Global Constraints

- Capture complex, non-binary and recurring combinatorial substructures arising in a variety of applications.
- Embed specialized propagation which exploits the substructure.

Benefits of Global Constraints

- Modelling benefits
  - Reduce the gap between the problem statement and the model.
  - May allow the expression of constraints that are otherwise not possible to state using primitive constraints (semantic).
- Solving benefits
  - Strong inference in propagation (operational).
  - Efficient propagation (algorithmic).

Some Groups of Global Constraints

- Counting
- Sequencing
- Scheduling
- Ordering
- Balancing
- Distance
- Packing
- Graph-based
- …

Counting Constraints

- Restrict the number of variables satisfying a condition or the number of times values are taken.

Alldifferent Constraint

- alldifferent($[X_1, X_2, \ldots, X_k]$) iff
  $X_i \neq X_j$ for $i < j \in \{1,\ldots,k\}$
  - permutation constraint with $|D(X_i)| = k$.
  - alldifferent([3,5,2,1,4])
- Useful in a variety of context, like:
  - puzzles (e.g., sudoku and n-queens);
  - timetabling (e.g. allocation of activities to different slots);
  - scheduling (e.g. a team can play at most once in a week);
  - configuration (e.g. a particular product cannot have repeating components).

Nvalue Constraint

- Constrains the number of distinct values assigned to the variables.
- Nvalue($[X_1, X_2, \ldots, X_k]$, N) iff $N = |\{X_i \mid 1 \leq i \leq k\}|$
  - Nvalue([1, 2, 2, 1, 3], 3).
  - alldifferent when N = k.
- Useful e.g. in:
  - resource allocation (e.g. limit the number of resource types).

Global Cardinality Constraint

- Constrains the number of times each value is taken by the variables.
- gcc($[X_1, X_2, \ldots, X_k], [v_1, \ldots, v_m], [O_1, \ldots, O_m]$) iff

  forall $j \in \{1,\ldots, m\}$ $O_j = |\{X_i \mid X_i = v_j, 1 \leq i \leq k\}|$
  - gcc([1, 1, 3, 2, 3], [1, 2, 3, 4], [2, 1, 2, 0])
  - alldifferent when $O_j \leq 1$.
- Useful e.g. in:
  - resource allocation (e.g. limit the usage of each resource).

Among Constraint

- Constrains the number of variables taken from a given set of values.
- among($[X_1, X_2, \ldots, X_k]$, s, N) iff
  $$N = |\{i \mid X_i \in s, 1 \leq i \leq k\}|$$
  - among([1, 5, 3, 2, 5, 4], {1,2,3,4}, 4)
- among($[X_1, X_2, \ldots, X_k]$, s, l, u) iff
  $$l \leq |\{i \mid X_i \in s, 1 \leq i \leq k\}| \leq u$$
  - among([1, 5, 3, 2, 5, 4], {1,2,3,4}, 3, 4)
- Useful in sequencing problems, as we see next.

Sequencing Constraints

- Ensure a sequence of variables obey certain patterns.

Sequence/AmongSeq Constraint

- Constrains the number of values taken from a given set in any subsequence of q variables.
- sequence(l, u, q, $[X_1, X_2, \ldots, X_k]$, s) iff among($[X_i, X_{i+1}, \ldots, X_{i+q-1}]$, s, l, u) for $1 \leq i \leq k-q+1$
  - sequence(1,2,3,[1,0,2,0,3],{0,1})
- Useful e.g. in:
  - rostering (e.g. every employee has 2 days off in any 7 day of period);
  - production line (e.g. at most 1 in 3 cars along the production line can have a sun-roof fitted).

Scheduling Constraints

- Help schedule tasks with respective release times, duration, and deadlines, using limited resources in a time interval.

Disjunctive Resource Constraint

- Requires that tasks do not overlap in time.
  - Known also as noOverlap constraint.
- Given tasks $t_1, \ldots, t_k$, each associated with a start time S_i and duration D_i:

  disjunctive($[S_1, \ldots, S_k]$,$[D_1, \ldots, D_k]$) iff for all $i < j$

  $$(S_i + D_i \leq S_j) \vee (S_j + D_j \leq S_i)$$

- Useful when a resource can execute at most one task at a time.

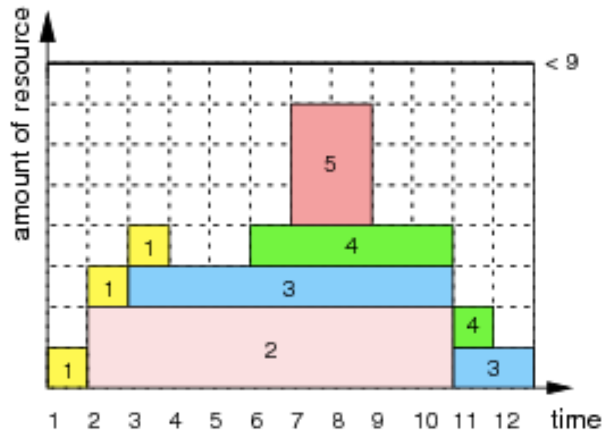
Cumulative Resource Constraint

- Constrains the usage of a shared resource.
- Given tasks $t_1, \ldots, t_k$, each associated with a start time S_i, duration D_i, resource requirement R_i, and a resource with a capacity C:

  cumulative($[S_1, \ldots, S_k]$, $[D_1, \ldots, D_k]$, $[R_1, \ldots, R_k]$, C) iff

  $\sum_{i|S_i \le u < S_i + D_i} R_i \le C$ forall u in D

- Useful when a resource with a capacity can execute multiple tasks at a time.



Ordering Constraints

- Enforce an ordering between the variables or the values.

Lexicographic Ordering Constraint

- Requires a sequence of variables to be lexicographically less than or equal to another sequence of variables.
- lex≤($[X_1, X_2, \ldots, X_k]$, $[Y_1, Y_2, \ldots, Y_k]$) holds iff:

$$X_1 \leq Y_1 \wedge$$
$$(X_1 = Y_1 \rightarrow X_2 \leq Y_2) \wedge$$
$$(X_1 = Y_1 \wedge X_2 = Y_2 \rightarrow X_3 \leq Y_3) \ldots$$
$$(X_1 = Y_1 \wedge X_2 = Y_2 \ldots. X_{k-1} = Y_{k-1} \rightarrow X_k \leq Y_k)$$

  - lex≤([1, 2, 4],[1, 3, 3])
- Useful in symmetry breaking.
  - Avoid permutations of (groups of) variables.

Permutation of Variables

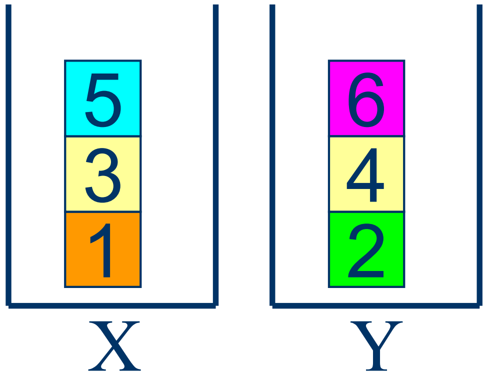
- lex≤([X_1, X_2, …, X_k] , π([X_1, X_2, …, X_k])) for some π.
- E.g., with n-Queens:

```
constraint
    lex_lesseq(array1d(qb), [ qb[j,i] | i,j in 1..n ])
/\  lex_lesseq(array1d(qb), [ qb[i,j] | i in reverse(1..n), j in 1..n ])
/\  lex_lesseq(array1d(qb), [ qb[j,i] | i in 1..n, j in reverse(1..n) ])
/\  lex_lesseq(array1d(qb), [ qb[i,j] | i in 1..n, j in reverse(1..n) ])
/\  lex_lesseq(array1d(qb), [ qb[j,i] | i in reverse(1..n), j in 1..n ])
/\  lex_lesseq(array1d(qb), [ qb[i,j] | i,j in reverse(1..n) ])
/\  lex_lesseq(array1d(qb), [ qb[j,i] | i,j in reverse(1..n) ])
;
```

Permutation of Two Sequences of Variables

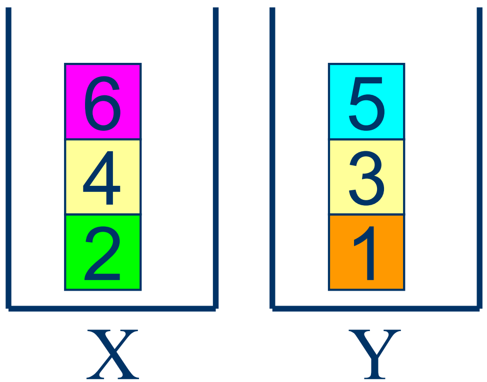
- Assignments of items to two identical bins can be represented by a matrix of Boolean variables:

a)



	i_1	i_2	i_3	i_4	i_5	i_6
X	1	0	1	0	1	0
Y	0	1	0	1	0	1

b)

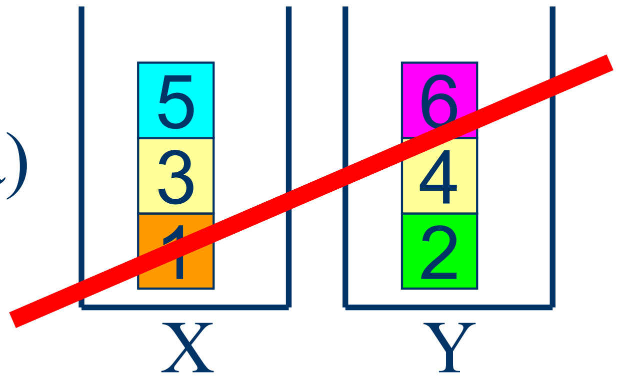


	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0

Permutation of Two Sequences of Variables

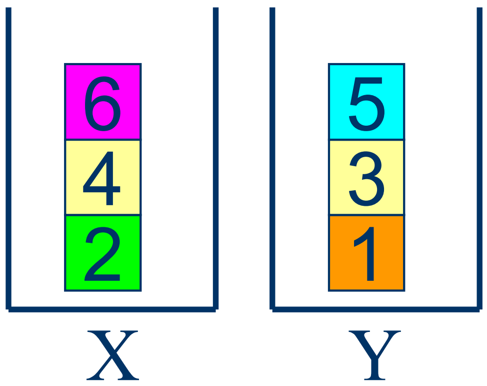
- Need to avoid the symmetric assignments.

a)



	i_1	i_2	i_3	i_4	i_5	i_6
X	1	0	1	0	1	0
Y	0	1	0	1	0	1

b)

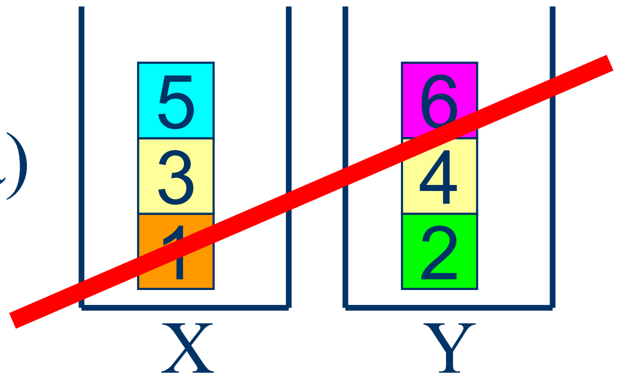


	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0

Permutation of Two Sequences of Variables

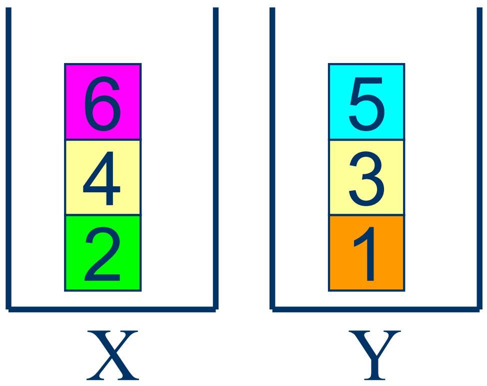
- lex ≤(X , Y).

a)



	i_1	i_2	i_3	i_4	i_5	i_6
X	1	0	1	0	1	0
Y	0	1	0	1	0	1

b)

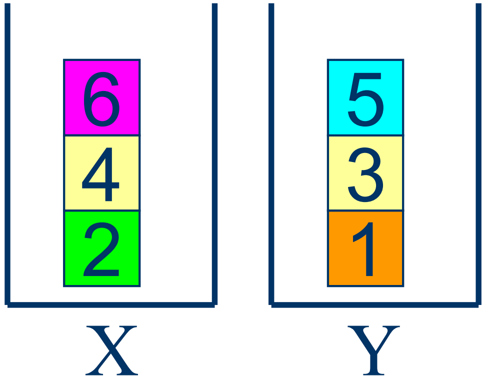


	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0

Permutation of Two Sequences of Variables

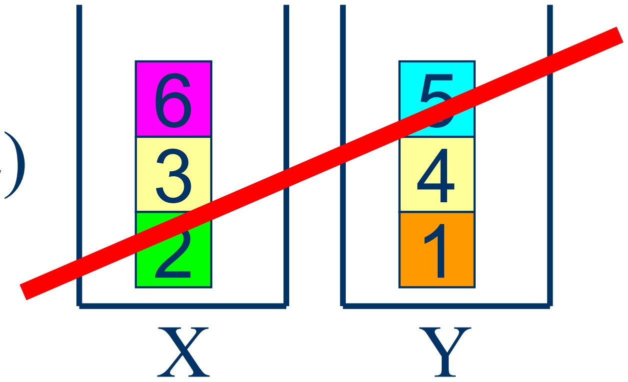
- Need to avoid the symmetric assignments.

b)



	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0

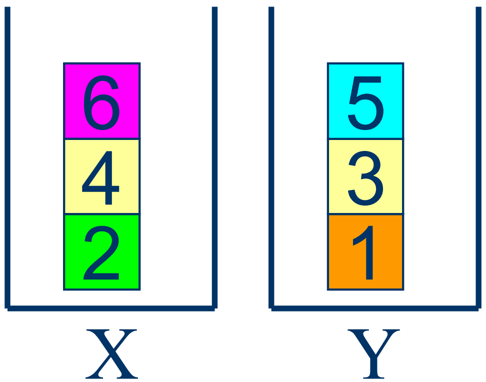
c)



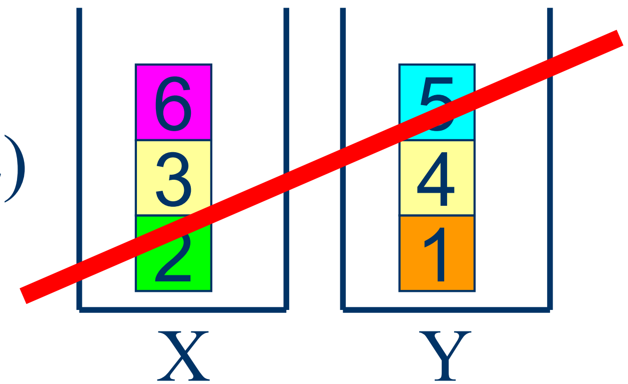
	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	1	0	0	1
Y	1	0	0	1	1	0

Permutation of Two Sequences of Variables

- lex ≤(i_3, i_4).

b) 

	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0

c) 

	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	1	0	0	1
Y	1	0	0	1	1	0

Value Precedence Constraint

- Requires a value to precede another value in a sequence of variables.
- value_precede(v_{j1}, v_{j2}, [X_1, X_2, …, X_k]) holds iff:
  - $\min\{ i \mid X_i = v_{j1} \lor i = k+1\} < \min\{ i \mid X_i = v_{j2} \lor i = k + 2\}$.
  - value_precede(5, 4, [2, 5, 3, 5, 4])
- Useful in symmetry breaking.
  - Avoid permutations of values.

Specialized Propagation for Global Constraints

- How do we develop specialized propagation for global constraints?
- Two main approaches:
  - constraint decomposition;
  - dedicated ad-hoc algorithm.

Constraint Decomposition

- A global constraint is decomposed into smaller and simpler constraints, each of which has a known propagation algorithm.
- Propagating each of the constraints gives a propagation algorithm for the original global constraint.
  - A very effective and efficient method for some global constraints.

A Decomposition of Among

- among($[X_1, X_2, \ldots, X_k]$, s, N)
- Decomposition as a conjunction of logical constrains and a sum constraint.
  - B_i with $D(B_i) = \{0, 1\}$ for $1 \leq i \leq k$
  - C_i: $B_i = 1 \leftrightarrow X_i \in s$ for $1 \leq i \leq k$
  - C_{k+1}: $\sum_i B_i = N$
- AC(C_i) for all i and BC($\sum_i B_i = N$) ensures GAC on among.

A Decomposition of Lex

- lex $\leq([X_1, X_2, \ldots, X_k], [Y_1, Y_2, \ldots, Y_k])$
- Decomposition as a conjunction of disjunctions.
  - B_i with $D(B_i) = \{0, 1\}$ for $1 \leq i \leq k+1$ to indicate the vectors have been ordered by position i-1.
  - $B_1 = 0$
  - C_i: $(B_i = B_{i+1} = 0$ AND $X_i = Y_i)$ OR $(B_i = 0$ AND $B_{i+1} = 1$ AND $X_i < Y_i)$ OR $(B_i = B_{i+1} = 1)$ for $1 \leq i \leq k$
- GAC(C_i) for all i ensures GAC on lex $\leq$.

Constraint Decompositions

- May not always provide an effective propagation.
- Often GAC on the original constraint is stronger than (G)AC on the constraints in the decomposition.

A Decomposition of Alldifferent

- alldifferent($[X_1, X_2, \ldots, X_k]$)
- Decomposition as a conjunction of difference constraints.
  - C_{ij}: $X_i \neq X_j$ for $i < j \in \{1,\ldots,k\}$
- AC(C_{ij}) for all $i < j$ is weaker than GAC on alldifferent.
  - E.g., alldifferent($[X_1, X_2, X_3]$) with $D(X_1) = D(X_2) = D(X_3) = \{1,2\}$.
  - alldifferent is not GAC but the decomposition does not prune anything.

A Decomposition of Sequence

- sequence(l, u, q, [X_1, X_2, …, X_k], s)
- Decomposition as a conjunction of among constraints.
  - C_i: among([X_i, X_{i+1}, …, X_{i+q-1}], s, l, u) for $1 \leq i \leq k-q+1$
- GAC(C_i) for all i is weaker than GAC on sequence.
  - E.g., sequence(2, 3, 5, [X_1, X_2, …, X_7], {1}) with $X_1 = X_2 = 1$, $X_6 = 0$, $D(X_i) = \{0,1\}$ for $i \in \{3,4,5,7\}$.
  - sequence is not GAC but the decomposition does not prune anything.

A Decomposition of Sequence

- 1 1 {0,1} {0,1} {0,1} 0 {0,1} q=5, l =2, u =3,v={1}
- 1 1 {0,1} {0,1} {0,1} 0 {0,1} GAC(among)
- 1 1 {0,1} {0,1} {0,1} 0 {0,1} GAC(among)
- 1 1 {0,1} {0,1} {0,1} 0 {0,1} GAC(among)

A Decomposition of Sequence

- $1\ \ 1\ \{0,1\}\ \{0,1\}\ \{0,1\}\ 0\ \{0,1\}$ $q=5,\ l=2,\ u=3, v=\{1\}$
- $\boxed{1\ \ 1\ \{0,1\}\ \{0,1\}\ \{0,1\}}\ 0\ \{0,1\}$ GAC(among)
- $1\ \boxed{1\ \{0,1\}\ \{0,1\}\ \{0,1\}\ 0}\ \{0,1\}$ GAC(among)
- $1\ \ 1\ \boxed{\{0,1\}\ \{0,1\}\ \{0,1\}\ 0\ \{\cancel{0},1\}}$ GAC(among)

A Decomposition of Lex

- lex $\leq([X_1, X_2, \ldots, X_k], [Y_1, Y_2, \ldots, Y_k])$
- Decomposition as a conjunction of implications
  - $X_1 \leq Y_1$ AND $(X_1 = Y_1 \rightarrow X_2 \leq Y_2)$ AND …

    $(X_1 = Y_1$ AND $X_2 = Y_2$ AND …. $X_{k-1} = Y_{k-1} \rightarrow X_k \leq Y_k)$
- AC on the decomposition is weaker than GAC on lex≤.
  - E.g., lex $\leq([X_1, X_2], [Y_1, Y_2])$ with $D(X_1) = \{0,1\}$, $X_2 = 1$,
    $D(Y_1) = \{0,1\}$, $Y_2 = 0$
  - lex ≤ is not GAC but the decomposition does not prune anything.

Decomposition vs Ad-hoc Algorithm

- Even if a decomposition is effective, may not always provide an efficient propagation.
- Often propagating a constraint via an ad-hoc algorithm is faster than propagating the (many) constraints in the decomposition.
  - Thanks to incremental computation!

Incremental Computation

- A propagation algorithm is often called multiple times.
  - We don't want to re-compute everything each time.
- Incremental computation can improve efficiency.
  - At the first call, some partial results are cached.
  - At the next invoke, we exploit the cached data.
- This requires access to more details about propagation:
  - which variable has been pruned?
  - which values have been pruned?

Dedicated BC Algorithm for Sum

- C: $\sum_i X_i = N$ where X_i and N are integer variables.
  - min(N) ≥ $\sum_i \min(X_i)$
  - max(N) ≤ $\sum_i \max(X_i)$
  - $\min(X_i)$ ≥ min(N) - $\sum_{j \neq i} \max(X_j)$ for 1 ≤ i ≤ n
  - $\max(X_i)$ ≤ max(N) - $\sum_{j \neq i} \min(X_j)$ for 1 ≤ i ≤ n

BC Decomposition for Sum

- C: $\sum_i X_i = N$ where X_i and N are integer variables.
  - $X_1 + X_2 = Y_1$
  - $Y_1 + X_3 = Y_2$
  - ...
  - $Y_{(n-1)} + X_n = N$

Filtering min(N)

- C: $\sum_i X_i = N$ where X_i and N are integer variables.
  - $\min(X_1) + \min(X_2) \le \min(Y_1)$
  - $\min(Y_1) + \min(X_3) \le \min(Y_2)$
  - ...
  - $\min(Y_{(n-1)}) + \min(X_n) \le \min(N)$

  which is equivalent to

  $$\sum_i \min(X_i) \le \min(N)$$

Number of Operations

- C: $\sum_i X_i = N$ where X_i and N are integer variables.
  - $\min(X_1) + \min(X_2) \le \min(Y_1)$
  - $\min(Y_1) + \min(X_3) \le \min(Y_2)$
  - ...
  - $\min(Y_{(n-1)}) + \min(X_n) \le \min(N)$

Read access: 2(n-1)
Write access: n-1
Sum: n-1

$$\sum_i \min(X_i) \le \min(N)$$

Read access: n
Write access: 1
Sum: n-1

Number of Operations

- C: $\sum_i X_i = N$ where X_i and N are integer variables.
  - $\max(X_1) + \max(X_2) \geq \max(Y_1)$
  - $\max(Y_1) + \max(X_3) \geq \max(Y_2)$
  - ...
  - $\max(Y_{(n-1)}) + \max(X_n) \geq \max(N)$

Read access: 2(n-1)
Write access: n-1
Sum: n-1

$$\sum_i \max(X_i) \geq \max(N)$$

Read access: n
Write access: 1
Sum: n-1

Incremental Computation

- C: $\sum_i X_i = N$ where X_i and N are integer variables.
  - max(N) ≤ $\sum_i \max(X_i)$
    - Cache max(N) as max$(N)
    - Whenever the bounds of a variable X_i is pruned:
      - max(N) ≤ max$(N) – (old(max($X_i$)) - max($X_i$))   O(1)

Incremental Computation

- C: $\sum_i X_i = N$ where X_i and N are integer variables.
  - Complexity reduces to O(1) from O(n)

Classical Sum	Incremental Sum
Read access: n	Read access: 3
Write access: 1	Write access: 1
Sum: n-1	Sum: 2

Dedicated Propagation Algorithms

- Dedicated ad-hoc algorithms provide effective and efficient propagation.
- Often:
  - GAC is maintained in polynomial time;
  - many more inconsistent values are detected compared to the decompositions;
  - computation is done incrementally.