Topic 16: Propagators

(Version of 13th November 2020)

Pierre Flener

Optimisation Group
Department of Information Technology
Uppsala University
Sweden

Course 1DL441:
Combinatorial Optimisation and Constraint Programming,
whose part 1 is Course 1DL451:
Modelling for Combinatorial Optimisation

1Based partly on material by N. Beldiceanu and Ch. Schulte
Outline

1. Reification
2. Global Constraints
3. linear
4. channel
5. element
6. extensional
7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
Outline

1. Reification
2. Global Constraints
3. linear
4. channel
5. element
6. extensional
7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
Reification

Implementation of \( b \iff \gamma(\cdots) \):

When there are search guesses or other constraints on the reifying 0/1-variable \( b \):

- If the variable \( b \) becomes fixed to 1, then the constraint \( \gamma(\cdots) \) is propagated.

- If the variable \( b \) becomes fixed to 0, then the constraint \( \neg\gamma(\cdots) \) is propagated.

- If the constraint \( \gamma(\cdots) \) is subsumed, then the variable \( b \) is fixed to 1.

- If the constraint \( \neg\gamma(\cdots) \) is subsumed, then the variable \( b \) is fixed to 0.
Constraint combination with reification:
With reification, constraints can be arbitrarily combined with logical connectives: negation ($\neg$), disjunction ($\lor$), conjunction ($\land$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$). However, propagation may be very poor!

Example
The composite constraint $$(\gamma_1 \land \gamma_2) \lor \gamma_3$$ is modelled as

$$(b_1 \Leftrightarrow \gamma_1) \land (b_2 \Leftrightarrow \gamma_2) \land (b_3 \Leftrightarrow \gamma_3)$$

$$\land (b_1 \cdot b_2 = b) \land (b + b_3 \geq 1)$$

Hence even the constraints $\gamma_1$ and $\gamma_2$ must be reified. If $\gamma_1$ is $x = y + 1$ and $\gamma_2$ is $y = x + 1$, then $\gamma_1 \land \gamma_2$ is unsat; however, $b$ is then not fixed to value 0 by propagation, as each propagator works individually and there is no communication through the shared variables $x$ and $y$; hence $b_3 = 1$ is not propagated and $\gamma_3$ is not forced to hold.
Remember the warning in Topic 2: Basic Modelling that the disjunction and negation of constraints (with \/, xor, not, <-, ->, <->, exists, xorall, if \( \theta \) then \( \phi \) else \( \psi \) endif) in MiniZinc often makes the solving slow?

**Example**

The MiniZinc disjunctive constraint

```plaintext
constraint x = 0 \/ x = 9;
```

is flattened for Gecode as follows, with reification:

\[
(b_0 \Leftrightarrow x = 0) \& (b_9 \Leftrightarrow x = 9) \& (b_0 + b_9 \geq 1)
\]

But it is logically equivalent to the variable declaration

```plaintext
var \{0, 9\}: x;
```

where no reification is involved, and even no propagation.
Remember the strong warning in Topic 2: Basic Modelling about a conditional \( \text{if } \theta \text{ then } \phi_1 \text{ else } \phi_2 \text{ endif} \) or a comprehension, say \([i \mid i \in \rho \text{ where } \theta]\), in MiniZinc having a test \( \theta \) that depends on variables?

**Example**

Consider \( \text{var } 1..9: x \) and \( \text{var } 1..9: y \) for

\[
\text{forall}(i \text{ in } 1..9 \text{ where } i > x)(i > y)
\]

Recall that this is syntactic sugar for

\[
\text{forall}([i > y \mid i \text{ in } 1..9 \text{ where } i > x])
\]

This is flattened for Gecode into the equivalent of

\[
\text{forall}(i \text{ in } 1..9)(i > x \rightarrow i > y)
\]

that is with a logical implication \((-\rightarrow)\), hence with a hidden logical disjunction \( (\lor) \): for each \( i \), both sub-constraints are **reified** as both have variables.
Outline

1. Reification

2. Global Constraints

3. linear

4. channel

5. element

6. extensional

7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
Definition

- A primitive constraint is not decomposable.
- A global constraint is definable by a logical formula (usually a conjunction) involving primitive constraints, but not always in a trivial way.

For domain consistency, all solutions to a constraint need to be considered: a naïve propagator, first computing all the solutions and then projecting them onto the domains of the variables, often takes too much time and space:

Example (already seen in Topic 13: Consistency)

The store \( \{ x \mapsto \{2, \ldots, 7\}, y \mapsto \{0, 1, 2\}, z \mapsto \{-1, \ldots, 2\} \} \) has the solutions \( \langle 3, 1, 0 \rangle, \langle 5, 0, 1 \rangle, \text{ and } \langle 6, 2, 0 \rangle \) to the linear equality constraint \( x = 3 \cdot y + 5 \cdot z \).

Hence the store \( \{ x \mapsto \{3, 5, 6\}, y \mapsto \{0, 1, 2\}, z \mapsto \{0, 1\} \} \) is domain-consistent. (Continued on slide 18.)
Globality from a Semantic Point of View

Some constraints cannot be defined by a conjunction of primitive constraints without introducing more variables:

**Example** \(\text{count}([x_1, \ldots, x_n], v, \geq, \ell)\)

At least \(\ell\) variables of \([x_1, \ldots, x_n]\) take the constant value \(v\):

\[
(\forall i \in 1..n : b_i \iff x_i = v) \land \sum_{i=1}^{n} b_i \geq \ell
\]

Some constraints can be defined by a conjunction of primitive constraints without introducing more variables:

**Example** \(\text{distinct}([x_1, \ldots, x_n])\)

\(\forall i, j \in 1..n \text{ where } i < j : x_i \neq x_j\)
Some constraints can be defined by a conjunction of primitive constraints, but it leads to weak propagation:

Example

Consider the store \{x_1, x_2, x_3 \mapsto \{4, 5\}\}:

- Upon `distinct([x_1, x_2, x_3])`:
  Propagation fails under domain or bounds consistency.

- Upon `x_1 \neq x_2 \land x_1 \neq x_3 \land x_2 \neq x_3`:
  Propagation succeeds, and it is only search that fails.
Globality from a Propagation Point of View

Some constraints can be defined by a conjunction of primitive constraints, with strong propagation, but it leads to propagation with poor time or memory performance:

Example

- Upon `strictly_increasing([a,b,c,d,a])`, which is `rel([a,b,c,d,a],IRT_LE)` in Gecode: Propagation fails.
- Upon `a < b & b < c & c < d & d < a`: Propagation also fails, but the runtime complexity depends on the sizes of the domains, rather than on the number of variables.
Outline

1. Reification
2. Global Constraints
3. linear
4. channel
5. element
6. extensional
7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
The linear Predicate

Definition

A linear \([a_1, \ldots, a_n] , [x_1, \ldots, x_n] , R , d\) constraint, with

- \([a_1, \ldots, a_n]\) a sequence of non-zero integer constants,
- \([x_1, \ldots, x_n]\) a sequence of integer variables,
- \(R\) in \(\{<, \leq, =, \neq, \geq, >\}\), and
- \(d\) an integer constant,

holds iff the linear relation \(\sum_{i=1}^{n} a_i \cdot x_i \) \(R\) \(d\) holds.

We now show how to enforce bounds consistency cheaply on linear equality. For simplicity of notation, we pick \(n = 2\), giving \(a_1 \cdot x_1 + a_2 \cdot x_2 = d\), and rename into \(a \cdot x + b \cdot y = d\).
**BC propagator for a binary linear equality:**

Rewrite for $x$ (the handling of $y$ is analogous and omitted):

$$a \cdot x + b \cdot y = d \iff x = (d - b \cdot y) / a$$

Upper bound on $x$, starting from store $s$:

$$x \leq \max \left\{ \frac{(d - b \cdot n)}{a} \mid n \in s(y) \right\}$$

and (analogously, hence further details are omitted):

$$x \geq \left[ \min \left\{ \frac{(d - b \cdot n)}{a} \mid n \in s(y) \right\} \right]$$

Computing $M$:

$$M = \begin{cases} \max \left\{ \frac{(d - b \cdot n)}{a} \mid n \in s(y) \right\} / a & \text{if } a > 0 \\ \min \left\{ \frac{(d - b \cdot n)}{a} \mid n \in s(y) \right\} / a & \text{if } a < 0 \end{cases}$$
BC propagator for a binary linear equality (end):
For \( a > 0 \) (the case \( a < 0 \) is analogous and omitted):

\[
M = \max \{ (d - b \cdot n) \mid n \in s(y) \} / a
\]
\[
= \left( d - \min \{ b \cdot n \mid n \in s(y) \} \right) / a
\]
\[
= \begin{cases} 
(d - b \cdot \min(s(y))) / a & \text{if } b > 0 \\
(d - b \cdot \max(s(y))) / a & \text{if } b < 0 
\end{cases}
\]

This value can be computed and rounded in constant time, since the constants \( \min(s(y)) \) and \( \max(s(y)) \) can be queried in constant time and since \( a, b, d \) are constants.
BC propagator for $n$-ary linear equality, with $n \geq 1$:

Iterate until fixpoint, to achieve idempotency if wanted:

- propagate for each variable $x_i$.

A speed-up can be obtained by computing two general expressions once and then adjusting them for each $x_i$:

\[ \text{see § 6.4 of Krzysztof R. Apt, Principles of Constraint Programming, Cambridge University Press, 2003.} \]

**Example (Justification for aiming at idempotency)**

Propagate $2 \cdot x = 3 \cdot y$ for \( \{x \mapsto \{0, \ldots, 8\}, y \mapsto \{0, \ldots, 9\}\} \).

- Propagating for $x$ gives: \( \{x \mapsto \{0, \ldots, 8\}, y \mapsto \{0, \ldots, 9\}\} \)

- Propagating for $y$ gives: \( \{x \mapsto \{0, \ldots, 8\}, y \mapsto \{0, \ldots, 5\}\} \)

Four values were deleted from $\text{dom}(y)$ without failing to find supports, but the bound 8 of $x$ is no longer supported!

- Propagating for $x$ gives: \( \{x \mapsto \{0, \ldots, 7\}, y \mapsto \{0, \ldots, 5\}\} \)

- Propagating for $y$ gives: \( \{x \mapsto \{0, \ldots, 7\}, y \mapsto \{0, \ldots, 4\}\} \)

- Propagating for $x$ gives: \( \{x \mapsto \{0, \ldots, 6\}, y \mapsto \{0, \ldots, 4\}\} \)

- Propagating for $y$ gives: \( \{x \mapsto \{0, \ldots, 6\}, y \mapsto \{0, \ldots, 4\}\} \)
Consistency on $n$-ary linear constraints:

- **Linear equality ($=$):** The described propagator enforces $\text{BC}(\mathbb{R})$ in $O(n)$ time per iteration, but enforcing $\text{DC}$ is NP-hard (so it currently takes time exponential in $n$).

### Example (Why $\text{BC}(\mathbb{R})$ and not $\text{BC}(\mathbb{Z} / \mathbb{D})$ for equality?)

Propagate $x = 3 \cdot y + 5 \cdot z$ from the store

$$
\{ x \mapsto \{2, \ldots, 7\}, \ y \mapsto \{0, 1, 2\}, \ z \mapsto \{0, 1\} \}.
$$

The described bounds($\mathbb{R}$) propagator gives

$$
\{ x \mapsto \{2, \ldots, 7\}, \ y \mapsto \{0, 1, 2\}, \ z \mapsto \{0, 1\} \},
$$

while a bounds($\mathbb{Z}$) or bounds($\mathbb{D}$) propagator would give

$$
\{ x \mapsto \{3, \ldots, 6\}, \ y \mapsto \{0, 1, 2\}, \ z \mapsto \{0, 1\} \}.
$$

The described propagator considers **real-number supports**, even though the constraint is over **integer** variables. Compare with the domain-consistent store on slide 9.

- **Linear disequality ($\neq$):** $\text{BC}(\cdot) = \text{DC}; O(n)$ time.
- **Linear inequality ($<, \leq, \geq, >$):** $\text{BC}(\mathbb{R}) = \text{DC}; O(n)$ time.
Outline

1. Reification
2. Global Constraints
3. linear
4. channel
5. element
6. extensional
7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
The channel Predicate

Definition

A channel([x₁, ..., xₙ], [y₁, ..., yₙ]) constraint holds iff:

\[ \forall i, j \in 1..n : x_i = j \iff y_j = i \]

Propagator for domain consistency:

- For each \( i \notin \text{dom}(y_j) \): delete \( j \) from \( \text{dom}(x_i) \).
- For each \( j \notin \text{dom}(x_i) \): delete \( i \) from \( \text{dom}(y_j) \).
- Post distinct([x₁, ..., xₙ]) as implied constraint: if \( x_a = j = x_b \) with \( a \neq b \), then \( y_j \) has to take two distinct values, namely \( a \) and \( b \), which is impossible.
- Posting also distinct([y₁, ..., yₙ]) as implied constraint would bring no further propagation.
Outline

1. Reification
2. Global Constraints
3. linear
4. channel
5. element
6. extensional
7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
The element Predicate

Definition (Van Hentenryck and Carillon, 1988)

An \( \text{element}([x_1, \ldots, x_n], i, e) \) constraint, where the \( x_j \) are variables, \( i \) is an integer variable, and \( e \) is a variable, holds if and only if \( x_i = e \).

Example

From the store \( \{i \mapsto \{1, 2, 3, 4\}, e \mapsto \{7, 8, 9\}\} \), the constraint \( \text{element}([6, 8, 7, 8], i, e) \) propagates under DC to fixpoint \( \{i \mapsto \{2, 3, 4\}, e \mapsto \{7, 8\}\} \). If the domain of \( i \) is pruned to \( \{2, 4\} \) by another constraint or a search guess, then \( e \mapsto \{8\} \) and subsumption are inferred under DC.

Possible definition of \( \text{element}([x_1, \ldots, x_n], i, e) \):

\( (i = 1 \Rightarrow x_1 = e) \ & \cdots \ & (i = n \Rightarrow x_n = e) \), with implicative constraints \( \alpha(\cdots) \Rightarrow \beta(\cdots) \) definable, under little propagation, by \( a \Leftrightarrow \alpha(\cdots) \ & b \Leftrightarrow \beta(\cdots) \ & a \leq b \).
Propagation on an array of constants:
We insist on domain consistency, as BC would be too weak. Objective, for element\([x_1, \ldots, x_n], i, e\) and a store \(s\):

- \(i\) Keep only \(k\) in \(s(i)\) such that \(x_k = j\) for some \(j\) in \(s(e)\).
- \(e\) Keep only \(j\) in \(s(e)\) such that \(x_k = j\) for some \(k\) in \(s(i)\).

Naïve DC propagator:
The computed new domains must be ordered sets:

- \(i\) The new domain of \(i\) is \(s(i) \cap \{ k \in 1..n \mid x_k \in s(e) \}\).
- \(e\) The new domain of \(e\) is \(s(e) \cap \{ x_k \mid k \in s(i) \}\).

Sources of inefficiency:
- This **always** iterates over the entire array \([x_1, \ldots, x_n]\).
- This **always** requires set intersection.
- This **always** requires sorting the 2nd argument of the 2nd intersection (or performing ordered set insertion).
Example

Consider the constraint $\text{element}([4, 5, 9, 7], i, e)$ and the store $s = \{ i \mapsto \{2, 3, 4\}, e \mapsto \{2, 3, 4, 5, 6, 7, 8\}\}$. Domain consistency gives the store $\{ i \mapsto \{2, 4\}, e \mapsto \{5, 7\}\}$.

Smart DC propagator (Van Hentenryck & Carillon, 1988):
Construct from $[4, 5, 9, 7]$ two ordered doubly-linked lists:

![Diagram of ordered doubly-linked lists]

1. Follow the $i$-links: if a value is not in $s(i)$, then unlink the corresponding two nodes from the two lists.
2. Follow the $e$-links: if a value is not in $s(e)$, then unlink the corresponding two nodes from the two lists.
3. The lists are sorted and are the new domains of $i$ and $e$. 
Consider the constraint $\text{element}([4, 5, 9, 7], i, e)$ and the store $s = \{i \mapsto \{2, 3, 4\}, e \mapsto \{2, 3, 4, 5, 6, 7, 8\}\}$. Domain consistency gives the store $\{i \mapsto \{2, 4\}, e \mapsto \{5, 7\}\}$.

**Smart DC propagator (Van Hentenryck & Carillon, 1988):**

Construct from $[4, 5, 9, 7]$ two ordered doubly-linked lists:

- Follow the $i$-links: if a value is not in $s(i)$, then unlink the corresponding two nodes from the two lists.
- Follow the $e$-links: if a value is not in $s(e)$, then unlink the corresponding two nodes from the two lists.

The lists are sorted and are the new domains of $i$ and $e$. 
Example

Consider the constraint \texttt{element}([4, 5, 9, 7], i, e) and the store \texttt{s} = \{i \mapsto \{2, 3, 4\}, e \mapsto \{2, 3, 4, 5, 6, 7, 8\}\}. Domain consistency gives the store \{i \mapsto \{2, 4\}, e \mapsto \{5, 7\}\}.

**Smart DC propagator (Van Hentenryck & Carillon, 1988):**

Construct from [4, 5, 9, 7] two ordered doubly-linked lists:

```
i  1  2  3  4
  e  4  5  9  7
```

Follow the \texttt{i}\,-links: if a value is not in \texttt{s}(i), then unlink the corresponding two nodes from the two lists.
Example

Consider the constraint `element([4, 5, 9, 7], i, e)` and the store `s = {i ↦→ {2, 3, 4}, e ↦→ {2, 3, 4, 5, 6, 7, 8}}`. Domain consistency gives the store `{i ↦→ {2, 4}, e ↦→ {5, 7}}`.

Smart DC propagator (Van Hentenryck & Carillon, 1988): Construct from `[4, 5, 9, 7]` two ordered doubly-linked lists:

```
        e               i
       / \               / \               / \               / \
      4   5   9   7     1   2   3   4
         \     \   \     \   \   \   \   \   \   \   \   \   \   \   \\  
          \               \               \               \               \
           /               /               /               /               /  \
         4   5   9   7     1   2   3   4
```

- Follow the `i`-links: if a value is not in `s(i)`, then unlink the corresponding two nodes from the two lists.
Example

Consider the constraint $\text{element}([4, 5, 9, 7], i, e)$ and the store $s = \{i \mapsto \{2, 3, 4\}, e \mapsto \{2, 3, 4, 5, 6, 7, 8\}\}$. Domain consistency gives the store $\{i \mapsto \{2, 4\}, e \mapsto \{5, 7\}\}$.

**Smart DC propagator (Van Hentenryck & Carillon, 1988):**

Construct from $[4, 5, 9, 7]$ two ordered doubly-linked lists:

- Follow the $i$-links: if a value is not in $s(i)$, then unlink the corresponding two nodes from the two lists.
- Follow the $e$-links: if a value is not in $s(e)$, then unlink the corresponding two nodes from the two lists.
Example

Consider the constraint \( \text{element}([4, 5, 9, 7], i, e) \) and the store \( s = \{ i \mapsto \{2, 3, 4\}, e \mapsto \{2, 3, 4, 5, 6, 7, 8\}\} \). Domain consistency gives the store \( \{ i \mapsto \{2, 4\}, e \mapsto \{5, 7\}\} \).

**Smart DC propagator (Van Hentenryck & Carillon, 1988):**

Construct from \([4, 5, 9, 7]\) two ordered doubly-linked lists:

- Follow the \( i \)-links: if a value is not in \( s(i) \), then unlink the corresponding two nodes from the two lists.
- Follow the \( e \)-links: if a value is not in \( s(e) \), then unlink the corresponding two nodes from the two lists.
Example

Consider the constraint \texttt{element}([4, 5, 9, 7], i, e) and the store \( s = \{i \mapsto \{2, 3, 4\}, \ e \mapsto \{2, 3, 4, 5, 6, 7, 8\}\} \). Domain consistency gives the store \( \{i \mapsto \{2, 4\}, \ e \mapsto \{5, 7\}\} \).

\textbf{Smart DC propagator (Van Hentenryck & Carillon, 1988):}
Construct from [4, 5, 9, 7] two ordered doubly-linked lists:

```
i
1 --- 2 --- 3 --- 4
```
```
e
4 --- 5 --- 9 --- 7
```

- Follow the \( i \)-links: if a value is not in \( s(i) \), then unlink the corresponding two nodes from the two lists.
- Follow the \( e \)-links: if a value is not in \( s(e) \), then unlink the corresponding two nodes from the two lists.

The lists are sorted and are the new domains of \( i \) and \( e \).
Analysis:

- Each unlinking takes constant time.
- No set intersection needs to be computed.

Definition

An incremental propagator, instead of throwing away an internal data structure when at fixpoint, keeps it for its next invocation: it first repairs that data structure according to the pruning done by other propagators since its previous invocation, and then only attempts its own pruning.

Incremental propagation for element:

- This requires sorting only at the first invocation, namely of the array (here \([4, 5, 9, 7]\)).
- This always iterates over an array at most as long as at the previous invocation.
Outline

1. Reification
2. Global Constraints
3. linear
4. channel
5. element
6. extensional
7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
Deterministic Finite Automaton (DFA)

Example (DFA for regular expression \(ss(ts)^*|ts(t|ss)^*)\)

Conventions:
- **Start state**, marked by arc coming in from nowhere: A.
- **Accepting states**, marked by double circles: D and E.
- **Determinism**: There is one outgoing arc per symbol in alphabet \(\Sigma = \{s, t\}\); missing arcs go to a non-accepting missing state that has self-loops on every symbol in \(\Sigma\).
The extensional Predicate

Definition

An extensional([x_1, \ldots, x_n], D) constraint holds iff the values taken by the sequence [x_1, \ldots, x_n] of variables form a string of the regular language accepted by the DFA D.

Example

The constraint extensional([x_1, x_2, x_3, x_4], A), where A is the DFA of the previous slide, is propagated under domain consistency from the store

\[
\{ \ x_1 \mapsto \{s, t\}, \ x_2 \mapsto \{s, t\}, \ x_3 \mapsto \{s, t\}, \ x_4 \mapsto \{s, t\} \ \}
\]

to the fixpoint

\[
\{ \ x_1 \mapsto \{s, t\}, \ x_2 \mapsto \{s\}, \ x_3 \mapsto \{s, t\}, \ x_4 \mapsto \{s, t\} \ \}
\]
Let us propagate \textit{extensional}([x_1, x_2, x_3, x_4], \mathcal{A}), where \mathcal{A} is the DFA of two slides ago, from the following store:

\[
\begin{align*}
    x_1 &\mapsto \{s, t\} \\
    x_2 &\mapsto \{s, t\} \\
    x_3 &\mapsto \{s, t\} \\
    x_4 &\mapsto \{s, t\}
\end{align*}
\]
Efficient DC Propagator (Pesant, 2004)

**Forward Phase:** Build all paths according to the values in the domains.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]

A0
Efficient DC Propagator (Pesant, 2004)

**Forward Phase:** Build all paths according to the values in the domains.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]

![Diagram](image)
Efficient DC Propagator (Pesant, 2004)

**Forward Phase:** Build all paths according to the values in the domains.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]

![Graph Diagram]
Efficient DC Propagator (Pesant, 2004)

**Forward Phase:** Build all paths according to the values in the domains.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]
Efficient DC Propagator (Pesant, 2004)

**Forward Phase:** Build all paths according to the values in the domains.

\[
\begin{align*}
    x_1 &\mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}
\end{align*}
\]

![Diagram showing paths and variables]

\[
\begin{align*}
    A0 &\rightarrow [s] \quad B1 &\rightarrow [s] \quad D2 \\
    C1 &\rightarrow [t] \quad E2 &\rightarrow [s]
\end{align*}
\]
Efficient DC Propagator (Pesant, 2004)

Forward Phase: Build all paths according to the values in the domains.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]

![Graph](image-url)
Efficient DC Propagator (Pesant, 2004)

**Forward Phase:** Build all paths according to the values in the domains.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]
Efficient DC Propagator (Pesant, 2004)

**Forward Phase:** Build all paths according to the values in the domains.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]

![Diagram of the Efficient DC Propagator](attachment://diagram.png)
Efficient DC Propagator (Pesant, 2004)

**Forward Phase:** Build all paths according to the values in the domains.

\[
x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}
\]
Efficient DC Propagator (Pesant, 2004)

**Forward Phase:** Build all paths according to the values in the domains.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]
Efficient DC Propagator (Pesant, 2004)

Forward Phase: Build all paths according to the values in the domains.

\[ \begin{align*}
    x_1 & \rightarrow \{s, t\} \\
    x_2 & \rightarrow \{s, t\} \\
    x_3 & \rightarrow \{s, t\} \\
    x_4 & \rightarrow \{s, t\}
\end{align*} \]
Efficient DC Propagator (Pesant, 2004)

Forward Phase: Build all paths according to the values in the domains. (B3 & C3 and D4 & E4 can be merged.)

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]
Efficient DC Propagator (Pesant, 2004)

**Backward Phase:** Delete all paths not of length 4 or not ending in a vertex corresponding to an accepting state.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]
Efficient DC Propagator (Pesant, 2004)

**Backward Phase:** Delete all paths not of length 4 or not ending in a vertex corresponding to an accepting state.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]
Efficient DC Propagator (Pesant, 2004)

Pruning Phase: Delete unsupported values; at fixpoint.

\[
\begin{align*}
x_1 &\mapsto \{s,t\} & x_2 &\mapsto \{s,t\} & x_3 &\mapsto \{s,t\} & x_4 &\mapsto \{s,t\}
\end{align*}
\]

Diagram:

- A0 to B1: s
- B1 to D2: s
- D2 to B3: t
- B3 to D4: s
- C1 to E2: s
- E2 to C3: s
- C3 to E4: s
- E3 to E4: t
- E3 to E4: t
- E3 to E4: t
Efficient DC Propagator (Pesant, 2004)

Pruning Phase: Delete unsupported values; at fixpoint.

$x_1 \mapsto \{s, t\}$  $x_2 \mapsto \{s\}$  $x_3 \mapsto \{s, t\}$  $x_4 \mapsto \{s, t\}$
Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_1 = t$ to fixpoint.

\[ x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\} \]
Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_1 = t$ to fixpoint.

$x_1 \mapsto \{t\}$  $x_2 \mapsto \{s\}$  $x_3 \mapsto \{s, t\}$  $x_4 \mapsto \{s, t\}$

Graphical representation: [Diagram with nodes A0, B1, D2, B3, D4, C1, E2, C3, E4, E3, showing the propagation process with arcs labeled with variables $s$ and $t$.]
Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_1 = t$ to fixpoint.

$$x_1 \mapsto \{t\} \quad x_2 \mapsto \{s\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$
Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_3 = s$ to subsumption.

$x_1 \mapsto \{t\} \quad x_2 \mapsto \{s\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$
Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_3 = s$ to subsumption.

$$x_1 \mapsto \{t\} \quad x_2 \mapsto \{s\} \quad x_3 \mapsto \{s\} \quad x_4 \mapsto \{s, t\}$$
Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_3 = s$ to subsumption.

\[ x_1 \mapsto \{t\} \quad x_2 \mapsto \{s\} \quad x_3 \mapsto \{s\} \quad x_4 \mapsto \{s, t\} \]
Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_3 = s$ to subsumption.

$x_1 \mapsto \{t\} \quad x_2 \mapsto \{s\} \quad x_3 \mapsto \{s\} \quad x_4 \mapsto \{s\}$
Complexity and Incrementality

**Complexity:**
The described DC propagator takes $O(n \cdot m \cdot q)$ time and space for $n$ variables, $m$ values in their domains, and $q$ states in the DFA.

**Incrementality via a stateful propagator:**
Keep the graph between propagator invocations. When the propagator is re-invoked:

1. Delete edges that no longer correspond to the store.
2. Run the pruning phase.

**Generalisation:**
The described propagator works unchanged for an NFA (non-deterministic finite automaton): Gecode offers no syntax for this, but MiniZinc has `regular_nfa`. 
Bibliography


Outline

1. Reification
2. Global Constraints
3. linear
4. channel
5. element
6. extensional
7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
The distinct Predicate

Definition (Laurière, 1978)

A \texttt{distinct}([x_1, \ldots, x_n]) constraint holds if and only if all the variables \(x_i\) take different values.

This is equivalent to \(\frac{n \cdot (n-1)}{2}\) disequality constraints:

\[
\forall i, j \in 1..n \text{ where } i < j : x_i \neq x_j
\]

Originally, the \texttt{distinct} constraint was just a wrapper for posting those \(\frac{n \cdot (n-1)}{2}\) disequality constraints. The first efficient domain-consistency propagators for \texttt{distinct} were introduced in 1994; one of them is discussed below. After that, several other efficient propagators have been proposed to enforce various consistencies.
Example

Consider the store \( \{ x_1, x_2, x_3 \mapsto \{4, 5\}\} \) and the constraint \( \text{distinct}([x_1, x_2, x_3]) \):

- Value consistency: Nothing is done to the domains.
- Bounds consistency: A failure is detected.
- Domain consistency (DC): A failure is detected.

What consistency to use is problem-dependent and even instance-dependent!

Example (\( \text{distinct}([u, v, w, x, y, z]) \))

From the store

\[
\begin{align*}
u &\mapsto \{0, 1\}, & v &\mapsto \{1, 2\}, & w &\mapsto \{0, 2\}, \\
x &\mapsto \{1, 3\}, & y &\mapsto \{2, 3, 4, 5\}, & z &\mapsto \{5, 6\}
\end{align*}
\]

the pink values are pruned upon DC.
Is DC Needed for distinct?

Example (Golomb Rulers)

Design a ruler with $n$ ticks such that:

- The distances between any 2 distinct ticks are distinct.
- The length of the ruler is minimal.

For $n = 6$, an optimal ruler is $[0, 1, 4, 10, 12, 17]$.

This very hard problem has applications in crystallography.

<table>
<thead>
<tr>
<th>$n$</th>
<th>value consistency</th>
<th>domain consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>950 nodes</td>
<td>474 nodes</td>
</tr>
<tr>
<td>8</td>
<td>7,622 nodes</td>
<td>3,076 nodes</td>
</tr>
<tr>
<td>9</td>
<td>55,930 nodes</td>
<td>16,608 nodes</td>
</tr>
<tr>
<td>10</td>
<td>413,922 nodes</td>
<td>97,782 nodes</td>
</tr>
<tr>
<td>11</td>
<td>6,330,568 nodes</td>
<td>1,448,666 nodes</td>
</tr>
</tbody>
</table>

Good search-tree reduction: worth looking for a propagator!
Outline

1. Reification
2. Global Constraints
3. linear
4. channel
5. element
6. extensional
7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
Variable-Value Graph:
Construct a bipartite graph from the current domains:

\[ u \mapsto \{0, 1\} \]
\[ v \mapsto \{1, 2\} \]
\[ w \mapsto \{0, 2\} \]
\[ x \mapsto \{1, 3\} \]
\[ y \mapsto \{2, 3, 4, 5\} \]
\[ z \mapsto \{5, 6\} \]
Variable-Value Graph:

A (maximum) matching is a (max-size) subset of edges so that no vertex is incident to two of its edges. Example 1:

- \( u \mapsto \{0, 1\} \)
- \( v \mapsto \{1, 2\} \)
- \( w \mapsto \{0, 2\} \)
- \( x \mapsto \{1, 3\} \)
- \( y \mapsto \{2, 3, 4, 5\} \)
- \( z \mapsto \{5, 6\} \)
Variable-Value Graph:
A (maximum) matching is a (max-size) subset of edges so that no vertex is incident to two of its edges. Example 2:

\[ u \mapsto \{0, 1\} \]
\[ v \mapsto \{1, 2\} \]
\[ w \mapsto \{0, 2\} \]
\[ x \mapsto \{1, 3\} \]
\[ y \mapsto \{2, 3, 4, 5\} \]
\[ z \mapsto \{5, 6\} \]
Variable-Value Graph:

A (maximum) matching is a (max-size) subset of edges so that no vertex is incident to two of its edges. Example 2:

- $u \mapsto \{0, 1\}$
- $v \mapsto \{1, 2\}$
- $w \mapsto \{0, 2\}$
- $x \mapsto \{1, 3\}$
- $y \mapsto \{2, 3, 4, 5\}$
- $z \mapsto \{5, 6\}$

A max matching is (here) perfect iff it covers all variables: it is a solution to the considered distinct constraint.
Naïve DC propagator:

1. If no perfect matching exists, then fail.
2. Compute all perfect matchings and mark their edges.
3. For every unmarked edge between a variable $v$ and a value $d$: prune value $d$ from $\text{dom}(v)$.

But there are as many perfect matchings as solutions!

We have not addressed the time issue.

Matching theory to the rescue!
There is a relationship between the edges in a maximum matching and the edges in all other maximum matchings!

Hence we need only compute one perfect matching!
Outline

1. Reification
2. Global Constraints
3. Linear
4. Channel
5. Element
6. Extensional
7. Distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
Efficient DC propagator (Régin, 1994) (Costa, 1994):
Start from a perfect matching, and orient all edges: if in matching, then from variable to value, else the other way.

$u \mapsto \{0, 1\}$

$v \mapsto \{1, 2\}$

$w \mapsto \{0, 2\}$

$x \mapsto \{1, 3\}$

$y \mapsto \{2, 3, 4, 5\}$

$z \mapsto \{5, 6\}$
Efficient DC propagator (Régisn, 1994) (Costa, 1994):
Start from a perfect matching, and orient all edges: if in matching, then from variable to value, else the other way.

\[ u \mapsto \{0, 1\} \]
\[ v \mapsto \{1, 2\} \]
\[ w \mapsto \{0, 2\} \]
\[ x \mapsto \{1, 3\} \]
\[ y \mapsto \{2, 3, 4, 5\} \]
\[ z \mapsto \{5, 6\} \]
Efficient DC propagator (Régin, 1994) (Costa, 1994):
Start from all unmatched vertices (necessarily values here) and mark all arcs on all simple paths: arcs can be flipped.

\[ u \mapsto \{0, 1\} \quad v \mapsto \{1, 2\} \quad w \mapsto \{0, 2\} \quad x \mapsto \{1, 3\} \quad y \mapsto \{2, 3, 4, 5\} \quad z \mapsto \{5, 6\} \]
Efficient DC propagator (Régis, 1994) (Costa, 1994): Start from all unmatched vertices (necessarily values here) and mark all arcs on all simple paths: arcs can be flipped.

- $u \mapsto \{0, 1\}$
- $v \mapsto \{1, 2\}$
- $w \mapsto \{0, 2\}$
- $x \mapsto \{1, 3\}$
- $y \mapsto \{2, 3, 4, 5\}$
- $z \mapsto \{5, 6\}$
Efficient DC propagator (Régin, 1994) (Costa, 1994): Start from all unmatched vertices (necessarily values here) and mark all arcs on all simple paths: arcs can be flipped.

\[
\begin{align*}
&u \mapsto \{0, 1\} \\
&v \mapsto \{1, 2\} \\
&w \mapsto \{0, 2\} \\
&x \mapsto \{1, 3\} \\
&y \mapsto \{2, 3, 4, 5\} \\
&z \mapsto \{5, 6\}
\end{align*}
\]
Efficient DC propagator (Régine, 1994) (Costa, 1994):
Start from all unmatched vertices (necessarily values here) and mark all arcs on all simple paths: arcs can be flipped.

\[ u \mapsto \{0, 1\} \]

\[ v \mapsto \{1, 2\} \]

\[ w \mapsto \{0, 2\} \]

\[ x \mapsto \{1, 3\} \]

\[ y \mapsto \{2, 3, 4, 5\} \]

\[ z \mapsto \{5, 6\} \]
Efficient DC propagator (Régisn, 1994) (Costa, 1994): Start from all unmatched vertices (necessarily values here) and mark all arcs on all simple paths: arcs can be flipped.

- \( u \mapsto \{0, 1\} \)
- \( v \mapsto \{1, 2\} \)
- \( w \mapsto \{0, 2\} \)
- \( x \mapsto \{1, 3\} \)
- \( y \mapsto \{2, 3, 4, 5\} \)
- \( z \mapsto \{5, 6\} \)
Efficient DC propagator (Régin, 1994) (Costa, 1994): Mark all arcs in all strongly connected components (SCCs): the variables of an SCC take all the values of that SCC.

\[ u \mapsto \{0, 1\} \]
\[ v \mapsto \{1, 2\} \]
\[ w \mapsto \{0, 2\} \]
\[ x \mapsto \{1, 3\} \]
\[ y \mapsto \{2, 3, 4, 5\} \]
\[ z \mapsto \{5, 6\} \]
**Efficient DC propagator (Régin, 1994) (Costa, 1994):**
Mark all arcs in all strongly connected components (SCCs): the variables of an SCC take all the values of that SCC.

- $u \mapsto \{0, 1\}$
- $v \mapsto \{1, 2\}$
- $w \mapsto \{0, 2\}$
- $x \mapsto \{1, 3\}$
- $y \mapsto \{2, 3, 4, 5\}$
- $z \mapsto \{5, 6\}$
**Efficient DC propagator (Régin, 1994) (Costa, 1994):** Mark all arcs in all strongly connected components (SCCs): the variables of an SCC take all the values of that SCC.

\[
\begin{align*}
  u & \mapsto \{0, 1\} \\
  v & \mapsto \{1, 2\} \\
  w & \mapsto \{0, 2\} \\
  x & \mapsto \{1, 3\} \\
  y & \mapsto \{2, 3, 4, 5\} \\
  z & \mapsto \{5, 6\}
\end{align*}
\]
Efficient DC propagator (Régis, 1994) (Costa, 1994):
Mark all arcs in all strongly connected components (SCCs): the variables of an SCC take all the values of that SCC.

\[
\begin{align*}
    u & \mapsto \{0, 1\} \\
v & \mapsto \{1, 2\} \\
w & \mapsto \{0, 2\} \\
x & \mapsto \{1, 3\} \\
y & \mapsto \{2, 3, 4, 5\} \\
z & \mapsto \{5, 6\}
\end{align*}
\]
Efficient DC propagator (Régis, 1994) (Costa, 1994):
Mark all arcs in all strongly connected components (SCCs): the variables of an SCC take all the values of that SCC.

\[
\begin{align*}
  u &\mapsto \{0, 1\} \\
  v &\mapsto \{1, 2\} \\
  w &\mapsto \{0, 2\} \\
  x &\mapsto \{1, 3\} \\
  y &\mapsto \{2, 3, 4, 5\} \\
  z &\mapsto \{5, 6\}
\end{align*}
\]
Efficient DC propagator (Régis, 1994) (Costa, 1994):
Mark all arcs in all strongly connected components (SCCs): the variables of an SCC take all the values of that SCC.

\[
\begin{align*}
\text{u} & \mapsto \{0, 1\} \\
\text{v} & \mapsto \{1, 2\} \\
\text{w} & \mapsto \{0, 2\} \\
\text{x} & \mapsto \{1, 3\} \\
\text{y} & \mapsto \{2, 3, 4, 5\} \\
\text{z} & \mapsto \{5, 6\}
\end{align*}
\]
Efficient DC propagator (Régis, 1994) (Costa, 1994): Mark all arcs in all strongly connected components (SCCs): the variables of an SCC take all the values of that SCC.

\[ u \mapsto \{0, 1\} \]
\[ v \mapsto \{1, 2\} \]
\[ w \mapsto \{0, 2\} \]
\[ x \mapsto \{1, 3\} \]
\[ y \mapsto \{2, 3, 4, 5\} \]
\[ z \mapsto \{5, 6\} \]

- 40 -
Efficient DC propagator (Régin, 1994) (Costa, 1994): Every arc that is neither in the chosen perfect matching nor marked is in no perfect matching: prune accordingly.

\[ u \mapsto \{0, 1\} \]
\[ v \mapsto \{1, 2\} \]
\[ w \mapsto \{0, 2\} \]
\[ x \mapsto \{1, 3\} \]
\[ y \mapsto \{2, 3, 4, 5\} \]
\[ z \mapsto \{5, 6\} \]
Efficient DC propagator (Régin, 1994) (Costa, 1994): Every arc that is neither in the chosen perfect matching nor marked is in no perfect matching: prune accordingly.

\[
\begin{align*}
  u &\mapsto \{0, 1\} \\
  v &\mapsto \{1, 2\} \\
  w &\mapsto \{0, 2\} \\
  x &\mapsto \{1, 3\} \\
  y &\mapsto \{2, 3, 4, 5\} \\
  z &\mapsto \{5, 6\}
\end{align*}
\]
Efficient DC propagator (Régin, 1994) (Costa, 1994):
Every arc that is neither in the chosen perfect matching nor marked is in no perfect matching: prune accordingly.

\[
\begin{align*}
  u & \mapsto \{0, 1\} & u & \rightarrow 0 \\
  v & \mapsto \{1, 2\} & v & \rightarrow 1 \\
  w & \mapsto \{0, 2\} & w & \rightarrow 2 \\
  x & \mapsto \{1, 3\} & x & \rightarrow 3 \\
  y & \mapsto \{2, 3, 4, 5\} & y & \rightarrow 4 \\
  z & \mapsto \{5, 6\} & z & \rightarrow 5 \\
\end{align*}
\]
Efficient DC propagator (Régine, 1994) (Costa, 1994):
Every arc that is neither in the chosen perfect matching nor marked is in no perfect matching: prune accordingly.

\[
\begin{align*}
u & \mapsto \{0, 1\} \\
v & \mapsto \{1, 2\} \\
w & \mapsto \{0, 2\} \\
x & \mapsto \{1, 3\} \\
y & \mapsto \{2, 3, 4, 5\} \\
z & \mapsto \{5, 6\}
\end{align*}
\]
Efficient DC propagator (Régine, 1994) (Costa, 1994):
Every arc that is in the chosen perfect matching but not marked is in every perfect matching: fixed variable.

- \( u \mapsto \{0, 1\} \)
- \( v \mapsto \{1, 2\} \)
- \( w \mapsto \{0, 2\} \)
- \( x \mapsto \{1, 3\} \)
- \( y \mapsto \{2, 3, 4, 5\} \)
- \( z \mapsto \{5, 6\} \)
Theorem (Berge, 1970) (Petersen, 1891)

Edge $e$ belongs to some maximum matching if and only if, for an arbitrarily chosen maximum matching $M$:

- $e$ belongs to a path of an even number of edges that starts at some node that is not incident to an edge of $M$ and that alternates between edges in $M$ and edges not in $M$;

- or $e$ belongs to a cycle of an even number of edges that alternates between edges in $M$ and edges not in $M$ (that is, the arc corresponding to $e$ belongs to an SCC).
Complexity and Incrementality

Complexity:
The described DC propagator takes \( \mathcal{O}(m \cdot \sqrt{n}) \) time and \( \mathcal{O}(m \cdot n) \) space for \( n \) variables and \( m \geq n \) values in their domains.

Incrementality via stateful propagator:
Keep the variable-value graph between invocations. When the propagator is re-invoked:

1. Delete marks on arcs.
2. Delete arcs that no longer correspond to the store.
3. If an arc of the old perfect matching was deleted, then first compute a new perfect matching.
4. Mark and prune.
Outline

1. Reification
2. Global Constraints
3. linear
4. channel
5. element
6. extensional
7. distinct
   Naïve DC Propagator
   Efficient DC Propagator
   Efficient BC Propagator
Is BC Needed for distinct?
Propagation to BC often suffices for distinct.

Example

Propagation to BC suffices to infer unsatisfiability for $\text{distinct}([x, y, z])$ from the store $\{x, y, z \mapsto \{4, 5\}\}$.

Efficient BC propagators:

There are BC propagators that take $O(n \cdot \lg n)$ time:

- Puget @ AAAI 1998
- Mehlhorn and Thiel @ CP 2000
- López-Ortiz, Quimper, Tromp, van Beek @ IJCAI 2003

The latter two run in $O(n)$ time if sorting can be avoided, say when there are as many values as variables.
Bibliography

Régin, Jean-Charles.  
A filtering algo. for constraints of difference in CSPs.  

Costa, Marie-Christine.  
Persistency in maximum cardinality bipartite matchings.  

Berge, Claude.  

Petersen, Julius.  
Die Theorie der regulären graphs.  

van Hoeve, Willem-Jan.  
The Alldifferent Constraint: A Survey.  
Extended from the version in ERCIM Workshop 2001.