTOPE2 |
| **Arc generator** | $PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | $\texttt{max}\left(0, \begin{array}{l}\texttt{max}(\texttt{orthotope1.ori}, \texttt{orthotope2.ori}) - \\ \texttt{min}(\texttt{orthotope1.end}, \texttt{orthotope2.end})\end{array}\right) = 0$ |
| **Graph property(ies)** | $\mathbf{NARC} = \|\text{ORTHOTOPE1}\|$ |

**Graph model** Parts (A) and (B) of Figure 5.792 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the unique arc of the final graph is stressed in bold. It corresponds to the fact that the projection onto dimension 1 of the two rectangles of the example overlap.

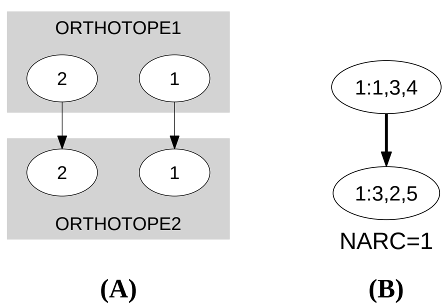



Figure 5.792: Initial and final graph of the `two_orth_are_in_contact` constraint

**Signature** Consider the second graph constraint. Since we use the arc generator $PRODUCT(=)$ on the collections ORTHOTOPE1 and ORTHOTOPE2, and because of the restriction $|\text{ORTHOTOPE1}| = |\text{ORTHOTOPE2}|$, the maximum number of arcs of the corresponding final graph is equal to $|\text{ORTHOTOPE1}|$. Therefore we can rewrite the graph property $\mathbf{NARC} = |\text{ORTHOTOPE1}|$ to $\mathbf{NARC} \geq |\text{ORTHOTOPE1}|$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

**Automaton** Figure 5.793 depicts the automaton associated with the `two_orth_are_in_contact` constraint. Let $\mathtt{ORI1}_i$, $\mathtt{SIZ1}_i$ and $\mathtt{END1}_i$ respectively be the `ori`, the `siz` and the `end` attributes of the $i^{th}$ item of the `ORTHOTOPE1` collection. Let $\mathtt{ORI2}_i$, $\mathtt{SIZ2}_i$ and $\mathtt{END2}_i$ respectively be the `ori`, the `siz` and the `end` attributes of the $i^{th}$ item of the `ORTHOTOPE2` collection. To each sextuple $(\mathtt{ORI1}_i, \mathtt{SIZ1}_i, \mathtt{END1}_i, \mathtt{ORI2}_i, \mathtt{SIZ2}_i, \mathtt{END2}_i)$ corresponds a signature variable $S_i$, which takes its value in $\{0, 1, 2\}$, as well as the following signature constraint:

$$((\mathtt{SIZ1}_i > 0) \wedge (\mathtt{SIZ2}_i > 0) \wedge (\mathtt{END1}_i > \mathtt{ORI2}_i) \wedge (\mathtt{END2}_i > \mathtt{ORI1}_i)) \Leftrightarrow S_i = 0$$

$$((\mathtt{SIZ1}_i > 0) \wedge (\mathtt{SIZ2}_i > 0) \wedge (\mathtt{END1}_i = \mathtt{ORI2}_i \vee \mathtt{END2}_i = \mathtt{ORI1}_i)) \Leftrightarrow S_i = 1.$$

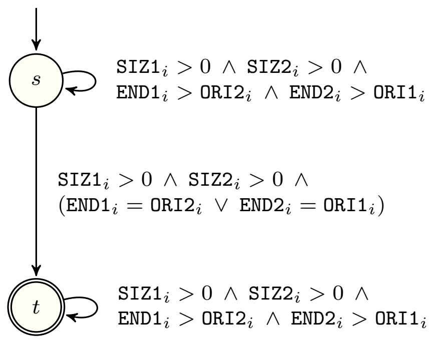



Figure 5.793: Automaton of the `two_orth_are_in_contact` constraint

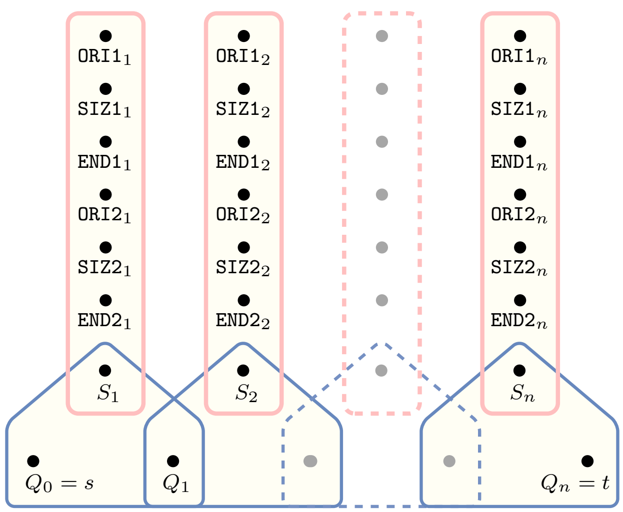



Figure 5.794: Hypergraph of the reformulation corresponding to the automaton of the `two_orth_are_in_contact` constraint

## 5.410 two_orth_column

DESCRIPTION LINKS GRAPH

**Origin** Used for defining `diffn_column`.

**Constraint** `two_orth_column(ORTHOTOPE1, ORTHOTOPE2, DIM)`

**Type** ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

**Arguments**

| | | |
|---|---|---|
| ORTHOTOPE1 | : | ORTHOTOPE |
| ORTHOTOPE2 | : | ORTHOTOPE |
| DIM | : | int |

**Restrictions**

$|\texttt{ORTHOTOPE}| > 0$
require_at_least(2, ORTHOTOPE, [ori, siz, end])
$\texttt{ORTHOTOPE.siz} \geq 0$
$\texttt{ORTHOTOPE.ori} \leq \texttt{ORTHOTOPE.end}$
$|\texttt{ORTHOTOPE1}| = |\texttt{ORTHOTOPE2}|$
orth_link_ori_siz_end(ORTHOTOPE1)
orth_link_ori_siz_end(ORTHOTOPE2)
$\texttt{DIM} > 0$
$\texttt{DIM} \leq |\texttt{ORTHOTOPE1}|$

**Purpose** Let $P_1$ and $P_2$ respectively denote the projections of ORTHOTOPE1 and ORTHOTOPE2 onto dimension DIM. If $P_1$ and $P_2$ overlap then the size of their intersection is equal to the size of ORTHOTOPE1 in dimension DIM, as well as to the size of ORTHOTOPE2 in dimension DIM.

**Example**

$$\left(\begin{array}{l}\langle \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4, \texttt{ori}-1\ \texttt{siz}-1\ \texttt{end}-2\rangle,\\ \langle \texttt{ori}-4\ \texttt{siz}-2\ \texttt{end}-6, \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4\rangle, 1\end{array}\right)$$

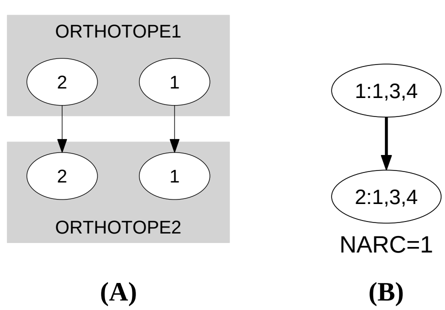



Figure 5.795: Initial and final graph of the `two_orth_column` constraint

**Typical** $|\texttt{ORTHOTOPE}| > 1$

**Symmetry** Arguments are permutable w.r.t. permutation $(\texttt{ORTHOTOPE1}, \texttt{ORTHOTOPE2})$ $(\texttt{DIM})$.

**Used in** `diffn_column`.

**See also** **implies:** `two_orth_include`.

**related:** `diffn` *(an extension of the* `diffn` *constraint).*

**Keywords** **constraint type:** logic.

**geometry:** geometrical constraint, positioning constraint, orthotope, guillotine cut.

| | |
|---|---|
| **Arc input(s)** | ORTHOTOPE1 ORTHOTOPE2 |
| **Arc generator** | $PRODUCT(=) \mapsto \mathtt{collection}(\mathtt{orthotope1}, \mathtt{orthotope2})$ |
| **Arc arity** | 2 |
| **Arc constraint(s)** | $\bigwedge\left(\begin{array}{l}\mathtt{orthotope1.key} = \mathtt{DIM},\\ \mathtt{orthotope1.ori} < \mathtt{orthotope2.end},\\ \mathtt{orthotope2.ori} < \mathtt{orthotope1.end},\\ \mathtt{orthotope1.siz} > 0,\\ \mathtt{orthotope2.siz} > 0\end{array}\right) \Rightarrow \bigwedge\left(\begin{array}{l}\begin{array}{l}\mathtt{min}(\mathtt{orthotope1.end}, \mathtt{orthotope2.end}) -\\ \mathtt{max}(\mathtt{orthotope1.ori}, \mathtt{orthotope2.ori})\end{array} = \\ \mathtt{orthotope1.siz} \\ \mathtt{orthotope1.siz} = \mathtt{orthotope2.siz}\end{array},\right)$ |
| **Graph property(ies)** | $\mathbf{NARC} = 1$ |

## 5.411 two_orth_do_not_overlap

DESCRIPTION LINKS GRAPH AUTOMATON

**Origin**

Used for defining diffn.

**Constraint**

two_orth_do_not_overlap(ORTHOTOPE1, ORTHOTOPE2)

**Type**

ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

**Arguments**

ORTHOTOPE1 : ORTHOTOPE
ORTHOTOPE2 : ORTHOTOPE

**Restrictions**

|ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori ≤ ORTHOTOPE.end
|ORTHOTOPE1| = |ORTHOTOPE2|
orth_link_ori_siz_end(ORTHOTOPE1)
orth_link_ori_siz_end(ORTHOTOPE2)

**Purpose**

For two orthotopes $O_1$ and $O_2$ enforce that there exists at least one dimension $i$ such that the projections on $i$ of $O_1$ and $O_2$ do not overlap.

**Example**

$$\left(\begin{array}{l}\langle \texttt{ori}-2\ \texttt{siz}-2\ \texttt{end}-4, \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4\rangle, \\ \langle \texttt{ori}-4\ \texttt{siz}-4\ \texttt{end}-8, \texttt{ori}-3\ \texttt{siz}-3\ \texttt{end}-6\rangle\end{array}\right)$$

Figure 5.796 represents the respective position of the two rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold. The two_orth_do_not_overlap constraint holds since the two rectangles do not overlap.

**Typical**

|ORTHOTOPE| > 1

**Symmetries**

- Arguments are permutable w.r.t. permutation (ORTHOTOPE1, ORTHOTOPE2).
- Items of ORTHOTOPE1 and ORTHOTOPE2 are permutable (*same permutation used*).
- ORTHOTOPE1.siz can be decreased to any value ≥ 0.
- ORTHOTOPE2.siz can be decreased to any value ≥ 0.

**Used in**

diffn.

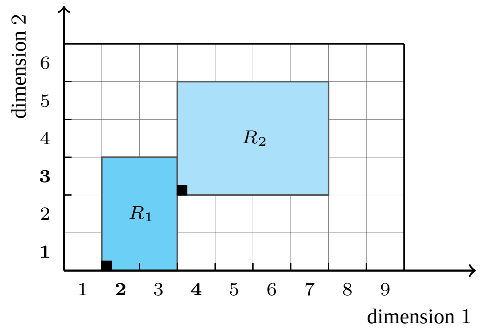
**See also**

**implied by:** two_orth_are_in_contact.

**Keywords**

**characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

**constraint network structure:** Berge-acyclic constraint network.




| | |
|---|---|
| $R_1$: | $\langle$ori − 2 siz − 2 end − 4, ori − 1 siz − 3 end − 4$\rangle$ |
| $R_2$: | $\langle$ori − 4 siz − 4 end − 8, ori − 3 siz − 3 end − 6$\rangle$ |

`ORTHOTOPES` (rectangles)

Figure 5.796: The two non overlapping rectangles of the **Example** slot

**constraint type:** logic.

**filtering:** arc-consistency, constructive disjunction.

**final graph structure:** bipartite, no loop.

**geometry:** geometrical constraint, non-overlapping, orthotope.

| | |
|---|---|
| **Arc input(s)** | ORTHOTOPE1 ORTHOTOPE2 |
| **Arc generator** | $SYMMETRIC_PRODUCT(=) \mapsto$collection(orthotope1, orthotope2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | orthotope1.end $\leq$ orthotope2.ori $\vee$ orthotope1.siz $= 0$ |
| **Graph property(ies)** | $\mathbf{NARC} \geq 1$ |
| **Graph class** | • BIPARTITE<br>• NO_LOOP |

**Graph model** We build an initial graph where each arc corresponds to the fact that, either the projection of an orthotope on a given dimension is empty, either it is located before the projection in the same dimension of the other orthotope. Finally we ask that at least one arc constraint remains in the final graph.

Parts (A) and (B) of Figure 5.797 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold. It corresponds to the fact that the projection in dimension 1 of the first orthotope is located before the projection in dimension 1 of the second orthotope. Therefore the two orthotopes do not overlap.

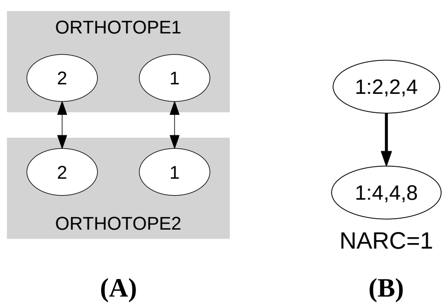



Figure 5.797: Initial and final graph of the two_orth_do_not_overlap constraint

**Automaton** Figure 5.798 depicts the automaton associated with the `two_orth_do_not_overlap` constraint. Let $\mathtt{ORI1}_i$, $\mathtt{SIZ1}_i$ and $\mathtt{END1}_i$ respectively be the `ori`, the `siz` and the `end` attributes of the $i^{th}$ item of the `ORTHOTOPE1` collection. Let $\mathtt{ORI2}_i$, $\mathtt{SIZ2}_i$ and $\mathtt{END2}_i$ respectively be the `ori`, the `siz` and the `end` attributes of the $i^{th}$ item of the `ORTHOTOPE2` collection. To each sextuple $(\mathtt{ORI1}_i, \mathtt{SIZ1}_i, \mathtt{END1}_i, \mathtt{ORI2}_i, \mathtt{SIZ2}_i, \mathtt{END2}_i)$ corresponds a 0-1 signature variable $S_i$ as well as the following signature constraint: $((\mathtt{SIZ1}_i > 0) \wedge (\mathtt{SIZ2}_i > 0) \wedge (\mathtt{END1}_i > \mathtt{ORI2}_i) \wedge (\mathtt{END2}_i > \mathtt{ORI1}_i)) \Leftrightarrow S_i$.

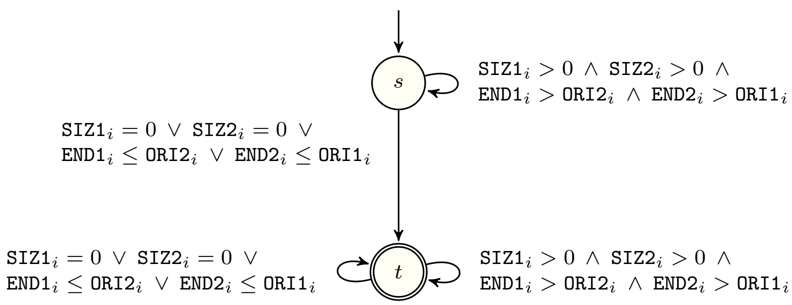



Figure 5.798: Automaton of the `two_orth_do_not_overlap` constraint

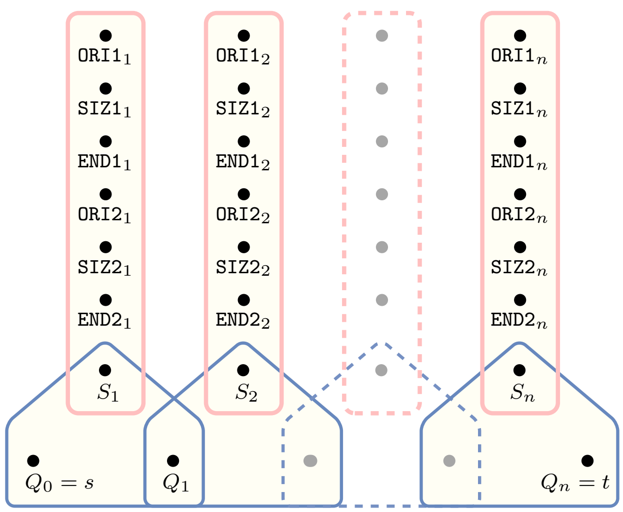



Figure 5.799: Hypergraph of the reformulation corresponding to the automaton of the `two_orth_do_not_overlap` constraint

## 5.412 two_orth_include

**DESCRIPTION** **LINKS** **GRAPH**

**Origin**

Used for defining `diffn_include`.

**Constraint**

`two_orth_include(ORTHOTOPE1, ORTHOTOPE2, DIM)`

**Type**

`ORTHOTOPE : collection(ori-dvar, siz-dvar, end-dvar)`

**Arguments**

| | | |
|---|---|---|
| `ORTHOTOPE1` | : | `ORTHOTOPE` |
| `ORTHOTOPE2` | : | `ORTHOTOPE` |
| `DIM` | : | `int` |

**Restrictions**

$|\texttt{ORTHOTOPE}| > 0$
$\texttt{require\_at\_least}(2, \texttt{ORTHOTOPE}, [\texttt{ori}, \texttt{siz}, \texttt{end}])$
$\texttt{ORTHOTOPE.siz} \geq 0$
$\texttt{ORTHOTOPE.ori} \leq \texttt{ORTHOTOPE.end}$
$|\texttt{ORTHOTOPE1}| = |\texttt{ORTHOTOPE2}|$
$\texttt{orth\_link\_ori\_siz\_end}(\texttt{ORTHOTOPE1})$
$\texttt{orth\_link\_ori\_siz\_end}(\texttt{ORTHOTOPE2})$
$\texttt{DIM} > 0$
$\texttt{DIM} \leq |\texttt{ORTHOTOPE1}|$

**Purpose**

Let $P_1$ and $P_2$ respectively denote the projections of `ORTHOTOPE1` and `ORTHOTOPE2` onto dimension `DIM`. If $P_1$ and $P_2$ overlap then, either $P_1$ is included in $P_2$, either $P_2$ is included in $P_1$.

**Example**

$$\left(\begin{array}{l}\langle \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4, \texttt{ori}-1\ \texttt{siz}-1\ \texttt{end}-2\rangle, \\ \langle \texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3, \texttt{ori}-2\ \texttt{siz}-3\ \texttt{end}-5\rangle, 1\end{array}\right)$$

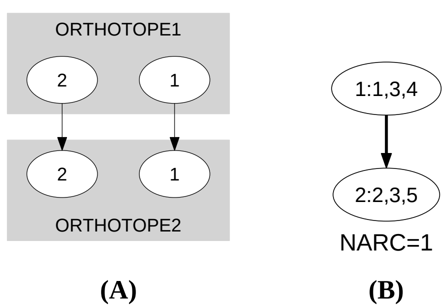



Figure 5.800: Initial and final graph of the `two_orth_include` constraint

**Typical** $|\mathtt{ORTHOTOPE}| > 1$

**Symmetry** Arguments are permutable w.r.t. permutation $(\mathtt{ORTHOTOPE1}, \mathtt{ORTHOTOPE2})$ $(\mathtt{DIM})$.

**Used in** `diffn_include`.

**See also** **implied by:** `two_orth_column`.

**related:** `diffn` *(an extension of the* `diffn` *constraint).*

**Keywords** **constraint type:** logic.

**geometry:** geometrical constraint, positioning constraint, orthotope.

| | |
|---|---|
| **Arc input(s)** | ORTHOTOPE1 ORTHOTOPE2 |
| **Arc generator** | $PRODUCT(=) \mapsto \texttt{collection}(\texttt{orthotope1}, \texttt{orthotope2})$ |
| **Arc arity** | 2 |
| **Arc constraint(s)** | $\bigwedge\begin{pmatrix} \texttt{orthotope1.key} = \texttt{DIM}, \\ \texttt{orthotope1.ori} < \texttt{orthotope2.end}, \\ \texttt{orthotope2.ori} < \texttt{orthotope1.end}, \\ \texttt{orthotope1.siz} > 0, \\ \texttt{orthotope2.siz} > 0 \end{pmatrix} \Rightarrow$ $\begin{matrix} \texttt{min}(\texttt{orthotope1.end}, \texttt{orthotope2.end}) - \\ \texttt{max}(\texttt{orthotope1.ori}, \texttt{orthotope2.ori}) \end{matrix} =$ $\texttt{min}(\texttt{orthotope1.siz}, \texttt{orthotope2.siz})$ |
| **Graph property(ies)** | $\textbf{NARC} = 1$ |

## 5.413 used_by

DESCRIPTION LINKS GRAPH AUTOMATON

**Origin** N. Beldiceanu

**Constraint** `used_by(VARIABLES1, VARIABLES2)`

**Arguments**

`VARIABLES1 : collection(var−dvar)`
`VARIABLES2 : collection(var−dvar)`

**Restrictions**

`|VARIABLES1| ≥ |VARIABLES2|`
`required(VARIABLES1, var)`
`required(VARIABLES2, var)`

**Purpose** All the values of the variables of collection `VARIABLES2` are used by the variables of collection `VARIABLES1`.

**Example** $(\langle 1,9,1,5,2,1\rangle, \langle 1,1,2,5\rangle)$

The `used_by` constraint holds since, for each value occurring within the collection `VARIABLES2` $= \langle 1,1,2,5\rangle$, its number of occurrences within `VARIABLES1` $= \langle 1,9,1,5,2,1\rangle$ is greater than or equal to its number of occurrences within `VARIABLES2`:

- Value 1 occurs 3 times within $\langle 1,9,1,5,2,1\rangle$ and 2 times within $\langle 1,1,2,5\rangle$.
- Value 2 occurs 1 times within $\langle 1,9,1,5,2,1\rangle$ and 1 times within $\langle 1,1,2,5\rangle$.
- Value 5 occurs 1 times within $\langle 1,9,1,5,2,1\rangle$ and 1 times within $\langle 1,1,2,5\rangle$.

**All solutions** Figure 5.801 gives all solutions to the following non ground instance of the `used_by` constraint: $U_1 \in \{1,5\}$, $U_2 \in [1,2]$, $U_3 \in [1,2]$, $V_1 \in [0,2]$, $V_2 \in [2,4]$, `used_by`$(\langle U_1,U_2,U_3\rangle, \langle V_1,V_2\rangle)$.

① $(\langle 1,1,2\rangle, \langle 1,2\rangle)$
② $(\langle 1,2,1\rangle, \langle 1,2\rangle)$
③ $(\langle 1,2,2\rangle, \langle 1,2\rangle)$
④ $(\langle 1,2,2\rangle, \langle 2,2\rangle)$
⑤ $(\langle 5,1,2\rangle, \langle 1,2\rangle)$
⑥ $(\langle 5,2,1\rangle, \langle 1,2\rangle)$
⑦ $(\langle 5,2,2\rangle, \langle 2,2\rangle)$

Figure 5.801: All solutions corresponding to the non ground example of the `used_by` constraint of the **All solutions** slot where identical values are coloured in the same way in both collections

**Typical**

$|\texttt{VARIABLES1}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$|\texttt{VARIABLES2}| > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

**Arg. properties**

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union).

**Algorithm**

As described in [47] we can pad VARIABLES2 with dummy variables such that its cardinality will be equal to that cardinality of VARIABLES1. The domain of a dummy variable contains all of the values. Then, we have a same constraint between the two sets. Direct arc-consistency and bound-consistency algorithms based on a flow model are also proposed in [47, 49, 231]. The leftmost part of Figure 3.31 illustrates this flow model.

More recently [129, 130] presents a second filtering algorithm also achieving arc-consistency based on a mapping of the solutions to the used_by constraint to var-perfect matchings[16] in a bipartite intersection graph derived from the domain of the variables of the constraint in the following way. To each variable of the VARIABLES1 and VARIABLES2 collection corresponds a vertex of the intersection graph. There is an edge between a vertex associated with a variable of the VARIABLES1 collection and a vertex associated with a variable of the VARIABLES2 collection if and only if the corresponding variables have at least one value in common in their domains.

**Reformulation**

The $\texttt{used\_by}(\langle \texttt{var}-U_1\ \texttt{var}-U_2, \ldots, \texttt{var}-U_{|\texttt{VARIABLES1}|}\rangle, \langle \texttt{var}-V_1\ \texttt{var}-V_2, \ldots, \texttt{var}-V_{|\texttt{VARIABLES2}|}\rangle)$ constraint can be expressed in term of a conjunction of $|\texttt{VARIABLES2}|$ reified constraints of the form:

$$\sum_{1\leq j\leq |\texttt{VARIABLES1}|}(V_i = U_j) \geq \sum_{1\leq j\leq |\texttt{VARIABLES2}|}(V_i = V_j)\ (i \in [1, |\texttt{VARIABLES2}|]).$$

**Used in**

int_value_precede_chain, k_used_by.

**See also**

**generalisation:** used_by_interval *(*variable *replaced by* variable/constant*)*, used_by_modulo *(*variable *replaced by* variable mod constant*)*, used_by_partition *(*variable *replaced by* variable ∈ partition*)*.

**implied by:** same.

**implies:** uses.

**soft variant:** soft_used_by_var *(variable-based violation measure)*.

**system of constraints:** k_used_by.

[16] A *var-perfect matching* is a maximum matching covering all vertices corresponding to the variables of VARIABLES2.

**Keywords** **characteristic of a constraint:** sort based reformulation, automaton, automaton with array of counters.

**combinatorial object:** multiset.

**constraint arguments:** constraint between two collections of variables.

**filtering:** flow, bipartite matching, arc-consistency, bound-consistency, DFS-bottleneck.

**modelling:** inclusion.

| | |
|---|---|
| **Arc input(s)** | `VARIABLES1` `VARIABLES2` |
| **Arc generator** | $PRODUCT$ ↦ `collection(variables1, variables2)` |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `variables1.var` = `variables2.var` |
| **Graph property(ies)** | • for all connected components: **NSOURCE**≥**NSINK**<br>• **NSINK**= \|`VARIABLES2`\| |

**Graph model** Parts (A) and (B) of Figure 5.802 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable assigned to value $9$ was removed from the final graph since there is no arc for which the associated equality constraint holds. The `used_by` constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to |`VARIABLES2`|.

**Signature** Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to |`VARIABLES2`|. Therefore we can rewrite **NSINK** = |`VARIABLES2`| to **NSINK** ≥ |`VARIABLES2`| and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.

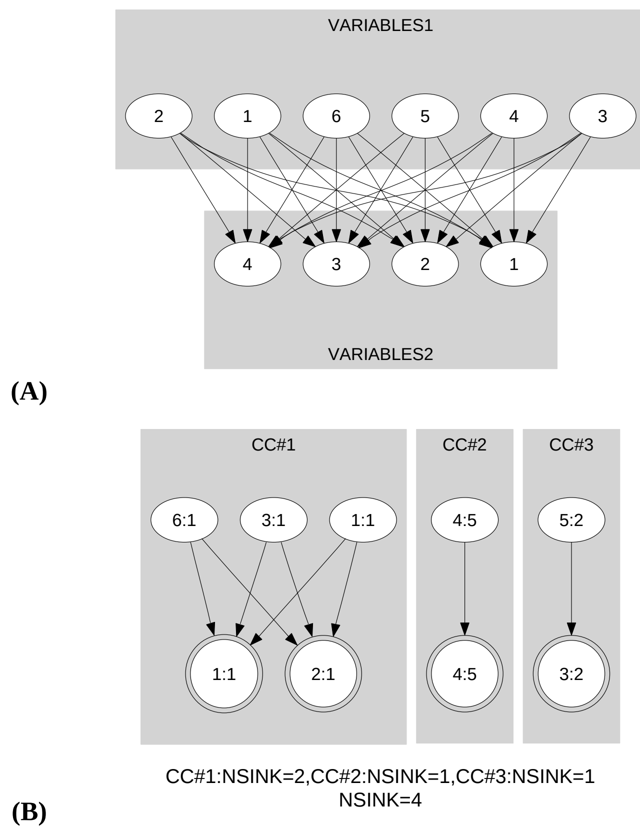



Figure 5.802: Initial and final graph of the `used_by` constraint

**Automaton** Figure 5.803 depicts the automaton associated with the `used_by` constraint. To each item of the collection `VARIABLES1` corresponds a signature variable $S_i$ that is equal to $0$. To each item of the collection `VARIABLES2` corresponds a signature variable $S_{i+|\mathtt{VARIABLES1}|}$ that is equal to $1$.

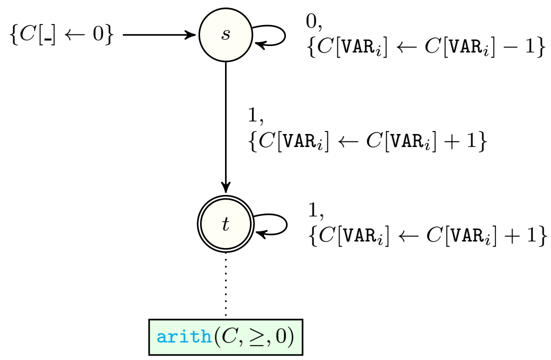



Figure 5.803: Automaton of the `used_by` constraint

## 5.414 used_by_interval

DESCRIPTION LINKS GRAPH

**Origin**

Derived from `used_by`.

**Constraint**

`used_by_interval(VARIABLES1, VARIABLES2, SIZE_INTERVAL)`

**Arguments**

```
VARIABLES1    : collection(var−dvar)
VARIABLES2    : collection(var−dvar)
SIZE_INTERVAL : int
```

**Restrictions**

$|\texttt{VARIABLES1}| \geq |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\texttt{SIZE\_INTERVAL} > 0$

**Purpose**

Let $N_i$ (respectively $M_i$) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval $[\texttt{SIZE\_INTERVAL} \cdot i, \texttt{SIZE\_INTERVAL} \cdot i + \texttt{SIZE\_INTERVAL} - 1]$. For all integer $i$ we have $M_i > 0 \Rightarrow N_i \geq M_i$.

**Example**

$(\langle 1, 9, 1, 8, 6, 2 \rangle, \langle 1, 0, 7, 7 \rangle, 3)$

In the example, the third argument $\texttt{SIZE\_INTERVAL} = 3$ defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where $k$ is an integer. Consequently the values of the collection $\texttt{VARIABLES2} = \langle 1, 0, 7, 7 \rangle$ are respectively located within intervals $[0, 2]$, $[0, 2]$, $[6, 8]$, $[6, 8]$. Therefore intervals $[0, 2]$ and $[6, 8]$ are respectively used 2 and 2 times.

Similarly, the values of the collection $\texttt{VARIABLES1} = \langle 1, 9, 1, 8, 6, 2 \rangle$ are respectively located within intervals $[0, 2]$, $[9, 11]$, $[0, 2]$, $[6, 8]$, $[6, 8]$, $[0, 2]$. Therefore intervals $[0, 2]$, $[6, 8]$ and $[9, 11]$ are respectively used 3, 2 and 1 times.

Consequently, the `used_by_interval` constraint holds since, for each interval associated with the collection $\texttt{VARIABLES2} = \langle 1, 0, 7, 7 \rangle$, its number of occurrences within $\texttt{VARIABLES1} = \langle 1, 9, 1, 8, 6, 2 \rangle$ is greater than or equal to its number of occurrences within VARIABLES2:

- Interval $[0, 2]$ occurs 3 times within $\langle 1, 9, 1, 8, 6, 2 \rangle$ and 2 times within $\langle 1, 0, 7, 7 \rangle$.
- Interval $[6, 8]$ occurs 2 times within $\langle 1, 9, 1, 8, 6, 2 \rangle$ and 2 times within $\langle 1, 0, 7, 7 \rangle$.

**Typical**

$|\texttt{VARIABLES1}| > 1$
$\text{range}(\texttt{VARIABLES1.var}) > 1$
$|\texttt{VARIABLES2}| > 1$
$\text{range}(\texttt{VARIABLES2.var}) > 1$
$\texttt{SIZE\_INTERVAL} > 1$
$\texttt{SIZE\_INTERVAL} < \text{range}(\texttt{VARIABLES1.var})$
$\texttt{SIZE\_INTERVAL} < \text{range}(\texttt{VARIABLES2.var})$

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var 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 VARIABLES2.var that belongs to the $k$-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

**Arg. properties**

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union), SIZE_INTERVAL(id).

**Reformulation**

The used_by_interval$(\langle \texttt{var}-U_1\ \texttt{var}-U_2,\ldots,\texttt{var}-U_{|\texttt{VARIABLES1}|}\rangle, \langle \texttt{var}-V_1\ \texttt{var}-V_2,\ldots,\texttt{var}-V_{|\texttt{VARIABLES2}|}\rangle, \texttt{SIZE_INTERVAL})$ constraint can be expressed by introducing $|\texttt{VARIABLES1}|+|\texttt{VARIABLES2}|$ *quotient* variables
$U_i = \texttt{SIZE\_INTERVAL}\cdot P_i + R_i,\ R_i \in [0, \texttt{SIZE\_INTERVAL}-1]\ (i \in [1, |\texttt{VARIABLES1}|])$,
$V_i = \texttt{SIZE\_INTERVAL}\cdot Q_i + S_i,\ S_i \in [0, \texttt{SIZE\_INTERVAL}-1]\ (i \in [1, |\texttt{VARIABLES2}|])$,
in term of a conjunction of $|\texttt{VARIABLES2}|$ reified constraints of the form:
$\sum_{1\le j\le|\texttt{VARIABLES1}|}(Q_i = P_j) \ge \sum_{1\le j\le|\texttt{VARIABLES2}|}(Q_i = Q_j)\ (i \in [1, |\texttt{VARIABLES2}|])$.

**Used in**

k_used_by_interval.

**See also**

**implied by:** same_interval.

**soft variant:** soft_used_by_interval_var (*variable-based violation measure*).

**specialisation:** used_by (variable/constant *replaced by* variable).

**system of constraints:** k_used_by_interval.

**Keywords**

**characteristic of a constraint:** sort based reformulation.

**constraint arguments:** constraint between two collections of variables.

**modelling:** inclusion, interval.

| | |
|---|---|
| **Arc input(s)** | VARIABLES1 VARIABLES2 |
| **Arc generator** | $PRODUCT \mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL |
| **Graph property(ies)** | • for all connected components: **NSOURCE**$\geq$**NSINK**<br>• **NSINK**$= \lvert$VARIABLES2$\rvert$ |

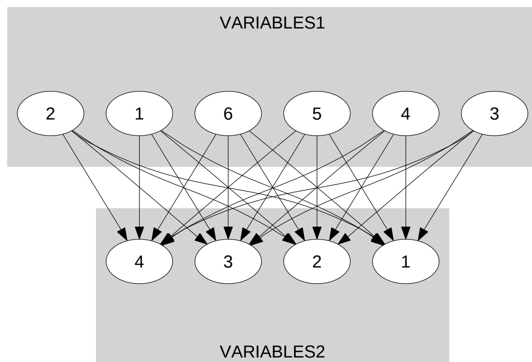
**Graph model**

Parts (A) and (B) of Figure 5.804 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value $9$ was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The used_by_interval constraint holds since:

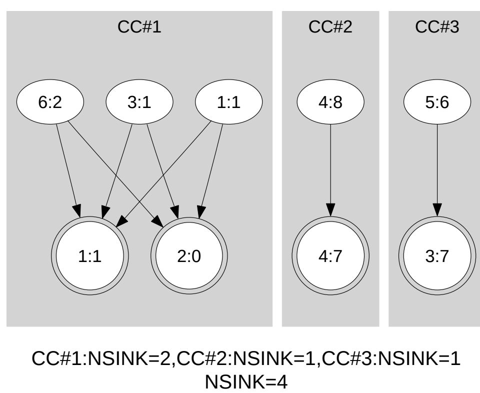
- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to $\lvert$VARIABLES2$\rvert$.

**Signature**

Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to $\lvert$VARIABLES2$\rvert$. Therefore we can rewrite **NSINK** $= \lvert$VARIABLES2$\rvert$ to **NSINK** $\geq \lvert$VARIABLES2$\rvert$ and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.




**(A)**




**(B)**

Figure 5.804: Initial and final graph of the `used_by_interval` constraint

## 5.415 used_by_modulo

DESCRIPTION LINKS GRAPH

**Origin** Derived from used_by.

**Constraint** used_by_modulo(VARIABLES1, VARIABLES2, M)

**Arguments**

| | | |
|---|---|---|
| VARIABLES1 | : | collection(var−dvar) |
| VARIABLES2 | : | collection(var−dvar) |
| M | : | int |

**Restrictions**

$|\mathtt{VARIABLES1}| \geq |\mathtt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\mathtt{M} > 0$

**Purpose**

For each integer $R$ in $[0, \mathtt{M}-1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have $R$ as a rest when divided by M. For all $R$ in $[0, \mathtt{M}-1]$ we have $N2_R > 0 \Rightarrow N1_R \geq N2_R$.

**Example**

$(\langle 1, 9, 4, 5, 2, 1\rangle, \langle 7, 1, 2, 5\rangle, 3)$

The values of the collection VARIABLES2 $= \langle 7, 1, 2, 5\rangle$ are respectively associated with the equivalence classes $7 \bmod 3 = 1$, $1 \bmod 3 = 1$, $2 \bmod 3 = 2$, $5 \bmod 3 = 2$. Therefore the equivalence classes 1 and 2 are respectively used 2 and 2 times.

Similarly, the values of the collection VARIABLES1 $= \langle 1, 9, 4, 5, 2, 1\rangle$ associated with the equivalence classes $1 \bmod 3 = 1$, $9 \bmod 3 = 0$, $4 \bmod 3 = 1$, $5 \bmod 3 = 2$, $2 \bmod 3 = 2$, $1 \bmod 3 = 1$. Therefore the equivalence classes 0, 1 and 2 are respectively used 1, 3 and 2 times.

Consequently, the used_by_modulo constraint holds since, for each equivalence class associated with the collection VARIABLES2 $= \langle 7, 1, 2, 5\rangle$, its number of occurrences within VARIABLES1 $= \langle 1, 9, 4, 5, 2, 1\rangle$ is greater than or equal to its number of occurrences within VARIABLES2:

- The equivalence class 1 occurs 3 times within $\langle 1, 9, 4, 5, 2, 1\rangle$ and 2 times within $\langle 7, 1, 2, 5\rangle$.
- The equivalence class 2 occurs 2 times within $\langle 1, 9, 4, 5, 2, 1\rangle$ and 2 times within $\langle 7, 1, 2, 5\rangle$.

**Typical**

$|\mathtt{VARIABLES1}| > 1$
range(VARIABLES1.var) $> 1$
$|\mathtt{VARIABLES2}| > 1$
range(VARIABLES2.var) $> 1$
$\mathtt{M} > 1$
M < maxval(VARIABLES1.var)
M < maxval(VARIABLES2.var)

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value $u$ of VARIABLES1.var can be replaced by any other value $v$ such that $v$ is congruent to $u$ modulo M.
- An occurrence of a value $u$ of VARIABLES2.var can be replaced by any other value $v$ such that $v$ is congruent to $u$ modulo M.

**Arg. properties**

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union), M(id).

**Used in** k_used_by_modulo.

**See also**

**implied by:** same_modulo.

**soft variant:** soft_used_by_modulo_var (*variable-based violation measure*).

**specialisation:** used_by (variable mod constant *replaced by* variable).

**system of constraints:** k_used_by_modulo.

**Keywords**

**characteristic of a constraint:** modulo, sort based reformulation.

**constraint arguments:** constraint between two collections of variables.

**modelling:** inclusion.

| | |
|---|---|
| **Arc input(s)** | `VARIABLES1` `VARIABLES2` |
| **Arc generator** | *PRODUCT* $\mapsto$ `collection`(`variables1`, `variables2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | `variables1.var` mod `M` = `variables2.var` mod `M` |
| **Graph property(ies)** | • for all connected components: **NSOURCE** $\geq$ **NSINK**<br>• **NSINK** $=$ \|`VARIABLES2`\| |

**Graph model** Parts (A) and (B) of Figure 5.805 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value $9$ was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The `used_by_modulo` constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to |`VARIABLES2`|.

**Signature** Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to |`VARIABLES2`|. Therefore we can rewrite **NSINK** $=$ |`VARIABLES2`| to **NSINK** $\geq$ |`VARIABLES2`| and simplify $\overline{\underline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.

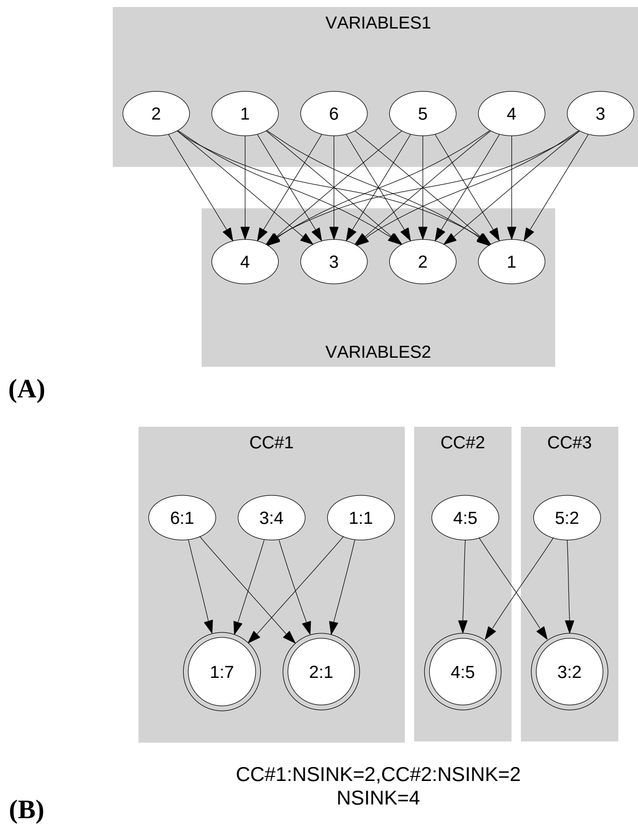



Figure 5.805: Initial and final graph of the `used_by_modulo` constraint

## 5.416 used_by_partition

DESCRIPTION LINKS GRAPH

**Origin** Derived from `used_by`.

**Constraint** `used_by_partition(VARIABLES1, VARIABLES2, PARTITIONS)`

**Type** `VALUES : collection(val-int)`

**Arguments**

```
VARIABLES1 : collection(var-dvar)
VARIABLES2 : collection(var-dvar)
PARTITIONS : collection(p - VALUES)
```

**Restrictions**

$|\texttt{VALUES}| \geq 1$
`required(VALUES, val)`
`distinct(VALUES, val)`
$|\texttt{VARIABLES1}| \geq |\texttt{VARIABLES2}|$
`required(VARIABLES1, var)`
`required(VARIABLES2, var)`
`required(PARTITIONS, p)`
$|\texttt{PARTITIONS}| \geq 2$

**Purpose** For each integer $i$ in $[1, |\texttt{PARTITIONS}|]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of `VARIABLES1` (respectively `VARIABLES2`) that take their value in the $i^{th}$ partition of the collection `PARTITIONS`. For all $i$ in $[1, |\texttt{PARTITIONS}|]$ we have $N2_i > 0 \Rightarrow N1_i \geq N2_i$.

**Example**

$$\left(\begin{array}{l} \langle 1,9,1,6,2,3\rangle, \\ \langle 1,3,6,6\rangle, \\ \langle \texttt{p}-\langle 1,3\rangle, \texttt{p}-\langle 4\rangle, \texttt{p}-\langle 2,6\rangle\rangle \end{array}\right)$$

The different values of the collection `VARIABLES2` $= \langle 1,3,6,6\rangle$ are respectively associated with the partitions `p` $-\langle 1,3\rangle$, `p` $-\langle 1,3\rangle$, `p` $-\langle 2,6\rangle$, and `p` $-\langle 2,6\rangle$. Therefore partitions `p` $-\langle 1,3\rangle$ and `p` $-\langle 2,6\rangle$ are respectively used 2 and 2 times.

Similarly, the different values of the collection `VARIABLES1` $= \langle 1,9,1,6,2,3\rangle$ (except value 9, which does not occur in any partition) are respectively associated with the partitions `p` $-\langle 1,3\rangle$, `p` $-\langle 1,3\rangle$, `p` $-\langle 2,6\rangle$, `p` $-\langle 2,6\rangle$, and `p` $-\langle 1,3\rangle$. Therefore partitions `p` $-\langle 1,3\rangle$ and `p` $-\langle 2,6\rangle$ are respectively used 3 and 2 times.

Consequently, the `used_by_partition` constraint holds since, for each partition associated with the collection `VARIABLES2` $= \langle 1,3,6,6\rangle$, its number of occurrences within `VARIABLES1` $= \langle 1,9,1,6,2,3\rangle$ is greater than or equal to its number of occurrences within `VARIABLES2`:

- Partition `p` $-\langle 1,3\rangle$ occurs 3 times within $\langle 1,9,1,6,2,3\rangle$ and 2 times within $\langle 1,3,6,6\rangle$.

- Partition p $= \langle 2, 6\rangle$ occurs 2 times within $\langle 1, 9, 1, 6, 2, 3\rangle$ and 2 times within $\langle 1, 3, 6, 6\rangle$.

**Typical**

$|\texttt{VARIABLES1}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$|\texttt{VARIABLES2}| > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$
$|\texttt{VARIABLES1}| > |\texttt{PARTITIONS}|$
$|\texttt{VARIABLES2}| > |\texttt{PARTITIONS}|$

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

**Arg. properties**

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union), PARTITIONS(id).

**Used in** k_used_by_partition.

**See also**

**implied by:** same_partition.

**soft variant:** soft_used_by_partition_var *(variable-based violation measure)*.

**specialisation:** used_by *(*variable $\in$ partition *replaced by* variable*)*.

**system of constraints:** k_used_by_partition.

**used in graph description:** in_same_partition.

**Keywords**

**characteristic of a constraint:** partition, sort based reformulation.

**constraint arguments:** constraint between two collections of variables.

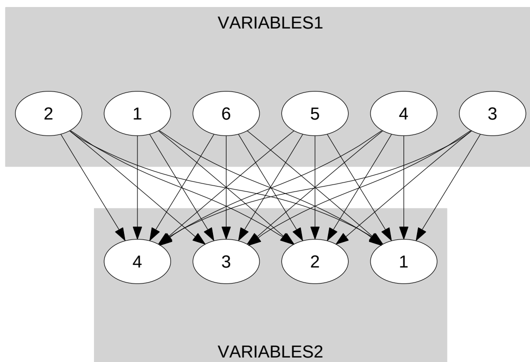
**modelling:** inclusion.

| | |
|---|---|
| **Arc input(s)** | VARIABLES1 VARIABLES2 |
| **Arc generator** | $PRODUCT$ ↦ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | in_same_partition(variables1.var, variables2.var, PARTITIONS) |
| **Graph property(ies)** | • for all connected components: **NSOURCE** ≥ **NSINK**<br>• **NSINK** = \|VARIABLES2\| |

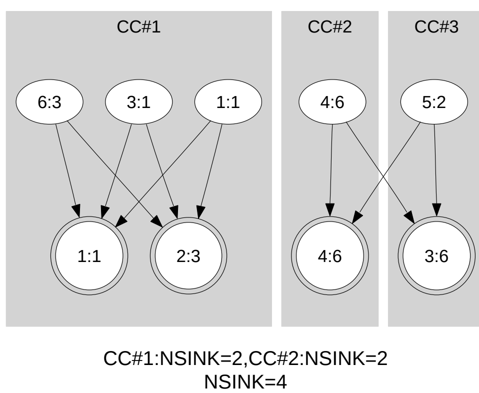
**Graph model** Parts (A) and (B) of Figure 5.806 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value $9$ was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The used_by_partition constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to |VARIABLES2|.

**Signature** Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to |VARIABLES2|. Therefore we can rewrite **NSINK** $=$ |VARIABLES2| to **NSINK** $\geq$ |VARIABLES2| and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.




**(A)**




**(B)**

Figure 5.806: Initial and final graph of the `used_by_partition` constraint

## 5.417 uses

DESCRIPTION LINKS GRAPH

**Origin** [63]

**Constraint** uses(VARIABLES1, VARIABLES2)

**Arguments**

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

**Restrictions**

$\min(1, |\texttt{VARIABLES1}|) \geq \min(1, |\texttt{VARIABLES2}|)$
required(VARIABLES1, var)
required(VARIABLES2, var)

**Purpose**

The set of values assigned to the variables of the collection of variables VARIABLES2 is included within the set of values assigned to the variables of the collection of variables VARIABLES1.

**Example**

$(\langle 3, 3, 4, 6\rangle, \langle 3, 4, 4, 4, 4\rangle)$

The uses constraint holds since the set of values $\{3, 4\}$ assigned to the items of collection $\langle 3, 4, 4, 4, 4\rangle$ is included within the set of values $\{3, 4, 6\}$ occurring within $\langle 3, 3, 4, 6\rangle$.

**All solutions**

Figure 5.807 gives all solutions to the following non ground instance of the uses constraint: $\texttt{U}_1 \in [0, 1]$, $\texttt{U}_2 \in [1, 2]$, $\texttt{V}_1 \in [0, 2]$, $\texttt{V}_2 \in [2, 4]$, $\texttt{V}_3 \in [2, 4]$, uses$(\langle \texttt{U}_1, \texttt{U}_2\rangle, \langle \texttt{V}_1, \texttt{V}_2, \texttt{V}_3\rangle)$.

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

Figure 5.807: All solutions corresponding to the non ground example of the uses constraint of the **All solutions** slot where identical values are coloured in the same way in both collections

**Typical**

$|\texttt{VARIABLES1}| > 1$
range(VARIABLES1.var) $> 1$
$|\texttt{VARIABLES2}| > 1$
range(VARIABLES2.var) $> 1$
$|\texttt{VARIABLES1}| \leq |\texttt{VARIABLES2}|$

**Symmetries**

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

**Arg. properties**

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union).

**Remark**

It was shown in [63] that, finding out whether a uses constraint has a solution or not is NP-hard. This was achieved by reduction from 3-SAT.

**See also**

**generalisation:** common.

**implied by:** used_by.

**related:** roots.

**Keywords**

**complexity:** 3-SAT.

**constraint arguments:** constraint between two collections of variables.

**final graph structure:** acyclic, bipartite, no loop.

**modelling:** inclusion.

| | |
|---|---|
| **Arc input(s)** | VARIABLES1 VARIABLES2 |
| **Arc generator** | *PRODUCT* $\mapsto$ collection(variables1, variables2) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables1.var = variables2.var |
| **Graph property(ies)** | **NSINK**= \|VARIABLES2\| |
| **Graph class** | • ACYCLIC<br>• BIPARTITE<br>• NO_LOOP |

**Graph model** Parts (A) and (B) of Figure 5.808 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK** graph property, the sink vertices of the final graph are stressed with a double circle. Note that all the vertices corresponding to the variables that take values 9 or 2 were removed from the final graph since there is no arc for which the associated equality constraint holds.

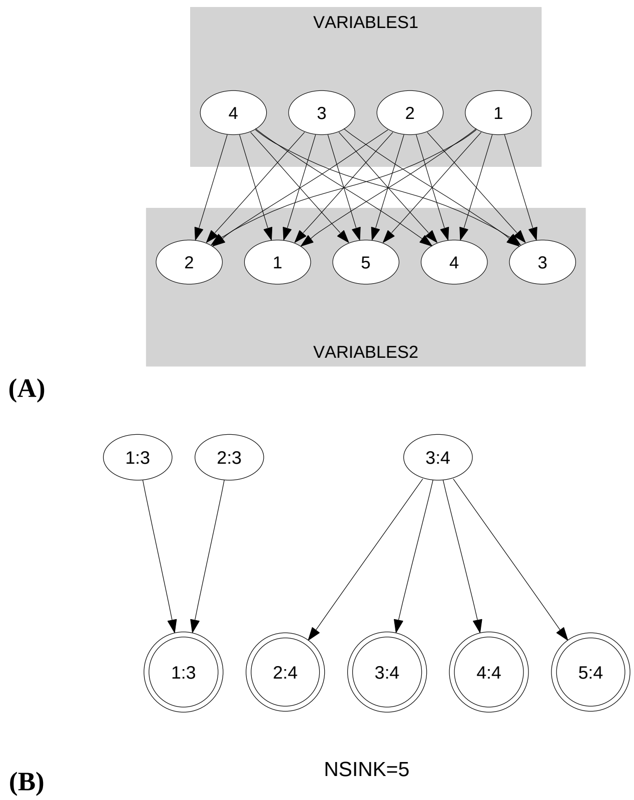



Figure 5.808: Initial and final graph of the uses constraint

## 5.418 valley

DESCRIPTION LINKS AUTOMATON

**Origin**

Derived from inflexion.

**Constraint**

valley(N, VARIABLES)

**Arguments**

N : dvar
VARIABLES : collection(var−dvar)

**Restrictions**

$\texttt{N} \geq 0$
$2 * \texttt{N} \leq \texttt{max}(|\texttt{VARIABLES}| - 1, 0)$
required(VARIABLES, var)

**Purpose**

A variable $V_v$ $(1 < v < m)$ of the sequence of variables $\texttt{VARIABLES} = V_1, \ldots, V_m$ is a *valley* if and only if there exists an $i$ (with $1 < i \leq v$) such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_v$ and $V_v < V_{v+1}$. N is the total number of valleys of the sequence of variables VARIABLES.

**Example**

$(1, \langle 1, 1, 4, 8, 8, 2, 7, 1 \rangle)$
$(0, \langle 1, 1, 4, 5, 8, 8, 4, 1 \rangle)$
$(4, \langle 1, 0, 4, 0, 8, 2, 4, 1, 2 \rangle)$

The first valley constraint holds since the sequence 1 1 4 8 8 2 7 1 contains one valley that corresponds to the variable that is assigned to value 2.

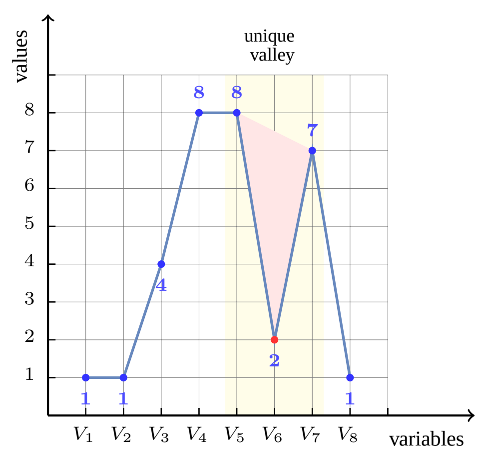



Figure 5.809: Illustration of the first example of the **Example** slot: a sequence of eight variables $V_1$, $V_2$, $V_3$, $V_4$, $V_5$, $V_6$, $V_7$, $V_8$ respectively fixed to values 1, 1, 4, 8, 8, 2, 7, 1 and its corresponding unique valley ($\texttt{N} = 1$)

**All solutions**

Figure 5.810 gives all solutions to the following non ground instance of the `valley` constraint: $\texttt{N} \in [1, 2]$, $\texttt{V}_1 \in [0, 1]$, $\texttt{V}_2 \in [0, 2]$, $\texttt{V}_3 \in [0, 2]$, $\texttt{V}_4 \in [0, 1]$, $\texttt{valley}(\texttt{N}, \langle \texttt{V}_1, \texttt{V}_2, \texttt{V}_3, \texttt{V}_4 \rangle)$.

① $(\mathbf{1}, \langle 0, 1, \mathbf{0}, 1 \rangle)$
② $(\mathbf{1}, \langle 0, 2, \mathbf{0}, 1 \rangle)$
③ $(\mathbf{1}, \langle 1, \mathbf{0}, \mathbf{0}, 1 \rangle)$
④ $(\mathbf{1}, \langle 1, \mathbf{0}, 1, 0 \rangle)$
⑤ $(\mathbf{1}, \langle 1, \mathbf{0}, 1, 1 \rangle)$
⑥ $(\mathbf{1}, \langle 1, \mathbf{0}, 2, 0 \rangle)$
⑦ $(\mathbf{1}, \langle 1, \mathbf{0}, 2, 1 \rangle)$
⑧ $(\mathbf{1}, \langle 1, 1, \mathbf{0}, 1 \rangle)$
⑨ $(\mathbf{1}, \langle 1, 2, \mathbf{0}, 1 \rangle)$

Figure 5.810: All solutions corresponding to the non ground example of the `valley` constraint of the **All solutions** slot where each valley is coloured in orange

**Typical**

$|\texttt{VARIABLES}| > 2$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$

**Symmetries**

- Items of `VARIABLES` can be reversed.
- One and the same constant can be added to the `var` attribute of all items of `VARIABLES`.

**Arg. properties**

- Functional dependency: `N` determined by `VARIABLES`.
- Contractible wrt. `VARIABLES` when $\texttt{N} = 0$.

**Usage**

Useful for constraining the number of *valleys* of a sequence of domain variables.

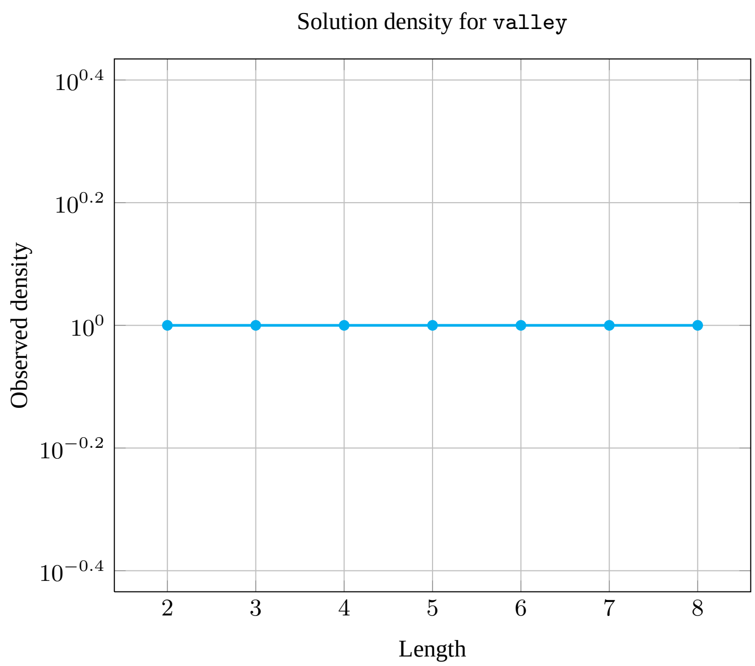
**Remark**

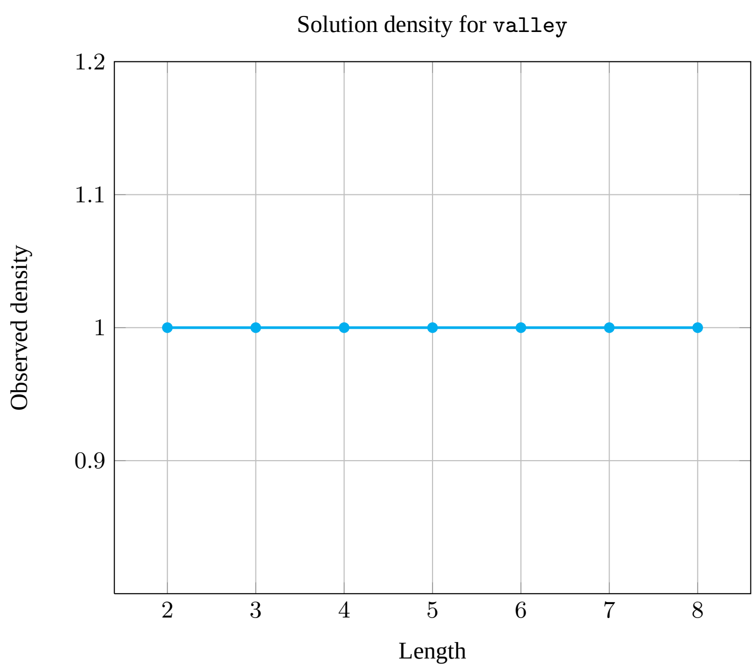
Since the arity of the arc constraint is not fixed, the `valley` constraint cannot be currently described with the graph-based representation. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 9 | 64 | 625 | 7776 | 117649 | 2097152 | 43046721 |

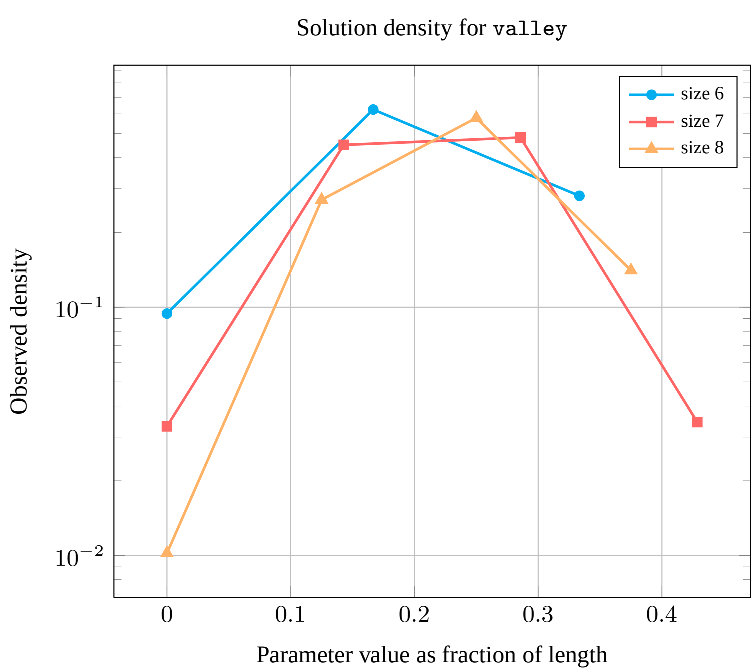
Number of solutions for `valley`: domains $0..n$


Solution density for valley
Observed density
$10^{0.4}$
$10^{0.2}$
$10^{0}$
$10^{-0.2}$
$10^{-0.4}$
2 3 4 5 6 7 8
Length

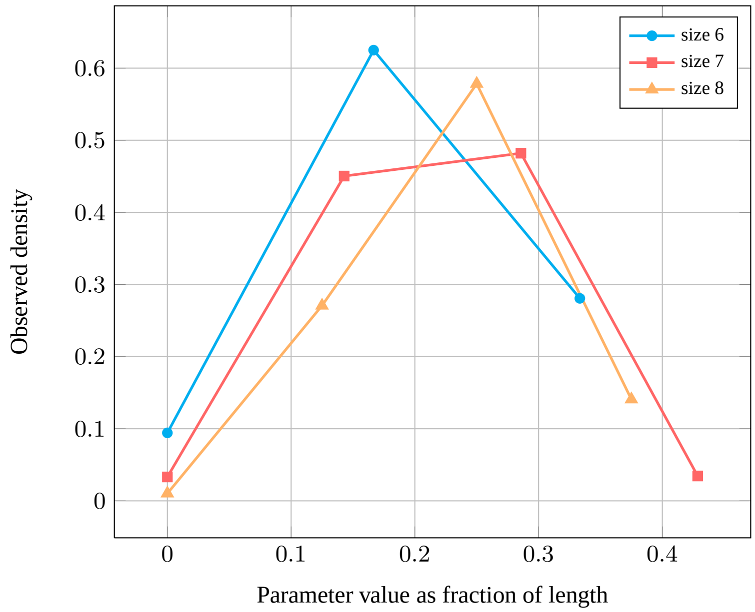

Solution density for valley
Observed density
1.2
1.1
1
0.9
2 3 4 5 6 7 8
Length

| Length ($n$) | | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Total | | 9 | 64 | 625 | 7776 | 117649 | 2097152 | 43046721 |
| Parameter value | 0 | 9 | 50 | 295 | 1792 | 11088 | 69498 | 439791 |
| | 1 | - | 14 | 330 | 5313 | 73528 | 944430 | 11654622 |
| | 2 | - | - | - | 671 | 33033 | 1010922 | 24895038 |
| | 3 | - | - | - | - | - | 72302 | 6057270 |

Solution count for `valley`: domains $0..n$



Solution density for `valley`




**See also**

**common keyword:** `deepest_valley`, `inflexion`, `min_dist_between_inflexion`, `min_width_valley` *(sequence)*.

**comparison swapped:** `peak`.

**generalisation:** `big_valley` *(a tolerance parameter is added for counting only big valleys)*.

**related:** `all_equal_valley`, `all_equal_valley_min`, `decreasing_valley`, `increasing_valley`, `no_peak`.

**specialisation:** `no_valley` *(the variable counting the number of valleys is set to* $0$ *and removed)*.

**Keywords**

**characteristic of a constraint:** automaton, automaton with counters, automaton with same input symbol.

**combinatorial object:** sequence.

**constraint arguments:** reverse of a constraint, pure functional dependency.

**constraint network structure:** sliding cyclic(1) constraint network(2).

**filtering:** glue matrix.

**modelling:** functional dependency.

**Cond. implications**

- `valley`(`N`, `VARIABLES`)
  with $\texttt{N} > 0$
  **implies** `atleast_nvalue`(`NVAL`, `VARIABLES`)
  when $\texttt{NVAL} = 2$.

- valley(N, VARIABLES)
  **implies** inflexion(N, VARIABLES)
  when N = peak(VARIABLES.var) + valley(VARIABLES.var).

**Automaton** Figure 5.811 depicts the automaton associated with the `valley` constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection `VARIABLES` corresponds a signature variable $S_i$. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and $S_i$: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATES SEMANTICS

| | | |
|---|---|---|
| $s$ | : stationary/increasing mode | $(\{<\mid=\}^*)$ |
| $u$ | : decreasing mode | $(>\{>\mid=\}^*)$ |

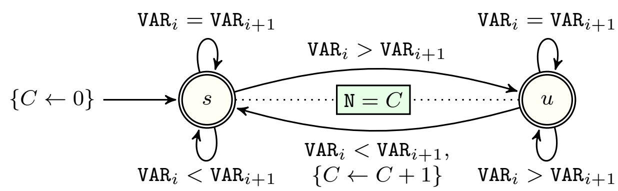



Figure 5.811: Automaton of the `valley` constraint

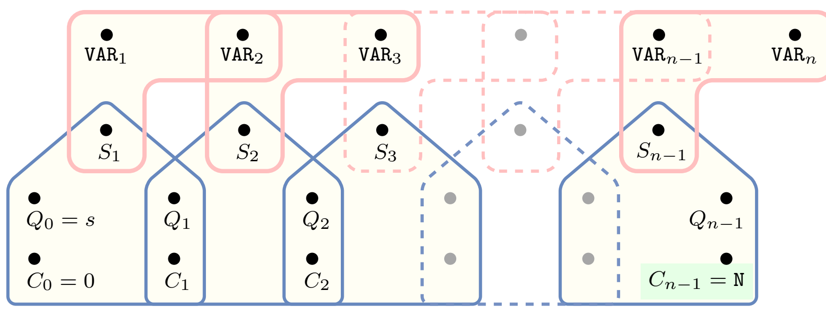



Figure 5.812: Hypergraph of the reformulation corresponding to the automaton of the `valley` constraint (since all states of the automaton are accepting there is no restriction on the last variable $Q_{n-1}$)

Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value $C$ at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence `VARIABLES`.

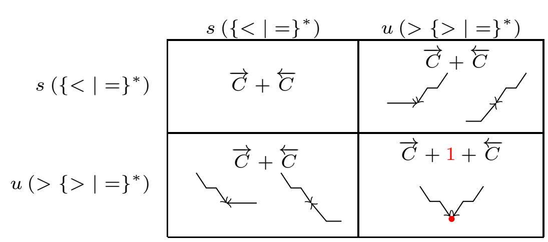



Figure 5.813: Glue matrix of the `valley` constraint

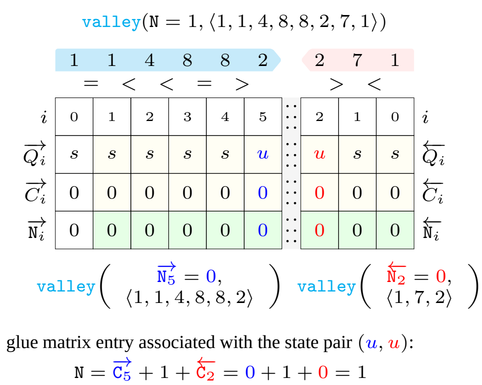



Figure 5.814: Illustrating the use of the state pair $(u, u)$ of the glue matrix for linking $\texttt{N}$ with the counters variables obtained after reading the prefix $1, 1, 4, 8, 8, 2$ and corresponding suffix $2, 7, 1$ of the sequence $1, 1, 4, 8, 8, 2, 7, 1$; note that the suffix $2, 7, 1$ (in pink) is proceed in reverse order; the left (resp. right) table shows the initialisation (for $i = 0$) and the evolution (for $i > 0$) of the state of the automaton and its counter $C$ upon reading the prefix $1, 1, 4, 8, 8, 2$ (resp. the reverse suffix $1, 7, 2$).

## 5.419 vec_eq_tuple

DESCRIPTION LINKS GRAPH

**Origin** Used for defining in_relation.

**Constraint** vec_eq_tuple(VARIABLES, TUPLE)

**Arguments**

VARIABLES : collection(var−dvar)
TUPLE : collection(val−int)

**Restrictions**

required(VARIABLES, var)
required(TUPLE, val)
$|$VARIABLES$|$ = $|$TUPLE$|$

**Purpose** Enforce a vector of domain variables to be equal to a tuple of values.

**Example** $(\langle 5, 3, 3\rangle, \langle 5, 3, 3\rangle)$

The vec_eq_tuple constraint holds since the first, the second and the third items of VARIABLES $= \langle 5, 3, 3\rangle$ are respectively equal to the first, the second and the third items of TUPLE $= \langle 5, 3, 3\rangle$.

**Typical**

$|$VARIABLES$|$ $> 1$
range(VARIABLES.var) $> 1$
range(TUPLE.val) $> 1$

**Symmetries**

- Arguments are permutable w.r.t. permutation (VARIABLES, TUPLE).
- Items of VARIABLES and TUPLE are permutable (*same permutation used*).

**Arg. properties** Contractible wrt. VARIABLES and TUPLE (*remove items from same position*).

**Used in** in_relation.

**See also** **generalisation:** lex_equal (integer *replaced by* variable *in second argument)*.

**implies:** lex_equal.

**Keywords** **characteristic of a constraint:** tuple.

**constraint type:** value constraint.

**filtering:** arc-consistency.

| | |
|---|---|
| **Arc input(s)** | VARIABLES TUPLE |
| **Arc generator** | $PRODUCT(=) \mapsto$ collection(variables, tuple) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | variables.var = tuple.val |
| **Graph property(ies)** | **NARC** = \|VARIABLES\| |

**Graph model** Parts (A) and (B) of Figure 5.815 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

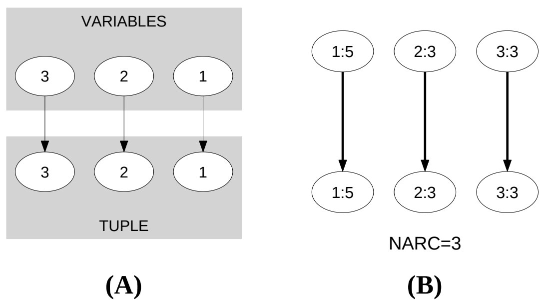



Figure 5.815: Initial and final graph of the vec_eq_tuple constraint

**Signature** Since we use the arc generator $PRODUCT(=)$ on the collections VARIABLES and TUPLE, and because of the restriction |VARIABLES| = |TUPLE|, the maximum number of arcs of the final graph is equal to |VARIABLES|. Therefore we can rewrite the graph property **NARC** = |VARIABLES| to **NARC** $\geq$ |VARIABLES| and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

## 5.420 visible

DESCRIPTION LINKS

**Origin** Extension of *accessibility* parameter of diffn.

**Constraint** visible(K, DIMS, FROM, OBJECTS, SBOXES)

**Types**

```
VARIABLES : collection(v-dvar)
INTEGERS  : collection(v-int)
POSITIVES : collection(v-int)
DIMDIR    : collection(dim-int, dir-int)
```

**Arguments**

```
K       : int
DIMS    : sint
FROM    : DIMDIR
OBJECTS : collection(oid-int,
                     sid-dvar,
                     x-VARIABLES,
                     start-dvar,
                     duration-dvar,
                     end-dvar)
SBOXES  : collection(sid-int,
                     t-INTEGERS,
                     l-POSITIVES,
                     f-DIMDIR)
```

**Restrictions**

$|\mathtt{VARIABLES}| \geq 1$
$|\mathtt{INTEGERS}| \geq 1$
$|\mathtt{POSITIVES}| \geq 1$
$\mathtt{required}(\mathtt{VARIABLES}, \mathtt{v})$
$|\mathtt{VARIABLES}| = \mathtt{K}$
$\mathtt{required}(\mathtt{INTEGERS}, \mathtt{v})$
$|\mathtt{INTEGERS}| = \mathtt{K}$
$\mathtt{required}(\mathtt{POSITIVES}, \mathtt{v})$
$|\mathtt{POSITIVES}| = \mathtt{K}$
$\mathtt{POSITIVES.v} > 0$
$\mathtt{required}(\mathtt{DIMDIR}, [\mathtt{dim}, \mathtt{dir}])$
$|\mathtt{DIMDIR}| > 0$
$|\mathtt{DIMDIR}| \leq \mathtt{K} + \mathtt{K}$
$\mathtt{distinct}(\mathtt{DIMDIR}, [])$
$\mathtt{DIMDIR.dim} \geq 0$
$\mathtt{DIMDIR.dim} < \mathtt{K}$
$\mathtt{DIMDIR.dir} \geq 0$
$\mathtt{DIMDIR.dir} \leq 1$
$\mathtt{K} \geq 0$
$\mathtt{DIMS} \geq 0$
$\mathtt{DIMS} < \mathtt{K}$
$\mathtt{distinct}(\mathtt{OBJECTS}, \mathtt{oid})$
$\mathtt{required}(\mathtt{OBJECTS}, [\mathtt{oid}, \mathtt{sid}, \mathtt{x}])$
$\mathtt{require\_at\_least}(2, \mathtt{OBJECTS}, [\mathtt{start}, \mathtt{duration}, \mathtt{end}])$
$\mathtt{OBJECTS.oid} \geq 1$
$\mathtt{OBJECTS.oid} \leq |\mathtt{OBJECTS}|$
$\mathtt{OBJECTS.sid} \geq 1$
$\mathtt{OBJECTS.sid} \leq |\mathtt{SBOXES}|$
$\mathtt{OBJECTS.duration} \geq 0$
$|\mathtt{SBOXES}| \geq 1$
$\mathtt{required}(\mathtt{SBOXES}, [\mathtt{sid}, \mathtt{t}, \mathtt{l}])$
$\mathtt{SBOXES.sid} \geq 1$
$\mathtt{SBOXES.sid} \leq |\mathtt{SBOXES}|$
$\mathtt{do\_not\_overlap}(\mathtt{SBOXES})$

**Purpose**

Holds if and only if:

1. The difference between the end in time and the start in time of each object is equal to its duration in time.
2. Given a collection of potential observations places FROM, where each observation place is specified by a *dimension* (i.e., an integer between $0$ and $k-1$) and by a *direction* (i.e., an integer between $0$ and $1$), and given for each shifted box of SBOXES a set of visible faces, enforce that *at least one visible face of each shifted box associated with an object* $o \in$ OBJECTS *should be entirely visible from at least one observation place of* FROM *at time* $o$.start *as well as at time* $o$.end $-1$. This notion is defined in a more formal way in the **Remark** slot.

**Example**

$$\left(\begin{array}{l}
2,\{0,1\},\\
\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\left\langle\begin{array}{llllll}
\texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,2\rangle & \texttt{start}-8 & \texttt{duration}-8 & \texttt{end}-16,\\
\texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 4,2\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16
\end{array}\right\rangle,\\
\left\langle\begin{array}{llll}
\texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle
\end{array}\right\rangle
\end{array}\right)$$

$$\left(\begin{array}{l}
2,\{0,1\},\\
\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\left\langle\begin{array}{llllll}
\texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,2\rangle & \texttt{start}-1 & \texttt{duration}-8 & \texttt{end}-9,\\
\texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 4,2\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16
\end{array}\right\rangle,\\
\left\langle\begin{array}{llll}
\texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle
\end{array}\right\rangle
\end{array}\right)$$

$$\left(\begin{array}{l}
2,\{0,1\},\\
\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\left\langle\begin{array}{llllll}
\texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,1\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16,\\
\texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 2,2\rangle & \texttt{start}-6 & \texttt{duration}-6 & \texttt{end}-12
\end{array}\right\rangle,\\
\left\langle\begin{array}{llll}
\texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle
\end{array}\right\rangle
\end{array}\right)$$

$$\left(\begin{array}{l}
2,\{0,1\},\\
\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\left\langle\begin{array}{llllll}
\texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 4,1\rangle & \texttt{start}-1 & \texttt{duration}-8 & \texttt{end}-9,\\
\texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 1,2\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16
\end{array}\right\rangle,\\
\left\langle\begin{array}{llll}
\texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle
\end{array}\right\rangle
\end{array}\right)$$

$$\left(\begin{array}{l}
2,\{0\},\\
\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\left\langle\begin{array}{llllll}
\texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 2,1\rangle & \texttt{start}-1 & \texttt{duration}-8 & \texttt{end}-9,\\
\texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 4,3\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16
\end{array}\right\rangle,\\
\left\langle\begin{array}{llll}
\texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\
\texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle
\end{array}\right\rangle
\end{array}\right)$$

The five previous examples correspond respectively to parts (I), (II) of Figure 5.817, to parts (III) and (IV) of Figure 5.818, and to Figure 5.819. Before introducing these five examples Figure 5.816 first illustrates the notion of *observations places* and of *visible faces*.

We first need to introduce a number of definitions in order to illustrate the notion of *visibility*.

**Definition 1.** *Consider two distinct objects $o$ and $o'$ of the* `visible` *constraint (i.e., $o, o' \in iobjects$) as well as an observation place defined by the pair $\langle \texttt{dim}, \texttt{dir}\rangle \in$* `FROM`*. The object $o$* is masked by *the object $o'$ according to the observation place $\langle \texttt{dim}, \texttt{dir}\rangle$ if there exist two shifted boxes $s$ and $s'$ respectively associated with $o$ and $o'$ such that conditions* **A**, **B**, **C**, **D** *and* **E** *all hold:*

- *(**A**) $o.\texttt{duration} > 0 \wedge o'.\texttt{duration} > 0 \wedge o.\texttt{end} > o'.\texttt{start} \wedge o'.\texttt{end} > o.\texttt{start}$ (i.e., the time intervals associated with $o$ and $o'$ intersect).*
- *(**B**) Discarding dimension* `dim`*, $s$ and $s'$ intersect in all dimensions specified by* `DIMS` *(i.e., objects $o$ and $o'$ are in vis-à-vis).*
- *(**C**) If $\texttt{dir} = 0$*

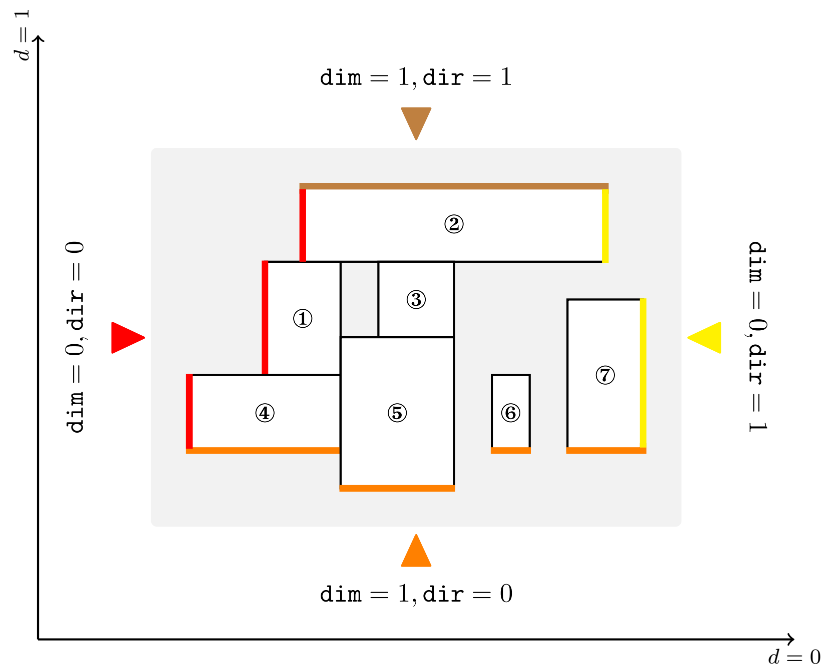



Figure 5.816: Entirely visible faces (depicted by a thick line) of rectangles ①, ②, ③, ④, ⑤, ⑥ and ⑦ from the four observation places $\langle \mathtt{dim} = 0, \mathtt{dir} = 1\rangle$, $\langle \mathtt{dim} = 0, \mathtt{dir} = 0\rangle$, $\langle \mathtt{dim} = 1, \mathtt{dir} = 1\rangle$ and $\langle \mathtt{dim} = 1, \mathtt{dir} = 0\rangle$ (depicted by a small triangle)

*then* $o.x[\mathtt{dim}] + s.t[\mathtt{dim}] \geq o'.x[\mathtt{dim}] + s'.t[\mathtt{dim}] + s'.l[\mathtt{dim}]$
*else* $o'.x[\mathtt{dim}] + s'.t[\mathtt{dim}] \geq o.x[\mathtt{dim}] + s.t[\mathtt{dim}] + s.l[\mathtt{dim}]$ *(i.e., in dimension* $\mathtt{dim}$*, $o$ and $o'$ are ordered in the wrong way according to direction* $\mathtt{dir}$*).*

- **(D)** $o.\mathtt{start} > o'.\mathtt{start} \vee o.\mathtt{end} < o'.\mathtt{end}$ *(i.e., instants $o.\mathtt{start}$ or $o.\mathtt{end}$ are located within interval $[o'.\mathtt{start}, o'.\mathtt{end}]$; we consider also condition* **A.***).*
- **(E)** *The observation place $\langle \mathtt{dim}, \mathtt{dir}\rangle$ occurs within the list of visible faces associated with the face attribute* $\mathtt{f}$ *of the shifted box $s$ (i.e., the pair $\langle \mathtt{dim}, \mathtt{dir}\rangle$ is a potentially visible face of $o$).*

**Definition 2.** *Consider an object $o$ of the collection* $\mathtt{OBJECTS}$ *as well as a possible observation place defined by the pair $\langle \mathtt{dim}, \mathtt{dir}\rangle$. The object $o$* is masked according to the observation place $\langle \mathtt{dim}, \mathtt{dir}\rangle$ *if and only if at least one of the following conditions holds:*

- *No shifted box associated with $o$ has the pair $\langle \mathtt{dim}, \mathtt{dir}\rangle$ as one of its potentially visible face.*
- *The object $o$ is masked according to the possible observation place $\langle \mathtt{dim}, \mathtt{dir}\rangle$ by another object $o'$.*

Figures 5.817, 5.818, and 5.819 respectively illustrate Definition 1 in the context of an observation place (depicted by a triangle) that is equal to the pair $\langle \mathtt{dim} = 0, \mathtt{dir} = 1\rangle$. Note that, in the context of Figure 5.819, as the `DIMS` parameter of the `visible` constraint only mentions dimension $0$ (and not dimension $1$), one object may be masked by another object even though the two objects do not intersect in any dimension: i.e., only their respective ordering in the dimension $\mathtt{dim} = 0$ as well as their positions in time matter.

**Definition 3.** *Consider an object $o$ of the collection* `OBJECTS` *as well as a possible observation place defined by the pair $\langle \mathtt{dim}, \mathtt{dir} \rangle$. The object $o$* is masked according to the observation place $\langle \mathtt{dim}, \mathtt{dir} \rangle$ *if and only if at least one of the following conditions holds:*

- *No shifted box associated with $o$ has the pair $\langle \mathtt{dim}, \mathtt{dir} \rangle$ as one of its potentially visible face.*
- *The object $o$ is masked according to the possible observation place $\langle \mathtt{dim}, \mathtt{dir} \rangle$ by another object $o'$.*

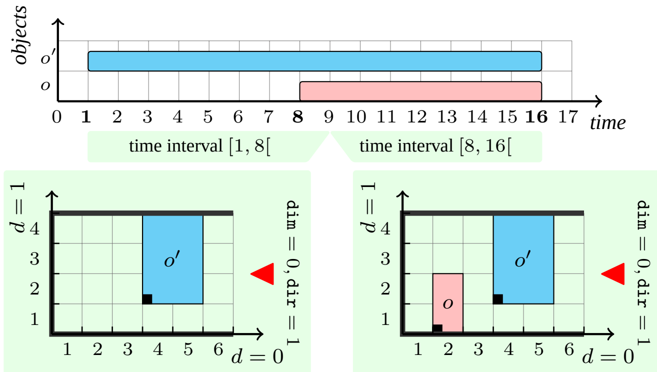
**Definition 4.** *An object of the collection* `OBJECTS` *constraint is* masked according to a set of possible observation places `FROM` *if it is masked according to each observation place of* `FROM`.

We are now in position to define the `visible` constraint.

**Definition 5.** *Given a* $\mathtt{visible}(\mathtt{K}, \mathtt{DIMS}, \mathtt{FROM}, \mathtt{OBJECTS}, \mathtt{SBOXES})$ *constraint, the* `visible` *constraint holds if none of the objects of* `OBJECTS` *is masked according to the dimensions of* `DIMS` *and to the set of possible observation places defined by* `FROM`*.*

$$\texttt{visible}\left(\begin{array}{c} 2,\ \{0,1\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 1,2\rangle & \texttt{start}-8 & \texttt{duration}-8 & \texttt{end}-16, \\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 4,2\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

(I)

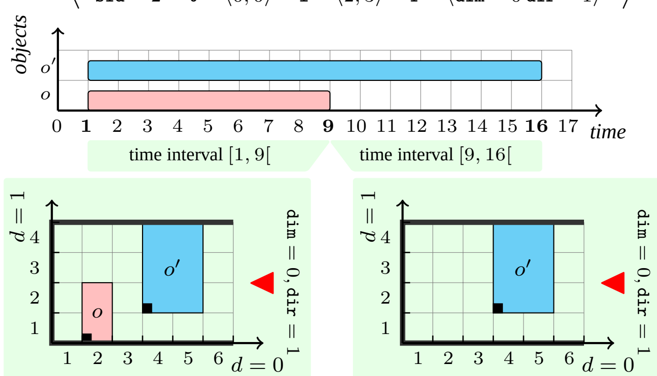



$o$ is masked by $o'$ according to $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ since:
**(A)** $o$ and $o'$ intersect in time,
**(B)** $o$ and $o'$ intersect in dimension 1,
**(C)** in dimension 0, $o'$ starts after the end of $o$,
**(D)** the start in time of $o$ is located after the start in time of $o'$,
**(E)** $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ is a potentially visible face of $o$.

---

$$\texttt{visible}\left(\begin{array}{c} 2,\ \{0,1\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 1,2\rangle & \texttt{start}-1 & \texttt{duration}-8 & \texttt{end}-9, \\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 4,2\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

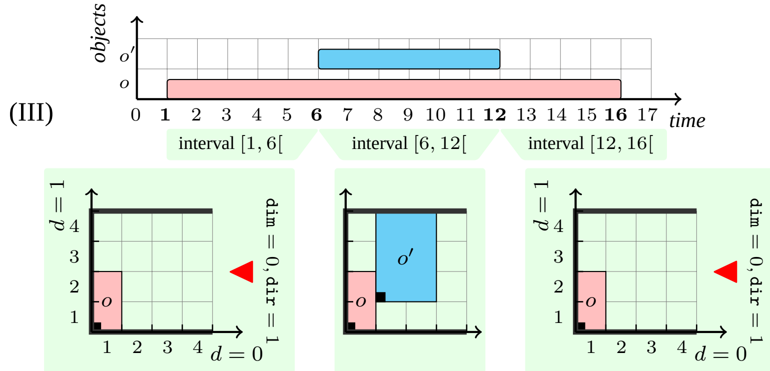
(II)




$o$ is masked by $o'$ according to $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ since:
**(A)** $o$ and $o'$ intersect in time,
**(B)** $o$ and $o'$ intersect in dimension 1,
**(C)** in dimension 0, $o'$ starts after the end of $o$,
**(D)** the end in time of $o$ is located before the end in time of $o'$,
**(E)** $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ is a potentially visible face of $o$.

Figure 5.817: Illustration of Definition 1: two examples (I) and (II) where an object $o$ is masked by an object $o'$ according to dimensions $\{0,1\}$ and to the observation place $\langle \texttt{dim} = 0, \texttt{dir} = 1\rangle$ because **(A)** $o$ and $o'$ intersect in time, **(B)** $o$ and $o'$ intersect in dimension 1, **(C)** $o$ and $o'$ are not well ordered according to the observation place, **(D)** there exists an instant where $o'$ if present (but not $o$) and **(E)** $\langle \texttt{dim} = 0, \texttt{dir} = 1\rangle$ is a potentially visible face of $o$.

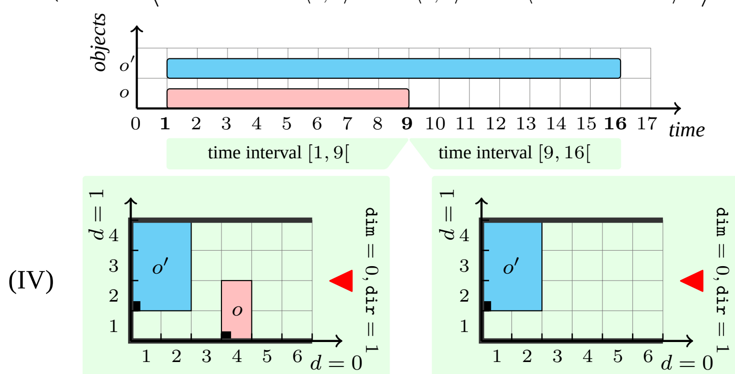
$$\texttt{visible}\left(\begin{array}{c} 2,\ \{0,1\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 1,1\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16, \\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 2,2\rangle & \texttt{start}-6 & \texttt{duration}-6 & \texttt{end}-12 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$




$o$ is not masked by $o'$ according to $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ since:
**(A)** Even though $o$ and $o'$ intersect in time,
**(B)** and even though $o$ and $o'$ intersect in dimension 1,
**(C)** and even though, in dimension 0, $o'$ starts after the end of $o$,
**(E)** and even though $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ is a potentially visible face of $o$,
condition **(D)** does not hold.

---

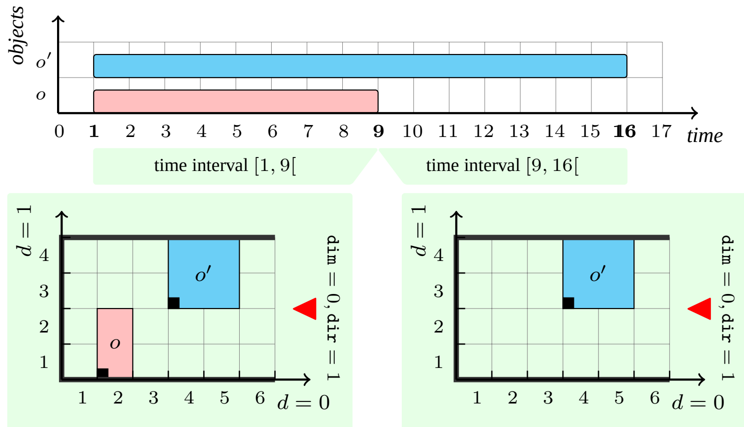
$$\texttt{visible}\left(\begin{array}{c} 2,\ \{0,1\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 4,1\rangle & \texttt{start}-1 & \texttt{duration}-8 & \texttt{end}-9, \\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 1,2\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$




$o$ is not masked by $o'$ according to $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ since:
**(A)** Even though $o$ and $o'$ intersect in time,
**(B)** and even though $o$ and $o'$ intersect in dimension 1, and even
**(D)** though the end in time of $o$ is located before the end in time of $o'$,
**(E)** and even though $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ is a potentially visible face of $o$,
condition **(C)** does not hold.

Figure 5.818: Illustration of Definition 1: two examples (III) and (IV) where an object $o$ is not masked by an object $o'$ according to the observation place $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$.

$$\texttt{visible}\left(\begin{array}{c} 2,\ \{0\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 2,1\rangle & \texttt{start}-1 & \texttt{duration}-8 & \texttt{end}-9, \\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 4,3\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$




$o$ is masked by $o'$ according to $\langle \texttt{dim}=0,\texttt{dir}=1\rangle$ since:

**A.** $o$ and $o'$ intersect in time,

**B.** in dimension $0$, $o'$ starts after the end of $o$,

**C.** the end in time of $o$ is located before the end in time of $o'$,

**D.** $\langle \texttt{dim}=0,\texttt{dir}=1\rangle$ is a potentially visible face of $o$.

Figure 5.819: Illustration of Definition 1: the case where an object $o$ is masked by an object $o'$ according to dimension $0$ and to the observation place $\langle \texttt{dim}=0,\texttt{dir}=1\rangle$ because: **(A)** $o$ and $o'$ intersect in time, **(C)** $o$ and $o'$ are not well ordered according to the observation place and **(D)** there exists an instant where $o'$ if present (but not $o$) and **(E)** $\langle \texttt{dim}=0,\texttt{dir}=1\rangle$ is a potentially visible face of $o$.

**Typical** $|\texttt{OBJECTS}| > 1$

**Symmetries**

- Items of `OBJECTS` are permutable.
- Items of `SBOXES` are permutable.

**Usage** We now give several typical concrete uses of the `visible` constraint, which all mention the `diffst` as well as the `visible` constraints:

- Figure 5.820 corresponds to a *ship loading problem* where containers are piled within a ship by a crane each time the ship visits a given harbour. In this context we have first to express the fact that *a container can only be placed on top of an already placed container* and second, that *a container can only be taken away if no container is placed on top of it*. These two conditions are expressed by a single `visible` constraint for which the `DIMS` parameter mentions all three dimensions of the placement space and the `FROM` parameter mentions the pair $\langle \texttt{dim} = 2, \texttt{dir} = 1\rangle$ as its unique observation place. In addition we also use a `diffst` constraint for expressing non-overlapping.

$$\texttt{visible}\left(\begin{array}{c} 3,\ \{0,1,2\},\ \langle \texttt{dim}-2\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-\mathbf{1} & \texttt{sid}-1 & \texttt{x}-\langle 1,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17, \\ \texttt{oid}-\mathbf{2} & \texttt{sid}-1 & \texttt{x}-\langle 1,1,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8, \\ \texttt{oid}-\mathbf{3} & \texttt{sid}-1 & \texttt{x}-\langle 4,1,1\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8, \\ \texttt{oid}-\mathbf{4} & \texttt{sid}-1 & \texttt{x}-\langle 1,1,3\rangle & \texttt{start}-8 & \texttt{duration}-9 & \texttt{end}-17, \\ \texttt{oid}-\mathbf{5} & \texttt{sid}-1 & \texttt{x}-\langle 4,1,1\rangle & \texttt{start}-8 & \texttt{duration}-16 & \texttt{end}-24, \\ \texttt{oid}-\mathbf{6} & \texttt{sid}-1 & \texttt{x}-\langle 1,1,1\rangle & \texttt{start}-17 & \texttt{duration}-7 & \texttt{end}-24 \end{array} \right\rangle, \\ \langle\ \texttt{sid}-1\quad \texttt{t}-\langle 0,0,0\rangle\quad \texttt{l}-\langle 2,4,2\rangle\quad \texttt{f}-\langle \texttt{dim}-2\ \texttt{dir}-1\rangle\ \rangle \end{array}\right)$$

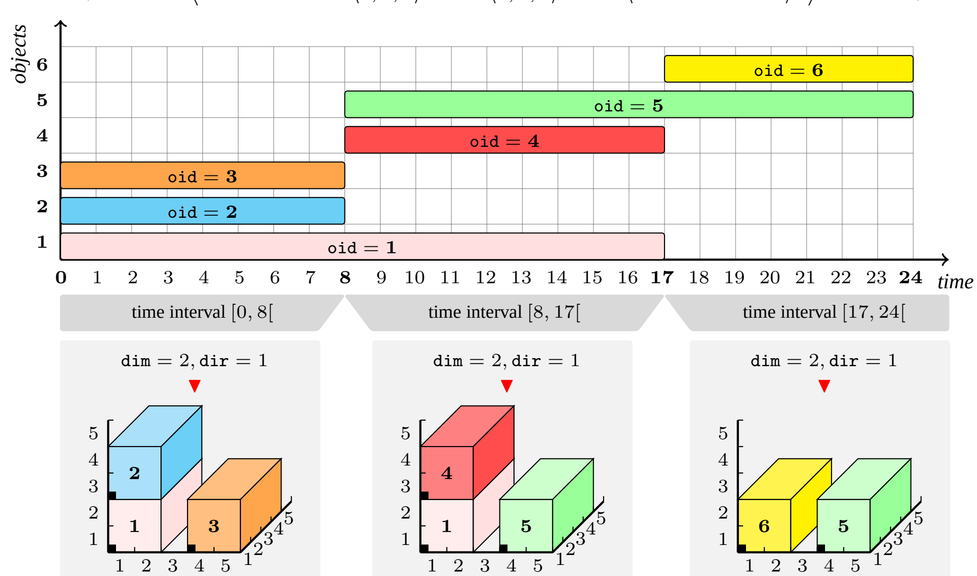



Figure 5.820: Illustration of the ship loading problem

- Figure 5.821 corresponds to a *container loading/unloading problem* in the context of a pick-up delivery problem where the loading/unloading takes place with respect to the front door of the container. Beside the `diffst` constraint used for expressing non-overlapping, we use two distinct `visible` constraints:
  - The first `visible` constraint takes care of the location of the front door of the container (each object $o$ has to be loaded/unloaded without moving around any other object, i.e., objects that are in the vis-à-vis of $o$ according to the front door of the container). This is expressed by a single `visible` constraint for which the `DIMS` parameter mentions all three dimensions of the placement space and the `FROM` parameter mentions the pair $\langle \texttt{dim} = 1, \texttt{dir} = 0\rangle$ as its unique observation place.
  - The second `visible` constraint takes care of the *gravity dimension* (i.e., each object that has to be loaded should not be put under another object, and reciprocally each object that has to be unloaded should not be located under another object). This is expressed by the same `visible` constraint that was used for the ship loading problem, i.e., a `visible` constraint for which the `DIMS` parameter mentions all three dimensions of the placement space and the `FROM` parameter mentions the pair $\langle \texttt{dim} = 2, \texttt{dir} = 1\rangle$ as its unique observation place.
- Figure 5.822 corresponds to a *pallet loading problem* where one has to place six objects on a pallet. Each object corresponds to a parallelepiped that has a bar code on one of its four sides (i.e., the sides that are different from the top and the bottom of the parallelepiped). If, for some reason, an object has no bar code then we simply remove it from the objects that will be passed to the `visible` constraint: this is for instance the case for the sixth object. In this context the constraint to enforce (beside the non-overlapping constraint between the parallelepipeds that are assigned to a same pallet) is the fact that the bar code of each object should be visible (i.e., visible from one of the four sides of the pallet). This is expressed by the `visible` constraint given in Part (F) of Figure 5.822.

**Remark** The `visible` constraint is a generalisation of the `accessibility` constraint initially introduced in the context of the `diffn` constraint.

**See also** **common keyword:** `diffn` *(geometrical constraint)*,
`geost`, `geost_time` *(geometrical constraint,sweep)*,
`non_overlap_sboxes` *(geometrical constraint)*.

**Keywords** **constraint type:** decomposition, predefined constraint.

**filtering:** sweep.

**geometry:** geometrical constraint.

$$\texttt{visible}\left(\begin{array}{c}
3,\ \{0,1,2\},\ \langle\texttt{dim}-1\ \texttt{dir}-0\rangle,\\
\left\langle\begin{array}{llllll}
\texttt{oid}-\mathbf{1} & \texttt{sid}-1 & \texttt{x}-\langle1,2,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8,\\
\texttt{oid}-\mathbf{2} & \texttt{sid}-2 & \texttt{x}-\langle1,3,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8,\\
\texttt{oid}-\mathbf{3} & \texttt{sid}-3 & \texttt{x}-\langle1,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17,\\
\texttt{oid}-\mathbf{4} & \texttt{sid}-4 & \texttt{x}-\langle4,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17,\\
\texttt{oid}-\mathbf{5} & \texttt{sid}-5 & \texttt{x}-\langle1,2,3\rangle & \texttt{start}-8 & \texttt{duration}-9 & \texttt{end}-17,\\
\texttt{oid}-\mathbf{6} & \texttt{sid}-6 & \texttt{x}-\langle3,1,1\rangle & \texttt{start}-8 & \texttt{duration}-12 & \texttt{end}-24,\\
\texttt{oid}-\mathbf{7} & \texttt{sid}-3 & \texttt{x}-\langle1,1,1\rangle & \texttt{start}-17 & \texttt{duration}-7 & \texttt{end}-24
\end{array}\right\rangle,\\
\left\langle\begin{array}{llll}
\texttt{sid}-1 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,1,1\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-2 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,2,2\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-3 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,4,2\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-4 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,4,1\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-5 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,3,1\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-6 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle1,2,2\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle
\end{array}\right\rangle
\end{array}\right)$$

$$\texttt{visible}\left(\begin{array}{c}
3,\ \{0,1,2\},\ \langle\texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\left\langle\begin{array}{llllll}
\texttt{oid}-\mathbf{1} & \texttt{sid}-1 & \texttt{x}-\langle1,2,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8,\\
\texttt{oid}-\mathbf{2} & \texttt{sid}-2 & \texttt{x}-\langle1,3,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8,\\
\texttt{oid}-\mathbf{3} & \texttt{sid}-3 & \texttt{x}-\langle1,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17,\\
\texttt{oid}-\mathbf{4} & \texttt{sid}-4 & \texttt{x}-\langle4,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17,\\
\texttt{oid}-\mathbf{5} & \texttt{sid}-5 & \texttt{x}-\langle1,2,3\rangle & \texttt{start}-8 & \texttt{duration}-9 & \texttt{end}-17,\\
\texttt{oid}-\mathbf{6} & \texttt{sid}-6 & \texttt{x}-\langle3,1,1\rangle & \texttt{start}-8 & \texttt{duration}-12 & \texttt{end}-24,\\
\texttt{oid}-\mathbf{7} & \texttt{sid}-3 & \texttt{x}-\langle1,1,1\rangle & \texttt{start}-17 & \texttt{duration}-7 & \texttt{end}-24
\end{array}\right\rangle,\\
\left\langle\begin{array}{llll}
\texttt{sid}-1 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,1,1\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-2 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,2,2\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-3 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,4,2\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-4 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,4,1\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-5 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle2,3,1\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\
\texttt{sid}-6 & \texttt{t}-\langle0,0,0\rangle & \texttt{l}-\langle1,2,2\rangle & \texttt{f}-\langle\texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle
\end{array}\right\rangle
\end{array}\right)$$

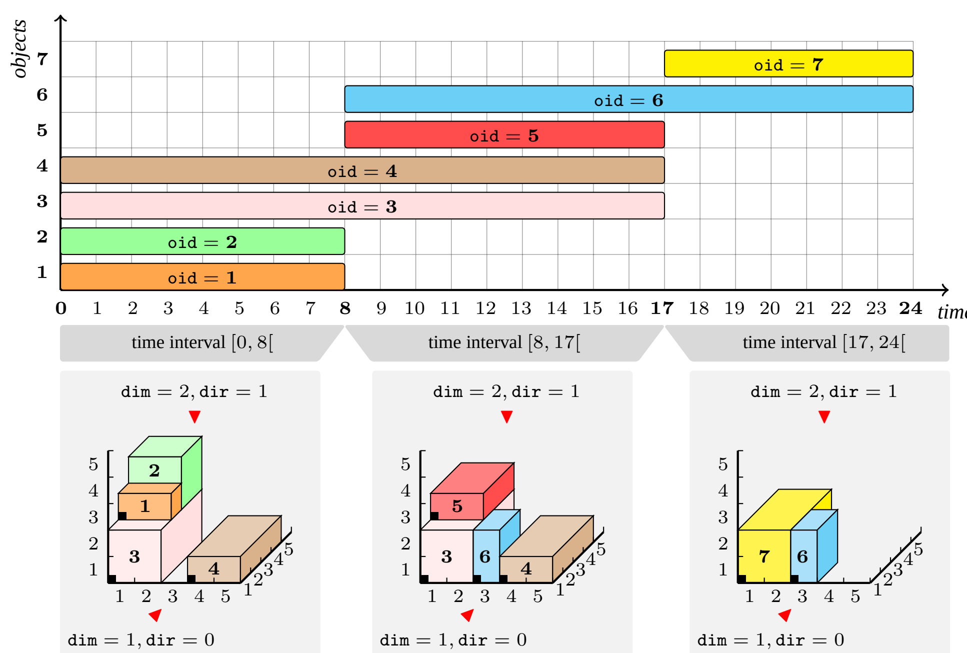
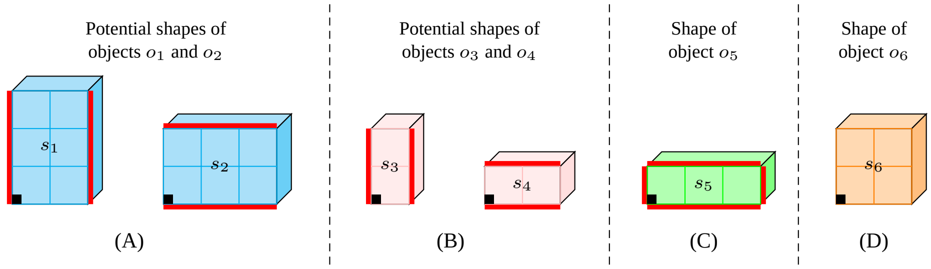
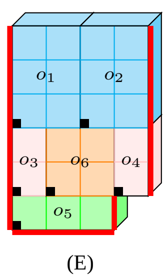



Figure 5.821: Illustration of the pick-up delivery problem




`visible`

$$\left(\begin{array}{l} 3,\ \{0,1,2\},\ \langle \mathtt{dim}-0\,\mathtt{dir}-0, \mathtt{dim}-0\,\mathtt{dir}-1, \mathtt{dim}-1\,\mathtt{dir}-0, \mathtt{dim}-1\,\mathtt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \mathtt{oid}-o_1 & \mathtt{sid}-s_1 & \mathtt{x}-\langle 1,4,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\ \mathtt{oid}-o_2 & \mathtt{sid}-s_1 & \mathtt{x}-\langle 3,4,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\ \mathtt{oid}-o_3 & \mathtt{sid}-s_2 & \mathtt{x}-\langle 1,2,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\ \mathtt{oid}-o_4 & \mathtt{sid}-s_2 & \mathtt{x}-\langle 4,1,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\ \mathtt{oid}-o_5 & \mathtt{sid}-s_3 & \mathtt{x}-\langle 1,1,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\ \mathtt{oid}-o_6 & \mathtt{sid}-s_4 & \mathtt{x}-\langle 2,2,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1 \end{array} \right\rangle, \\ \left\langle \begin{array}{llll} \mathtt{sid}-s_1 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 2,3,1\rangle & \mathtt{f}-\langle \mathtt{dim}-0\,\mathtt{dir}-0, \mathtt{dim}-0\,\mathtt{dir}-1\rangle, \\ \mathtt{sid}-s_2 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 3,2,1\rangle & \mathtt{f}-\angle \mathtt{dim}-1\,\mathtt{dir}-0, \mathtt{dim}-1\,\mathtt{dir}-1\rangle, \\ \mathtt{sid}-s_3 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 1,2,1\rangle & \mathtt{f}-\langle \mathtt{dim}-0\,\mathtt{dir}-0, \mathtt{dim}-0\,\mathtt{dir}-1\rangle, \\ \mathtt{sid}-s_4 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 2,1,1\rangle & \mathtt{f}-\langle \mathtt{dim}-1\,\mathtt{dir}-0, \mathtt{dim}-1\,\mathtt{dir}-1\rangle, \\ \mathtt{sid}-s_5 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 3,1,1\rangle & \mathtt{f}-\left\langle \begin{array}{l} \mathtt{dim}-0\,\mathtt{dir}-0, \mathtt{dim}-0\,\mathtt{dir}-1, \\ \mathtt{dim}-1\,\mathtt{dir}-0, \mathtt{dim}-1\,\mathtt{dir}-1 \end{array} \right\rangle, \\ \mathtt{sid}-s_6 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 2,2,1\rangle & \mathtt{f}-\langle\rangle, \end{array} \right\rangle \end{array}\right)$$

(F)

Figure 5.822: Illustration of the pallet loading problem

## 5.421 weighted_partial_alldiff

DESCRIPTION LINKS GRAPH

**Origin** [406, page 71]

**Constraint** weighted_partial_alldiff(VARIABLES, UNDEFINED, VALUES, COST)

**Synonyms** weighted_partial_alldifferent, weighted_partial_alldistinct, wpa.

**Arguments**

| | | |
|---|---|---|
| VARIABLES | : | collection(var−dvar) |
| UNDEFINED | : | int |
| VALUES | : | collection(val−int, weight−int) |
| COST | : | dvar |

**Restrictions**

required(VARIABLES, var)
$|\text{VALUES}| > 0$
required(VALUES, [val, weight])
in_attr(VARIABLES, var, VALUES, val)
distinct(VALUES, val)

**Purpose** All variables of the VARIABLES collection that are not assigned to value UNDEFINED must have pairwise distinct values from the val attribute of the VALUES collection. In addition COST is the sum of the weight attributes associated with the values assigned to the variables of VARIABLES. Within the VALUES collection, value UNDEFINED must be explicitly defined with a weight of 0.

**Example**

$$\left(\begin{array}{l}\langle 4,0,1,2,0,0\rangle, 0, \\ \left\langle\begin{array}{ll}\texttt{val}-0 & \texttt{weight}-0, \\ \texttt{val}-1 & \texttt{weight}-2, \\ \texttt{val}-2 & \texttt{weight}-{-1}, \\ \texttt{val}-4 & \texttt{weight}-7, \\ \texttt{val}-5 & \texttt{weight}-{-8}, \\ \texttt{val}-6 & \texttt{weight}-2\end{array}\right\rangle, 8\end{array}\right)$$

The weighted_partial_alldiff constraint holds since:

- No value, except value UNDEFINED $= 0$, is used more than once.
- COST $= 8$ is equal to the sum of the weights $2$, $-1$ and $7$ of the values $1$, $2$ and $4$ assigned to the variables of VARIABLES $= \langle 4,0,1,2,0,0\rangle$.

**Typical**

$|\text{VARIABLES}| > 0$
atleast(1, VARIABLES, UNDEFINED)
$|\text{VARIABLES}| \leq |\text{VALUES}| + 2$

**Symmetries**

- Items of VARIABLES are permutable.
- Items of VALUES are permutable.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val that are both different from UNDEFINED can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val that is different from UNDEFINED can be renamed to any unused value that is also different from UNDEFINED.

**Arg. properties**

Functional dependency: COST determined by VARIABLES and VALUES.

**Usage**

In his PhD thesis [406, pages 71–72], Sven Thiel describes the following three potential scenarios of the weighted_partial_alldiff constraint:

- Given a set of tasks (i.e., the items of the VARIABLES collection), assign to each task a resource (i.e., an item of the VALUES collection). Except for the resource associated with value UNDEFINED, every resource can be used at most once. The cost of a resource is independent from the task to which the resource is assigned. The cost of value UNDEFINED is equal to $0$. The total cost COST of an assignment corresponds to the sum of the costs of the resources effectively assigned to the tasks. Finally we impose an upper bound on the total cost.
- Given a set of persons (i.e., the items of the VARIABLES collection), select for each person an offer (i.e., an item of the VALUES collection). Except for the offer associated with value UNDEFINED, every offer should be selected at most once. The profit associated with an offer is independent from the person that selects the offer. The profit of value UNDEFINED is equal to $0$. The total benefit COST is equal to the sum of the profits of the offers effectively selected. In addition we impose a lower bound on the total benefit.
- The last scenario deals with an application to an over-constraint problem involving the alldifferent constraint. Allowing some variables to take an ”undefined” value is done by setting all weights of all the values different from UNDEFINED to $1$. As a consequence all variables assigned to a value different from UNDEFINED will have to take distinct values. The COST variable allows to control the number of such variables.

**Remark**

It was shown in [406, page 104] that, finding out whether the weighted_partial_alldiff constraint has a solution or not is NP-hard. This was achieved by reduction from subset sum.

**Algorithm**

A filtering algorithm is given in [406, pages 73–104]. After showing that, deciding whether the weighted_partial_alldiff has a solution is NP-complete, [406, pages 105–106] gives the following results of his filtering algorithm with respect to consistency under the $3$ scenarios previously described:

- For scenario $1$, if there is no restriction of the lower bound of the COST variable, the filtering algorithm achieves arc-consistency for all variables of the VARIABLES collection (but not for the COST variable itself).
- For scenario $2$, if there is no restriction of the upper bound of the COST variable, the filtering algorithm achieves arc-consistency for all variables of the VARIABLES collection (but not for the COST variable itself).

- Finally, for scenario 3, the filtering algorithm achieves arc-consistency for all variables of the VARIABLES collection as well as for the COST variable.

**See also**

**attached to cost variant:** alldifferent, alldifferent_except_0.

**common keyword:** global_cardinality_with_costs *(weighted assignment)*, minimum_weight_alldifferent *(cost filtering constraint,weighted assignment)*, soft_alldifferent_var *(soft constraint)*, sum_of_weights_of_distinct_values *(weighted assignment)*.

**Keywords**

**application area:** assignment.

**characteristic of a constraint:** all different, joker value.

**complexity:** subset sum.

**constraint type:** soft constraint, relaxation.

**filtering:** cost filtering constraint.

**modelling:** functional dependency.

**problems:** weighted assignment.

| | |
|---|---|
| **Arc input(s)** | VARIABLES VALUES |
| **Arc generator** | *PRODUCT* $\mapsto$ collection(variables, values) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | • variables.var $\neq$ UNDEFINED<br>• variables.var $=$ values.val |
| **Graph property(ies)** | • **MAX_ID** $\leq 1$<br>• **SUM**(VALUES, weight) $=$ COST |

**Graph model** Parts (A) and (B) of Figure 5.823 respectively show the initial and final graph associated with the **Example** slot. Since we also use the **SUM** graph property we show the vertices of the final graph from which we compute the total cost in a box.

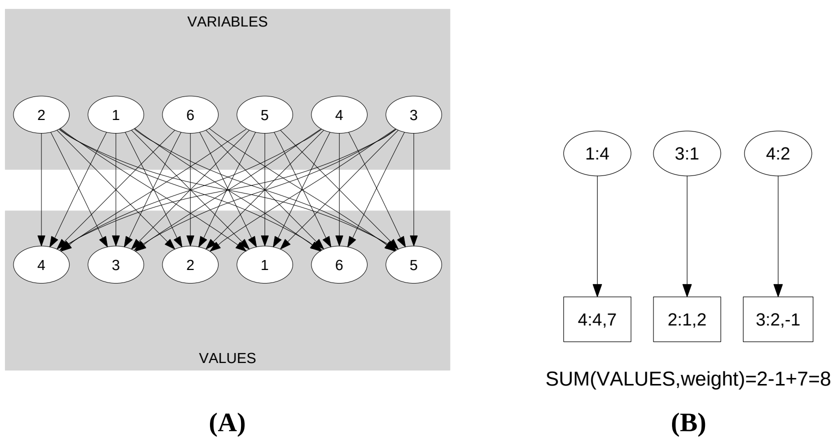



Figure 5.823: Initial and final graph of the `weighted_partial_alldiff` constraint

## 5.422 xor

DESCRIPTION LINKS AUTOMATON

**Origin** Logic

**Constraint** $\texttt{xor}(\texttt{VAR}, \texttt{VARIABLES})$

**Synonyms** odd, rel.

**Arguments**

$\texttt{VAR}$ : dvar
$\texttt{VARIABLES}$ : collection(var−dvar)

**Restrictions**

$\texttt{VAR} \geq 0$
$\texttt{VAR} \leq 1$
$|\texttt{VARIABLES}| = 2$
$\texttt{required}(\texttt{VARIABLES}, \texttt{var})$
$\texttt{VARIABLES.var} \geq 0$
$\texttt{VARIABLES.var} \leq 1$

**Purpose** Let VARIABLES be a collection of 0-1 variables $\texttt{VAR}_1, \texttt{VAR}_2$. Enforce $\texttt{VAR} = (\texttt{VAR}_1 \neq \texttt{VAR}_2)$.

**Example**

$(0, \langle 0, 0\rangle)$
$(1, \langle 0, 1\rangle)$
$(1, \langle 1, 0\rangle)$
$(0, \langle 1, 1\rangle)$

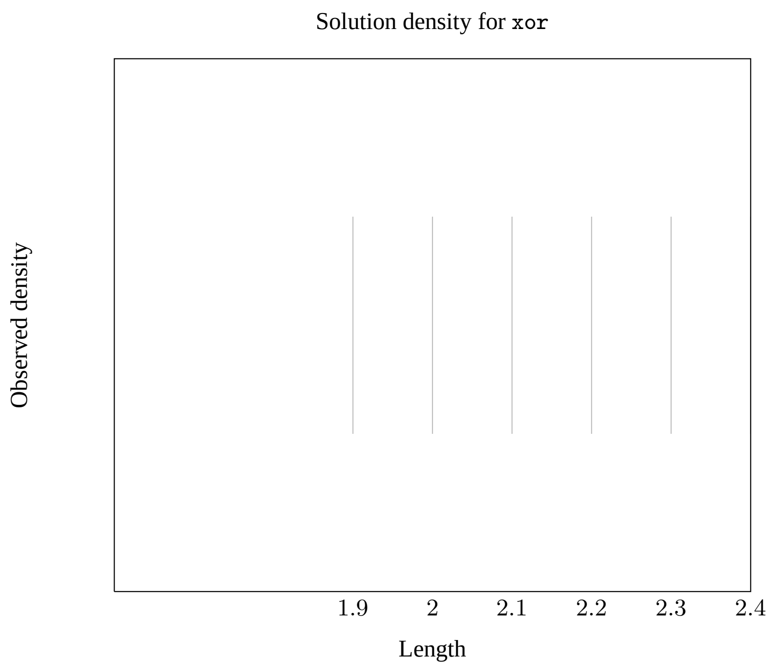
**Symmetry** Items of VARIABLES are permutable.

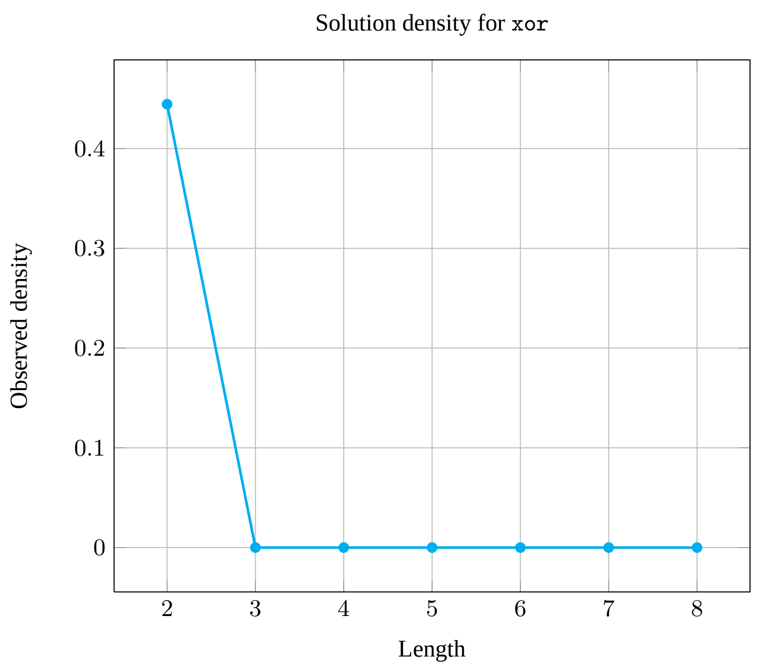
**Arg. properties** Functional dependency: VAR determined by VARIABLES.

**Counting**

| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 4 | 0 | 0 | 0 | 0 | 0 | 0 |

Number of solutions for xor: domains $0..n$

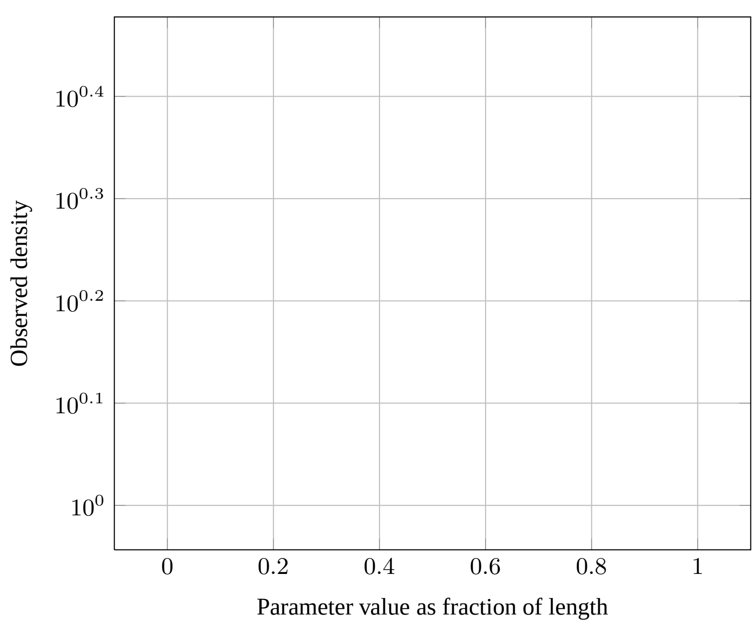

Solution density for xor
Observed density
1.9
2
2.1
2.2
2.3
2.4
Length

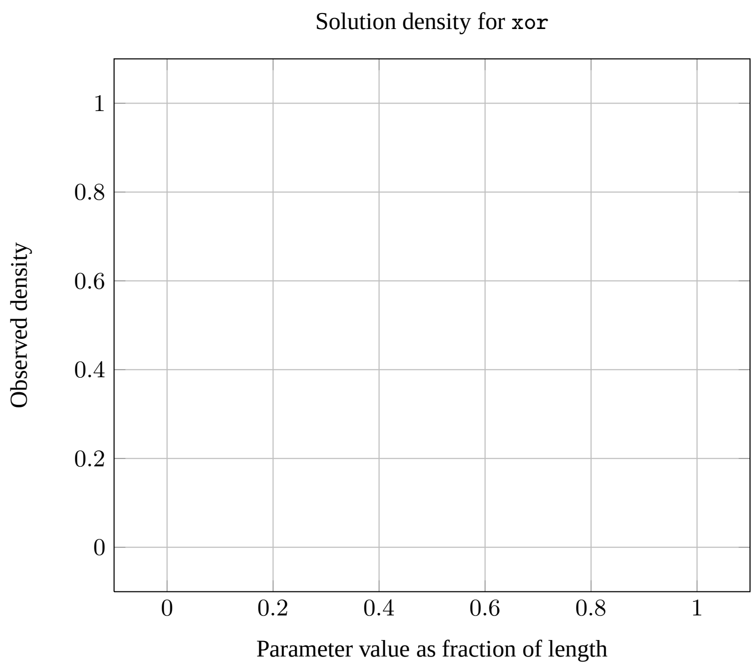

Solution density for xor
Observed density
0.4
0.3
0.2
0.1
0
2
3
4
5
6
7
8
Length

| Length ($n$) | | 2 |
|---|---|---|
| Total | | 4 |
| Parameter value | 0 | 2 |
| | 1 | 2 |

Solution count for `xor`: domains $0..n$

Solution density for `xor`






**Systems** reifiedXor in **Choco**, rel in **Gecode**, xorbool in **JaCoP**, #\ in **SICStus**.

**See also** **common keyword:** `and`, `equivalent`, `imply`, `nand`, `nor`, `or` *(Boolean constraint)*.

**implies:** `atleast_nvalue`, `soft_all_equal_max_var`, `soft_all_equal_min_var`.

**Keywords** **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

**constraint arguments:** pure functional dependency.

**constraint network structure:** Berge-acyclic constraint network.

**constraint type:** Boolean constraint.

**filtering:** arc-consistency.

**modelling:** functional dependency.

**Automaton** Figure 5.824 depicts the automaton associated with the xor constraint. To the first argument VAR of the xor constraint corresponds the first signature variable. To each variable $\mathtt{VAR}_i$ of the second argument VARIABLES of the xor constraint corresponds the next signature variable. There is no signature constraint.

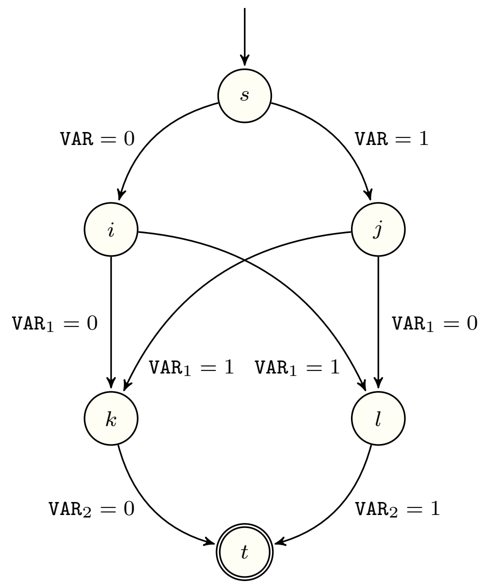



Figure 5.824: Automaton of the xor constraint

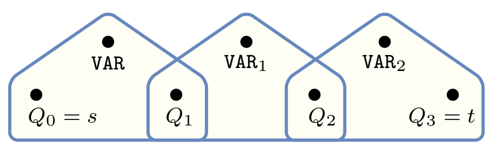



Figure 5.825: Hypergraph of the reformulation corresponding to the automaton of the xor constraint

## 5.423 zero_or_not_zero

DESCRIPTION LINKS

**Origin** Arithmetic.

**Constraint** zero_or_not_zero(VAR1, VAR2)

**Synonyms** zeros_or_not_zeros, not_zero_or_zero, not_zeros_or_zeros.

**Arguments**

| | | |
|---|---|---|
| VAR1 | : | dvar |
| VAR2 | : | dvar |

**Purpose** Enforce the fact that either both variables are equal to $0$, or both variables are not equal to $0$.

**Example** $(1, 8)$

The zero_or_not_zero constraint holds since values $1$ and $8$ are both not equal to zero.

**Symmetry** Arguments are permutable w.r.t. permutation (VAR1, VAR2).

**See also** **implied by:** abs_value, divisible_or, eq, sign_of.

**implies (if swap arguments):** abs_value.

**Keywords** **constraint arguments:** binary constraint.

**constraint type:** predefined constraint, arithmetic constraint.

## 5.424 zero_or_not_zero_vectors

DESCRIPTION LINKS

**Origin** Tournament scheduling

**Constraint** zero_or_not_zero_vectors(VECTORS)

**Synonyms** zeros_or_not_zeros_vectors, not_zero_or_zero_vectors, not_zeros_or_zeros_vectors.

**Type** VECTOR : collection(var−dvar)

**Argument** VECTORS : collection(vec − VECTOR)

**Restrictions**
|VECTOR| $\geq 1$
required(VECTOR, var)
|VECTORS| $\geq 1$
required(VECTORS, vec)
same_size(VECTORS, vec)

**Purpose** Given a collection of vectors enforces for each vector that either all its components are equal to $0$, or all its components are different from $0$. In addition imposes that at least one $0$ is used.

**Example**

$$\left(\left\langle\begin{array}{l}\texttt{vec} - \langle 5,6\rangle,\\ \texttt{vec} - \langle 5,6\rangle,\\ \texttt{vec} - \langle 0,0\rangle,\\ \texttt{vec} - \langle 9,3\rangle,\\ \texttt{vec} - \langle 0,0\rangle\end{array}\right\rangle\right)$$

The zero_or_not_zero_vectors constraint holds since:

- Both components of the first vector $\langle 5, 6\rangle$ are different from $0$.
- Both components of the second vector $\langle 5, 6\rangle$ are different from $0$.
- Both components of the third vector $\langle 0, 0\rangle$ are equal to $0$.
- Both components of the fourth vector $\langle 9, 3\rangle$ are different from $0$.
- Both components of the fifth vector $\langle 0, 0\rangle$ are equal to $0$.

**Typical**
|VECTOR| $> 1$
|VECTORS| $> 1$

**Arg. properties** Contractible wrt. VECTORS.

**Keywords** **characteristic of a constraint:** vector.
**constraint type:** predefined constraint, arithmetic constraint.