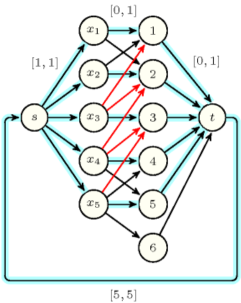
L04: Global Constraints



Prof. Tias Guns and Dr. Dimos Tsouros

KU LEUVEN

Partly based on slides from Pierre Flener, Uppsala University.

Outline

Definition

Global Constraint: an expressive and concise constraint that

- is defined over a non-fixed number of variables,
- captures a specific combinatorial substructure commonly found in constraint satisfaction problems

Example

Well-known global constraints

- ALLDIFFERENT()
- CIRCUIT()
- CUMULATIVE()
- ...

Outline

Why use Global Constraints?

- **Expressiveness**!
  More compact and intuitive models, closer to problem definition
  Many expressive predicates are available:
  islands of common combinatorial structure are identified in declarative high-level abstractions.
  See the Global-Constraint Catalogue.

- **Efficiency**! (In CP solvers)
  Faster solving,
  due to better inference and relaxation,
  enabled by more global information in the model
  (If supported by the used solver.)

Why use Global Constraints?

+ **Expressiveness**!
  More compact and intuitive models, closer to problem definition.
  Many expressive global constraints available
  - **Simplified modeling**: Global constraints enable the modeler to express complex conditions simpler.
    Simpler modeling reduces the chance of modeling errors.
  - **Compactness**: Instead of writing multiple smaller constraints, a single global constraint can capture the entire logic.
    Make the model more intuitive and readable

Example

Task allocation: I want all tasks to be allocated to a different team.

- *Without global*:
  $Task_0 \neq Task_1,\ Task_0 \neq Task_2,\ Task_1 \neq Task_2,\ \ldots$
- *With global*: $\textsc{AllDifferent}(\text{Task})$

Why use Global Constraints? – CPMpy

+ **Expressiveness**!
  More compact and intuitive models, closer to problem definition.
  Many expressive global constraints available
  - **Simplified modeling**: Global constraints enable the modeler to express complex conditions simpler.
    Simpler modeling reduces the chance of modeling errors.
  - **Compactness**: Instead of writing multiple smaller constraints, a single global constraint can capture the entire logic.
    Make the model more intuitive and readable

Example

Task allocation: I want all tasks to be allocated to a different team

- *Without global*:
  `Task[0] != Task[1], Task[0] != Task[2], Task[1] != Task[2], ...`
- *With global*: `cp.AllDifferent(Task)`

Why use Global Constraints?

+ **Efficiency**! (In CP solvers)
  Faster solving, due to better inference and relaxation, enabled by more global information in the model (If supported by the used solver.)
  - More global information in one constraint can result to advanced filtering of the search space
  - Specialized algorithms to detect conflicts faster during solving.

Example

Task allocation: I want to allocate n tasks to m teams, s.t. each task is assigned to a different team. Assume we have $m < n$.

- *Without global*: Each $! =$ constraint needs 2 values (teams) available for its tasks. Will realize that there are not enough values only after extensive search of possible assignments
- *With global*: ALLDIFFERENT(Task) will directly recognise that we cannot put m different values (teams) in n variables (tasks) if we have $m < n$ during search. $\leftarrow$ **Pigeonhole principle**

Modelling with Global Constraints:

Several global constraints exist, capturing different combinatorial properties: Global-Constraint Catalogue `https://sofdem.github.io/gccat`

Functional Global Constraints: Global constraints that have a functional component, such as MINIMUMEQ(), MAXIMUMEQ(), COUNTEQ(), NVALUEEQ(), etc.

Definition

A global constraint G(V) is functional if and only if there exists a partitioning of the arguments V of the constraint into two non-empty and non-overlapping subsets $V1$, $V2$, such that *the assignment of variables in subset* $V2$ *is defined using a function on the subset* $V1$.

Modelling with Global Constraints:

Definition

A global constraint G(V) is functional if and only if there exists a partitioning of the arguments V of the constraint into two non-empty and non-overlapping subsets $V1$, $V2$, such that *the assignment of variables in subset $V2$ is defined using a function on the subset $V1$*.

In many cases, this involves associating the value of the functional component with a variable:

Examples

- MAXIMUMEQ(X, v) implies that MAXIMUM(X) $= v$, allowing v to be used in other expressions.
- MINIMUMEQ(X, v) implies that MAXIMUM(X) $= v$, allowing v to be used in other expressions.
- ...

Modelling with Global Constraints – CPMpy

- Several commonly-used global constraints are available: `AllDifferent`, `AllEqual`, `Cumulative`, `Table`, ... API documentation: `http://cpmpy.readthedocs.io/en/latest/api/expressions/globalconstraints.html`
- **Functional Global Constraints**: A subset of them, the ones associating the result of a function to a variable, are modelled as 'Global functions' in CPMpy, representing only the *functional* component: e.g. `cp.Count(X,1)`
  - Can be used nested in any expression: e.g. `cp.Count(X,1)` > 0
  - Several Global functions available: `Minimum`, `Maximum`, `Count`, `NValue`, ... API documentation: `http://cpmpy.readthedocs.io/en/latest/api/expressions/globalfunctions.html`
- All global constraints can be reified - be nested in other expressions: e.g. `cp.sum(cp.AllDifferent(x), cp.AllDifferent(y), cp.AllDifferent(z))` > 2
- Can use globals that the chosen solver might not support. CPMpy will translate this to a lower-level solver decomposition for you.

Outline

1. Definition
2. Motivation
3. Common Global Constraints
    - ALLDIFFERENT()
    - GLOBALCARDINALITYCOUNT()
    - Scheduling with CUMULATIVE() and NOOVERLAP()
    - CIRCUIT()
    - TABLE()
    - COUNTEQ()
    - NVALUEEQ()
    - ELEMENTEQ()

Outline

AllDifferent()

Definition (Laurière, 1978)

The AllDifferent(X) constraint holds if and only if all the elements of the array X of decision variables take distinct values.

Its decomposition is a conjunction of $\frac{n \cdot (n-1)}{2}$ disequality constraints when X has n elements:

$$\forall i, j \in \{1, \ldots, n\}, i < j \implies X[i] \neq X[j]$$

Examples

- *n*-Queens, Photo Alignment problem, Student Seating problem.
- Sudoku, Room assignment, Task allocation ...

Variant: The AllDifferentExceptN(X, N) constraint allows multiple occurrences of the exception values in the set N.

AllDifferent() – CPMpy

Definition (Laurière, 1978)

The AllDifferent(X) constraint holds if and only if all the elements of the array X of decision variables take distinct values.

Its decomposition is a conjunction of $\frac{n \cdot (n-1)}{2}$ disequality constraints when X has n elements:

```
[var1 != var2 for var1, var2 in all_pairs(X)]
```

Examples

- *n*-Queens, Photo Alignment problem, Student Seating problem.
- Sudoku, Room assignment, Task allocation . . .

Variant: The AllDifferentExceptN(X, N) constraint allows multiple occurrences of the exception values in the set N.

Example

Sudoku: we want different values in rows, columns and blocks
using the AllDifferent(X) global constraint

$$\begin{array}{lll}
\text{AllDifferent}(\{G_{ij} \mid j \in \{1,\ldots,9\}\}) & \forall i \in \{1,\ldots,9\} & \text{(rows)} \\
\text{AllDifferent}(\{G_{ij} \mid i \in \{1,\ldots,9\}\}) & \forall j \in \{1,\ldots,9\} & \text{(columns)} \\
\text{AllDifferent}(\{G_{kl} \mid k \in \{i,\ldots,i+2\}, l \in \{j,\ldots,j+2\}\}) & \forall i,j \in \{1,4,7\} & \text{(blocks)}
\end{array}$$

Way more expressive (and efficient) than using binary not equal constraints

$$\begin{array}{ll}
G_{ij} \neq G_{ik} & \forall i \in \{1,\ldots,9\}, \forall j,k \in \{1,\ldots,9\}, j < k \quad \text{(rows)} \\
G_{ij} \neq G_{kj} & \forall j \in \{1,\ldots,9\}, \forall i,k \in \{1,\ldots,9\}, i < k \quad \text{(columns)} \\
G_{kl} \neq G_{mn} & \forall i,j \in \{1,4,7\},\ \forall k,m \in \{i,\ldots,i+2\}, \\
& \forall l,n \in \{j,\ldots,j+2\}, (k,l) < (m,n) \quad \text{(blocks)}
\end{array}$$

Example (CPMpy)

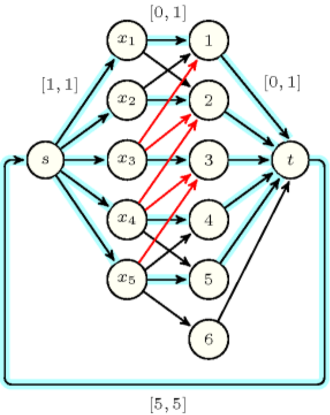
Sudoku: we want different values in rows, columns and blocks

```
[cp.AllDifferent(row) for row in grid],
[cp.AllDifferent(col) for col in grid.T],  # numpy's Transpose
[cp.AllDifferent(grid[i:i+3, j:j+3]) \
    for i in range(0, 9, 3) for j in range(0, 9, 3)],
```

Way more efficient solving than when using binary not equal constraints

```
[[cell1 != cell2 for cell1, cell2 in all_pairs(row)]
 for row in grid],
[[cell1 != cell2 for cell1, cell2 in all_pairs(col)]
 for col in grid.T],
[[cell1 != cell2 for cell1, cell2
  in all_pairs(grid[i:i+3, j:j+3].flat)]
 for i in [0, 3, 6] for j in [0, 3, 6]]
```

+ **Efficiency**! Better propagation in CP solvers due to capturing global properties



Flow model for the ALLDIFFERENT() constraint

- feasible flows in the flow model = solutions to the constraint
- detect arcs that cannot carry flow in any feasible solutions → remove values from the domains of variables
- Blue arcs represent feasible flows, red arcs represent infeasible ones
- **Detect inconsistency early**: flow from (some) variables cannot be directed through the available values → pigeonhole problem
  - Not detected early through binary constraints

Outline

GLOBALCARDINALITYCOUNT()

Definition (Régin, 1996)

The GLOBALCARDINALITYCOUNT(X, V, C) constraint holds if and only if the number of occurrences of each value V_i in the list of variables X is equal to C_i.

Its decomposition is expressed as:

$$\forall j \in \{1, \ldots, |V|\}, \quad \text{CountEq}(X, V_j, C_j)$$

which is:

$$\sum_i [X_i = V_j] = C_j$$

This constraint is equivalent to ALLDIFFERENT(X) if:

$$V = \bigcup_i \text{Domain}(X_i) \quad \text{and} \quad \text{Domain}(C_j) = \{0, 1\} \quad \forall j$$

However, always use the most specific available constraint predicate!

GlobalCardinalityCount() – CPMpy

Definition (Régin, 1996)

The GlobalCardinalityCount(X, V, C) constraint holds if and only if the number of occurrences of each value $V[i]$ in the list of variables X is equal to $C[i]$.

Its decomposition in CPMpy is:

```
[cp.Count(X, v) == c for v, c in zip(V, c)]
```

Add `closed=True` as a parameter if `V` must be forced as the domain of the variables in `X`.

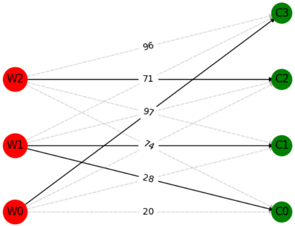
This constraint is equivalent to AllDifferent(X) if:

$$V = \bigcup_i \text{Domain}(X[i]) \quad \text{and} \quad \text{Domain}(C[j]) = \{0, 1\} \quad \forall j$$

However, always use the most specific available constraint predicate!

Facility Location

Warehouse location: we want to find which customers each warehouse will serve



Example

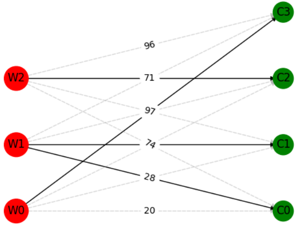
Use the GLOBALCARDINALITYCOUNT(assignments, warehouses, capacities) constraint to ensure that each warehouse is assigned the correct number of customers.

GLOBALCARDINALITYCOUNT() defines the exact number of occurencies, not a bound, i.e. takes capacity for each variable

But every argument can be a variable! So, you can use variables (with the specified bounds) as capacities

Facility Location – CPMpy

Warehouse location: we want to find which customers each warehouse will serve



Example (CPMpy)

```
# GlobalCardinalityCount constraint to ensure each warehouse is
    assigned correctly
model.add(cp.GlobalCardinalityCount(assignments, list(range(
    n_warehouses)), capacities))
```

GlobalCardinalityCount() defines the exact nr of occurencies, not a bound

But every argument can be a variable! So, you can use variables (with the specified bounds) as capacities

But, why not model it directly using `cp.Count`? We will discuss this later!

Outline

1. Definition
2. Motivation
3. Common Global Constraints
   - AllDifferent()
   - GlobalCardinalityCount()
   - Scheduling with Cumulative() and NoOverlap()
   - Circuit()
   - Table()
   - CountEq()
   - NvalueEq()
   - ElementEq()

Scheduling

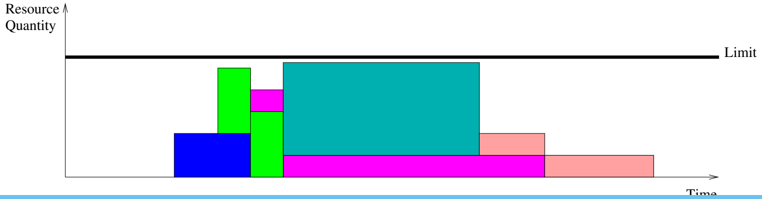
Assume we need to schedule a set of non-interruptible tasks under constraints (on resources, precedences, ...) such that the last task has the earliest end.

Definition

A task T_i is defined as a triple of parameters $T_i = \langle S_i, D_i, R_i \rangle$ or variables, where:

- S_i is the starting time of task T_i
- D_i is the duration of task T_i
- R_i is the quantity of a global reusable resource needed by T_i

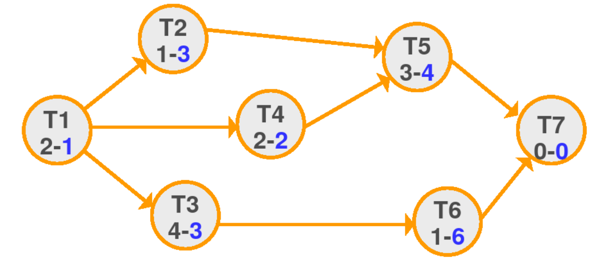
Tasks may be run in parallel when the capacity of the global resource suffices.



Definition

A precedence constraint of task T_1 on task T_2 requires that T_1 ends before T_2 starts. We say that task T_1 precedes task T_2.

Example (courtesy Magnus Rattfeldt)



Sample tasks (circles), durations (black numbers), resource requirements (blue numbers), and precedences (orange arrows). Task T7 is a dummy task, as we do not know which of tasks T5 and T6 will end last.

Let us temporarily **ignore the capacitated global reusable resource**:
If we have an uncapacitated global reusable resource or each task has enough of its own local reusable resource, then the (polynomial-time-solvable problem) of **finding the earliest ending time, under only the precedence constraints**, for performing all the tasks can be modelled using linear inequalities.

Example (continued)

The precedence constraints indicated by the orange arrows on slide 26 are modelled as follows, based on the task durations indicated there in black:

$$
\begin{aligned}
S_0 + D_0 \leq S_1, &\quad S_0 + D_0 \leq S_2, \\
S_0 + D_0 \leq S_3, &\quad S_1 + D_1 \leq S_4, \\
S_2 + D_2 \leq S_5, &\quad S_3 + D_3 \leq S_4, \\
S_4 + D_4 \leq S_6, &\quad S_5 + D_5 \leq S_6 \\
&\quad \text{Minimize } S_6
\end{aligned}
$$

Let us temporarily **ignore the capacitated global reusable resource**:
If we have an uncapacitated global reusable resource or each task has enough of its own local reusable resource, then the (polynomial-time-solvable problem) of **finding the earliest ending time, under only the precedence constraints**, for performing all the tasks can be modelled using linear inequalities.

Example (continued)

The precedence constraints indicated by the orange arrows on slide 26 are modelled as follows, based on the task durations indicated there in black:

```
model.add([S[0]+D[0] <= S[1], S[0]+D[0] <= S[2],
           S[0]+D[0] <= S[3], S[1]+D[1] <= S[4],
           S[2]+D[2] <= S[5], S[3]+D[3] <= S[4],
           S[4]+D[4] <= S[6], S[5]+D[5] <= S[6]])

model.minimize(S[6])
```

But how to model the capacitated global resource?

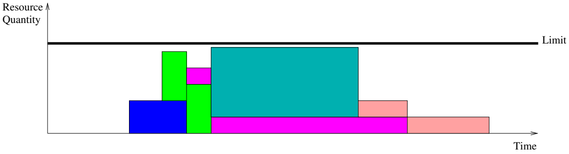
Definition (Aggoun and Beldiceanu, 1993)

The CUMULATIVE(S, D, R, c) constraint, for tasks $T_i = \langle S_i, D_i, R_i \rangle$, holds if and only if the total resource usage does not exceed the capacity c at any time.

The CUMULATIVE(S, D, R, c) ensures the following:

$$\sum_{i: S_i \leq t < S_i + D_i} R_i \leq c, \quad \forall t$$

Note that CUMULATIVE(S, D, R, c) does not ensure any precedence constraints between the tasks:
these have to be stated separately (as on the previous slide).



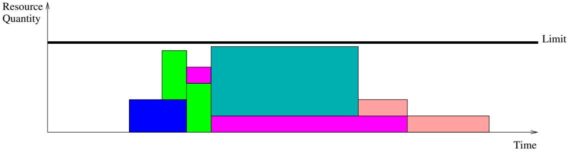
Example (Cumulative)

To ensure that the global reusable resource capacity of $c = 8$ units, say, is never exceeded under the resource requirements of the tasks indicated in blue on slide 26, use the following constraint:

CUMULATIVE(S, D, [1, 3, 3, 2, 4, 6, 0], 8)

Along with the precedence constraints described before:

$$S_0 + D_0 \le S_1, \quad S_0 + D_0 \le S_2, \quad S_0 + D_0 \le S_3, \quad S_1 + D_1 \le S_4,$$
$$S_2 + D_2 \le S_5, \quad S_3 + D_3 \le S_4, \quad S_4 + D_4 \le S_6, \quad S_5 + D_5 \le S_6$$
$$\text{Minimize } S_6$$



Example (Cumulative – CPMpy)

To ensure that the global reusable resource capacity of `c` = 8 units, say, is never exceeded under the resource requirements of the tasks indicated in blue on slide 26, use the following constraint:

```
# Need to define variables for E: end time of tasks!
model += cp.Cumulative(S,D,E,[1,3,3,2,4,6,0],8)
```

Along with the precedence constraints described before:

```
model.add([S[0]+D[0] <= S[1], S[0]+D[0] <= S[2],
           S[0]+D[0] <= S[3], S[1]+D[1] <= S[4],
           S[2]+D[2] <= S[5], S[3]+D[3] <= S[4],
           S[4]+D[4] <= S[6], S[5]+D[5] <= S[6]])

model.minimize(S[6])
```

Scheduling – NoOverlap

What if I just want tasks scheduled to not overlap?
A non-overlap constraint between tasks T_1 and T_2 requires that either T_1 precedes T_2 or T_2 precedes T_1.

Definition (Carlier, 1982)

The NoOverlap(S, D) constraint, where each task T_i has the starting time S_i and duration D_i, holds if and only if no two tasks T_i and T_j overlap in time.

Its decomposition is:

$$S_i + D_i \leq S_j \quad \text{or} \quad S_j + D_j \leq S_i \quad \forall i, j \text{ with } i \neq j$$

Can be also modeled as: Cumulative(S, D, [1, 1, . . . , 1], 1)
Always use the most specific available constraint predicate!

Scheduling – NoOverlap – CPMpy

What if I just want tasks scheduled to not overlap?
A non-overlap constraint between tasks T_1 and T_2 requires that either T_1 precedes T_2 or T_2 precedes T_1.

Definition (Carlier, 1982)

The NoOverlap(S, D) constraint, where each task T_i has the starting time S_i and duration D_i, holds if and only if no two tasks T_i and T_j overlap in time.

In CPMpy `cp.NoOverlap(S,D,E)` also needs an argument for the end times of the tasks!

Its decomposition in CPMpy is:

- ```
  for i in range(n):
    model += S[i] + D[i] == E[i]
  ```
- ```
  for i,j in all_pairs(range(n)):
    model += (E[i] <= S[j]) | (E[j] <= S[i])
  ```

Outline

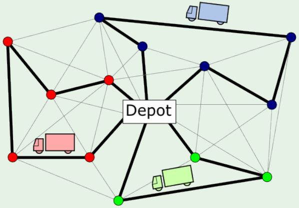
Enabling the representation of a circuit in a digraph

- Let decision variable S_v denote the successor of vertex v in the circuit.
- The domain of S_v is the set of vertices to which there is an arc from vertex v.

Definition (Laurière, 1978; Beldiceanu and Contejean, 1994)

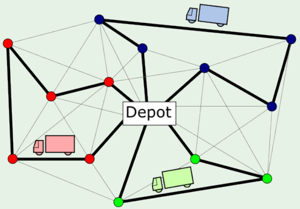
The CIRCUIT(S) constraint holds if and only if $\forall$ v the arcs $v \to S_v$ form a Hamiltonian circuit: each vertex is visited exactly once.

Example (Vehicle Routing)



- Find optimal routes for multiple vehicles visiting a set of locations.
- 1 vehicle = Traveling Salesman Problem.

Example (Vehicle routing)



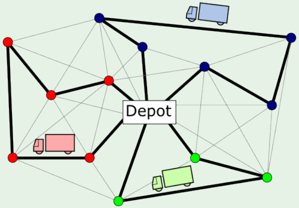
- Find optimal routes for multiple vehicles visiting a set of locations
- 1 vehicle = Traveling Salesman Problem

Travelling salesman problem (generalise this for vehicle routing problems with multiple vehicles or with side constraints):

$$\text{Circuit}(S)$$

$$\text{Minimize} \quad \sum_{v=1}^{n} \text{distance}(v, S_v)$$

Example (Vehicle routing)



- Find optimal routes for multiple vehicles visiting a set of locations
- 1 vehicle = Traveling Salesman Problem

Travelling salesman problem (generalise this for vehicle routing problems with multiple vehicles or with side constraints):

```
model.add(cp.Circuit(S))
model.minimize(sum(distance[city, S[city]] for city in range(
    cities)))
```

Outline

Definition

The TABLE(X, T) constraint holds if and only if the values of the 1D array X of decision variables form a row of the 2D array T of values. In other words, it restricts the values of the given variables in X to combinations listed in the predefined table T.

The 2D array T provides an extensional definition of the constraint we impose. Its decomposition is as follows:

$$\exists \text{ row } \in T \text{ such that } \forall i,\ X_i = \text{row}_i$$

Example

Assigning Workers W_1, W_2 to Shifts, but only specific assignments are allowed:

$$\mathrm{T} = \{(1,2),(1,3),(2,3)\}$$

The TABLE() constraint is applied as:

$$\textsc{Table}([W_1, W_2], T)$$

Definition

The TABLE(X, T) constraint holds if and only if the values of the 1D array X of decision variables form a row of the 2D array T of values. In other words, it restricts the values of the given variables in X to combinations listed in the predefined table T.

The 2D array T provides an extensional definition of the constraint we impose. Its decomposition in CPMpy is the following:

```
[cp.any(cp.all(ai == ri for ai, ri in zip(arr, row)) for row in tab)]
```

Example

Assigning Workers W_1, W_2 to Shifts, but only specific assignments are allowed:

```
# Allowed combinations of shifts (table of allowed tuples)
T = [(1, 2), (1, 3), (2, 3)]
model.add(cp.Table([W1, W2], T))
```

Outline

1. Definition
2. Motivation
3. Common Global Constraints
    - AllDifferent()
    - GlobalCardinalityCount()
    - Scheduling with Cumulative() and NoOverlap()
    - Circuit()
    - Table()
    - CountEq()
    - NvalueEq()
    - ElementEq()

Given an array of decision variables X, we often want to count the number of decision variables in X that are equal to a decision variable (or value) *val*.

Definition (The CountEq() functional global constraint)

The CountEq(X, val, res) functional glboal constraint holds if and only if the number of occurrences of the numeric value/value of the variable *res* in the array of decision variables X is equal to *res*.

Its decomposition is the following:

$$\sum_i [X_i = v] = res$$

Example (Unweighted Photo Alignment Problem)

$$\textsc{CountEq}(\{|\mathrm{Pos}_{who} - \mathrm{Pos}_{whom}| \mid (who, whom) \in \mathrm{Wishes}\}, 1, res)$$

$$\mathrm{Maximize}\,(res)$$

Given an array of decision variables X, we often want to count the number of decision variables in X that are equal to a decision variable (or value) *val*.

Definition (The CountEq() functional global constraint)

The CountEq(X, val, res) functional glboal constraint holds if and only if the number of occurrences of the numeric value/value of the variable *res* in the array of decision variables X is equal to *res*.

Its decomposition in CPMpy is the following:

```
res == cp.sum(X == val)
```

Example (Unweighted Photo Alignment Problem from L02)

```
model +=
cp.Count([abs(Pos[who] - Pos[whom]) for (who,whom) in Wishes], 1) == res
model.maximize(res)
```

Given an array of decision variables *X*, we often want to count the number of decision variables in *X* that are equal to a decision variable (or value) *val*.

Definition (The CountEq() functional global constraint)

The CountEq(X, val, res) functional glboal constraint holds if and only if the number of occurrences of the numeric value/value of the variable *res* in the array of decision variables *X* is equal to *res*.

Its decomposition in CPMpy is the following:

```
res == cp.sum(X == val)
```

Functional formulation (without explicit `res`):

Example (Unweighted Photo Alignment Problem from L02)

```
m.maximize(cp.Count([abs(Pos[who] - Pos[whom]) for (who,whom) in Wishes], 1))
```

A Common Source of Inefficiency in Models

Group constraints in (more specific) globals when possible:

Example

The constraint specification

$$\forall j \in \text{index\_set}(V),\; \text{CountEq}(X, V_j, C_j)$$

should be reformulated, due to the shared array X for each j, into:

$$\textsc{GlobalCardinalityCount}(X, V, C)$$

by applying the default definition backwards:

- At worst, it will be applied forward while decomposing;
- At best, the used solver will have better inference.

A Common Source of Inefficiency in Models

Group constraints in (more specific) globals when possible:

Example

The constraint specification

```
for v, c in zip(V, C): cp.Count(X, v) == c
```

should be reformulated, due to the shared array X for each j, into:

```
cp.GlobalCardinalityCount(X,V,C);
```

by applying the default definition backwards:

- At worst, it will be applied forward while decomposing;
- At best, the used solver will have better inference.

Outline

Definition (Pachet and Roy, 1999)

The NValueEq(X, res) functional global constraint holds if and only if the number of distinct values taken by the elements of the array X of decision variables is equal to *res*. If array X is 1d, with length n, then this means:

$$|\{X_0, \ldots, X_{n-1}\}|$$

If $|X| = n$ then NValueEq(X, n) means AllDifferent(X),
but: always use the most specific available constraint predicate!

Example

Graph colouring: Different colour on neighbouring nodes + minimize the number of colours, i.e., minimize the number of *distinct* values of our variables:

$$\begin{array}{ll} \text{node}_1 \neq \text{node}_2, & \forall(\text{node}_1, \text{node}_2) \in \text{Edges} \\ \text{NValueEq}(\text{nodes}, res) & \\ \text{minimize } res & \end{array}$$

Definition (Pachet and Roy, 1999)

The NVALUEEQ(X, res) functional global constraint holds if and only if the number of distinct values taken by the elements of the array X of decision variables is equal to *res*. If array X is 1d, with length n, then this means:

$$|\{X_0, \ldots, X_{n-1}\}|$$

If $|X| = n$ then NVALUEEQ(X, n) means ALLDIFFERENT(X),
but: always use the most specific available constraint predicate!

Example

Graph colouring: Different colour on neighbouring nodes + minimize the number of colours, i.e., minimize the number of *distinct* values of our variables:

```
# Adjacent vertices must have different colors
model.add([nodes[i] != nodes[j] for (i, j) in Edges])
# Minimize distinct colours used
model.minimize(NValue(nodes))
```

Outline

1. Definition
2. Motivation
3. Common Global Constraints
   - AllDifferent()
   - GlobalCardinalityCount()
   - Scheduling with Cumulative() and NoOverlap()
   - Circuit()
   - Table()
   - CountEq()
   - NvalueEq()
   - ElementEq()

Modeling an unknown element of an array.

Example (Job allocation at minimal salary cost)

Given the salaries of work applicants *Salary* for different jobs,
find a work applicant for each job **such that** some constraints (on the qualifications of the work applicants for the jobs, on workload distribution, etc) are satisfied and the **total salary cost is minimal**:

$$\text{total\_cost} = \sum_{w \in \text{workers}} \text{Salary}_w$$

$$\text{minimize total\_cost}$$

We do not know at modelling time the worker allocated to each job!

Modeling an unknown element of an array.

Example (Job allocation at minimal salary cost)

Given the salaries of work applicants *Salary* for different jobs,
find a work applicant for each job **such that** some constraints (on the qualifications of the work applicants for the jobs, on workload distribution, etc) are satisfied and the **total salary cost is minimal**:

```
# Objective: Minimize the total salary cost for assigned jobs
Salary = cp.cpm_array(Salary) # convert into a cpmpy-compatible array
total_cost = cp.sum([Salary[worker] for worker in workers])
model.minimize(total_cost)
```

We do not know at modelling time the worker allocated to each job!
We need to express the relation between variables/expressions and values of arrays based on unknown indices . . .

The ElementEq() Global Constraint

We need to express the the relation between variables/expressions and values of arrays based on unknown indices . . .

Definition (Van Hentenryck and Carillon, 1988)

The ElementEq(Arr, idx, res) functional global constraint holds if and only if the value $Arr[idx] = res$, where *Arr* is array of decision variables or constants and *idx* is an integer decision variable.

For better model readability, the ElementEq() predicate is typically not directly used, when modeling languages allow direct indexing; e.g. `res == Arr[idx]` even if `idx` is an integer expression involving at least one decision variable.

Note that Element() can be multi-dimensional: `Arr[idx1, idx2]` We assume that the indices can only take values within the bounds of the array.

Summary

Global Constraints:

- Non-fixed arity
- Model global properties of the problem

Building blocks that can be used in modeling a variety of problems . . .

- *Expressiveness*!! Closer to problem definition

Better inference during solving . . . *Efficiency*!!

- Due to modeling global properties together
- Potentially fewer auxiliary variables
- Potentially more effective propagators