Topic 16: Propagators [1]
(Version of 20th October 2021)

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

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

Outline

Outline

Reification

Implementation of $b \Leftrightarrow \gamma(\ldots)$:

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

- When the variable b gets fixed to 1, post the constraint $\gamma(\ldots)$.
- When the variable b gets fixed to 0, post the constraint $(\neg\gamma)(\ldots)$.
- When the constraint $\gamma(\ldots)$ gets subsumed, post the constraint $b = 1$.
- When the constraint $(\neg\gamma)(\ldots)$ gets subsumed, post the constraint $b = 0$.

where $(\neg\gamma)(\ldots)$ denotes the complement of $\gamma(\ldots)$, not the code for `not` $\gamma(\ldots)$, as CP solvers do not implement `not`.

Constraint combination with reification:

With reification, constraints can be arbitrarily combined with logical connectives: negation ($\neg$), disjunction ($\vee$), conjunction ($\&$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$). However, propagation may be very poor!

Example

The composite constraint $(\gamma_1 \ \& \ \gamma_2) \vee \gamma_3$ is modelled as

$$(b_1 \Leftrightarrow \gamma_1) \ \& \ (b_2 \Leftrightarrow \gamma_2) \ \& \ (b_3 \Leftrightarrow \gamma_3)$$
$$\& \ (b_1 \cdot b_2 = b) \ \& \ (b + b_3 \geq 1)$$

Hence even the constraints γ_1 and γ_2 must be reified.
If γ_1 is $x = y + 1$ and γ_2 is $y = x + 1$, then $\gamma_1 \ \& \ \gamma_2$ 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 γ_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` θ `then` ϕ `else` ψ `endif`) in MiniZinc often makes the solving slow?

Example

The MiniZinc disjunctive constraint

```
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

```
constraint x in {0,9};
```

where no reification is involved, and no further propagation.

Remember the strong warning in Topic 2: Basic Modelling about a conditional `if` θ `then` ϕ_1 `else` ϕ_2 `endif` or a comprehension, say `[i | i in` ρ `where` θ`]`, in MiniZinc having a test θ that depends on variables?

Example

Consider `var 1..9: x` and `var 1..9: y` for

```
forall(i in 1..9 where i > x)(i > y)
```

Recall that this is syntactic sugar for

```
forall([i > y | i in 1..9 where i > x])
```

This is flattened for Gecode into the equivalent of

```
forall(i in 1..9)(i > x -> i > y)
```

that is with a logical implication (`->`),
hence with a hidden logical disjunction (`\/`): for each `i`,
both sub-constraints are reified as both have variables.

Outline

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 $\langle 3, 1, 0\rangle$, $\langle 5, 0, 1\rangle$, 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.)

UPPSALA UNIVERSITET

Reification

Global Constraints

`linear`

`channel`

`Element`

`extensional`

`distinct`

Naïve DC Propagator

Efficient DC Propagator

Efficient BC Propagator

Globality from a Semantic Point of View

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

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

At least ℓ variables of $[x_1, \ldots, x_n]$ take the constant value v:

$$(\forall i \in 1..n : b_i \Leftrightarrow x_i = v) \ \& \ \sum_{i=1}^{n} b_i \geq \ell$$

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

Example ($\texttt{distinct}([x_1, \ldots, x_n])$)

$$\forall i, j \in 1..n \ \textbf{ where } \ i < j : x_i \neq x_j$$

Globality from a Propagation Point of View

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 $x_1 \neq x_2$ & $x_1 \neq x_3$ & $x_2 \neq x_3$:
  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

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 $\left(\sum_{i=1}^{n} a_i \cdot x_i\right) \; R \; d$ 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):

$$a \cdot x + b \cdot y = d \quad \Leftrightarrow \quad x = (d - b \cdot y) \;/\; a$$

Upper bound on x, starting from store s:

$$x \leq \left\lfloor \underbrace{\max\{(d - b \cdot n) \;/\; a \mid n \in s(y)\}}_{M} \right\rfloor$$

and (analogously, hence further details are omitted):

$$x \geq \lceil \min\{(d - b \cdot n) \;/\; a \mid n \in s(y)\} \rceil$$

Computing M:

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

BC propagator for a binary linear equality (end):

For $a > 0$ (the case $a < 0$ is analogous and omitted):

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

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:

☞ 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 BC($\mathbb{R}$) in $\mathcal{O}(n)$ time per iteration, but enforcing DC is NP-hard (so it currently takes time exponential in n).

Example (Why BC($\mathbb{R}$) and not BC($\mathbb{Z}$ / 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(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 ($\neq$): BC($\cdot$) = DC; $\mathcal{O}(n)$ time.
- Linear inequality ($<, \leq, \geq, >$): BC($\mathbb{R}$) = DC; $\mathcal{O}(n)$ time.

Outline

The `channel` Predicate

Definition

A `channel`$([x_1, \ldots, x_n], [y_1, \ldots, y_n])$ constraint holds iff:

$$\forall i, j \in 1..n : x_i = j \;\Leftrightarrow\; y_j = i$$

Propagator for domain consistency:

- For each $i \notin \mathrm{dom}(y_j)$: delete j from $\mathrm{dom}(x_i)$.
- For each $j \notin \mathrm{dom}(x_i)$: delete i from $\mathrm{dom}(y_j)$.
- Post `distinct`$([x_1, \ldots, x_n])$ 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`$([y_1, \ldots, y_n])$ as implied constraint would bring no further propagation.

UPPSALA UNIVERSITET

Outline

The `Element` Predicate

Definition (Van Hentenryck and Carillon, 1988)

An $\texttt{Element}([x_1, \ldots, x_n], i, e)$ 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 $\texttt{Element}([6,8,7,8], i, e)$ 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 $\texttt{Element}([x_1, \ldots, x_n], i, e)$:
$(i = 1 \Rightarrow x_1 = e) \;\; \& \;\; \cdots \;\; \& \;\; (i = n \Rightarrow x_n = e)$, 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:

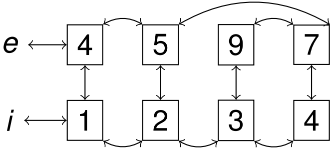
- This always iterates over the entire array $[x_1, \ldots, x_n]$.
- This always requires set intersection.
- This always requires sorting the 2nd argument of the 2nd intersection (or performing ordered set insertion).

Example

Consider the constraint `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:

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

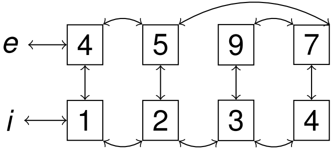


Example

Consider the constraint `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:

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



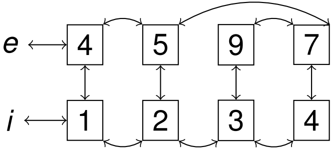
- i Follow the i-links: if a value is not in $s(i)$, then unlink the corresponding two nodes from the two lists.

Example

Consider the constraint `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:

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



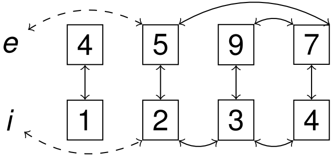
- *i* Follow the *i*-links: if a value is not in $s(i)$, then unlink the corresponding two nodes from the two lists.

Example

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

Smart DC propagator:

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



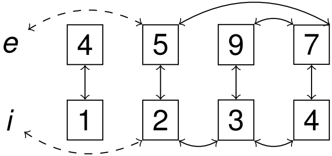
- Follow the i-links: if a value is not in $s(i)$, then unlink the corresponding two nodes from the two lists.

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 $\{i \mapsto \{2,4\},\ e \mapsto \{5,7\}\}$.

Smart DC propagator:

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



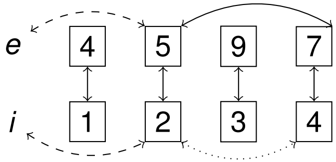
- *i* Follow the *i*-links: if a value is not in $s(i)$, then unlink the corresponding two nodes from the two lists.
- *e* Follow the *e*-links: if a value is not in $s(e)$, then unlink the corresponding two nodes from the two lists.

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 $\{i \mapsto \{2,4\},\ e \mapsto \{5,7\}\}$.

Smart DC propagator:

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



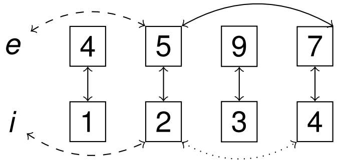
- *i* Follow the *i*-links: if a value is not in $s(i)$, then unlink the corresponding two nodes from the two lists.
- *e* Follow the *e*-links: if a value is not in $s(e)$, then unlink the corresponding two nodes from the two lists.

Example

Consider the constraint `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:

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



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

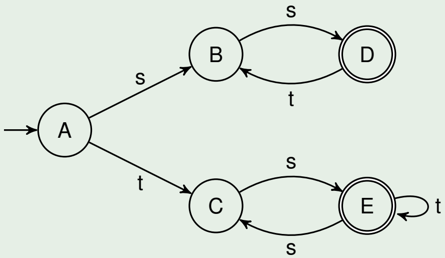
An incremental propagator, instead of throwing away an internal data structure when at fixpoint, keeps it for its next invocation: it first repairs that data structure according to the pruning done by other propagators since its previous invocation, and then only attempts its own pruning.

- 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

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 Σ.

The `extensional` Predicate

Definition

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

Example

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

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

to the fixpoint

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

Efficient DC Propagator (Pesant, 2004)

Let us propagate `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:

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

Efficient DC Propagator (Pesant, 2004)

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

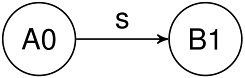
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$

A0

Efficient DC Propagator (Pesant, 2004)

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

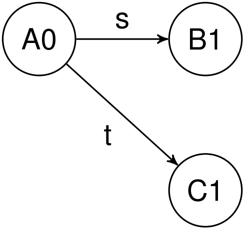
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$



Efficient DC Propagator (Pesant, 2004)

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

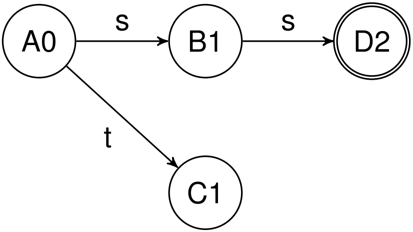
$$x_1 \mapsto \{\mathrm{s}, \mathrm{t}\} \quad x_2 \mapsto \{\mathrm{s}, \mathrm{t}\} \quad x_3 \mapsto \{\mathrm{s}, \mathrm{t}\} \quad x_4 \mapsto \{\mathrm{s}, \mathrm{t}\}$$



Efficient DC Propagator (Pesant, 2004)

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

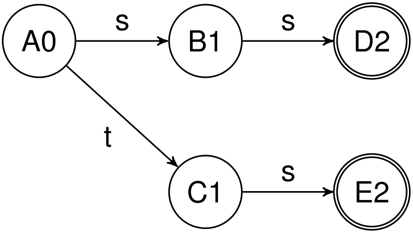
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$



Efficient DC Propagator (Pesant, 2004)

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

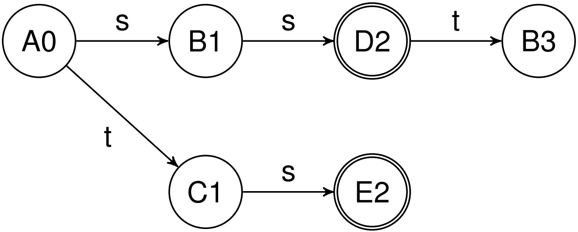
$$x_1 \mapsto \{\mathrm{s}, \mathrm{t}\} \quad x_2 \mapsto \{\mathrm{s}, \mathrm{t}\} \quad x_3 \mapsto \{\mathrm{s}, \mathrm{t}\} \quad x_4 \mapsto \{\mathrm{s}, \mathrm{t}\}$$



Efficient DC Propagator (Pesant, 2004)

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

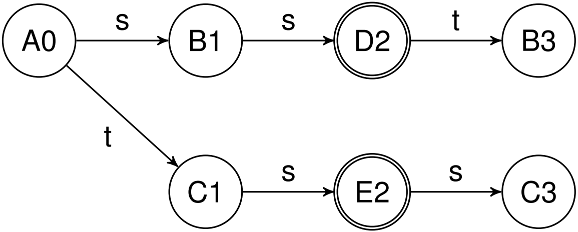
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$



Efficient DC Propagator (Pesant, 2004)

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

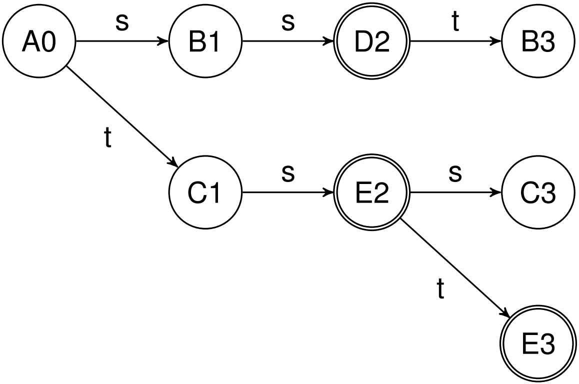
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$



Efficient DC Propagator (Pesant, 2004)

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

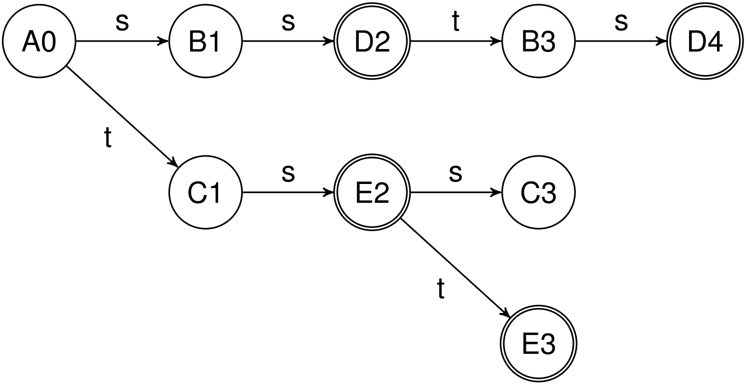
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$



Efficient DC Propagator (Pesant, 2004)

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

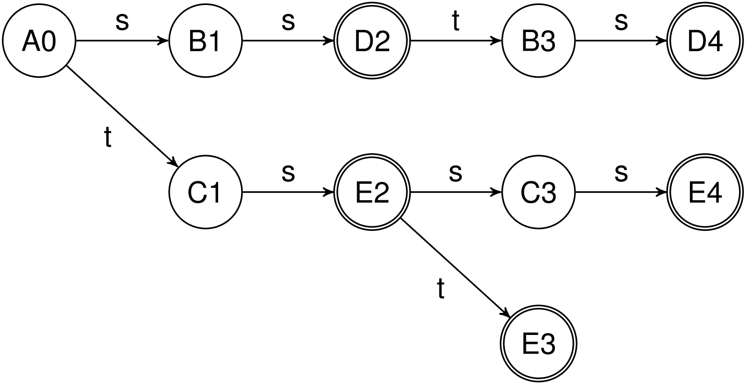
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$



Efficient DC Propagator (Pesant, 2004)

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

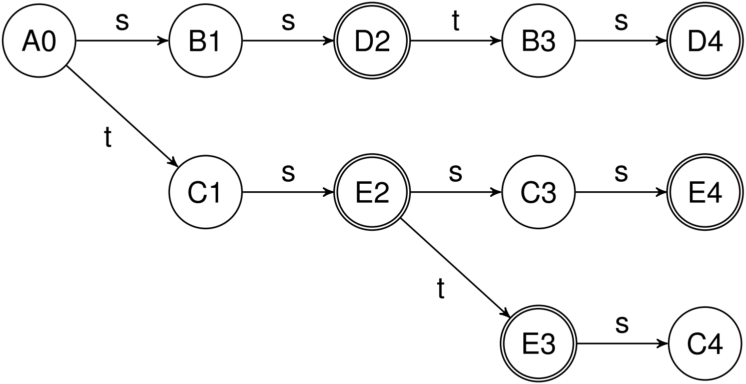
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$



Efficient DC Propagator (Pesant, 2004)

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

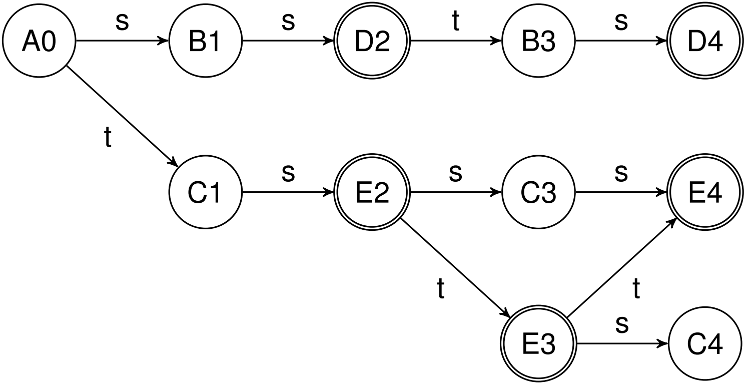
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$



Efficient DC Propagator (Pesant, 2004)

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

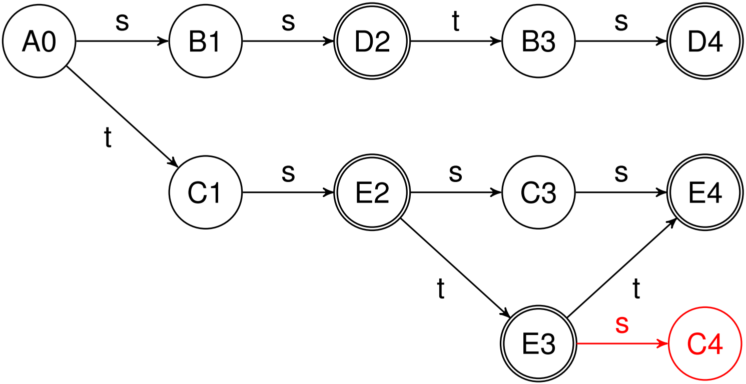
$$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$$



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.

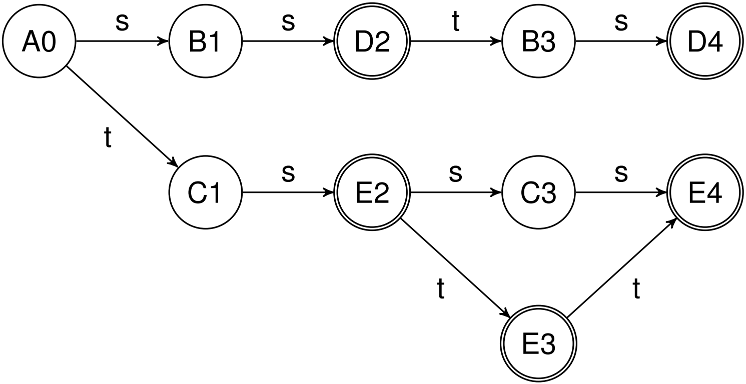
$x_1 \mapsto \{s, t\} \quad x_2 \mapsto \{s, t\} \quad x_3 \mapsto \{s, t\} \quad x_4 \mapsto \{s, t\}$



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.

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



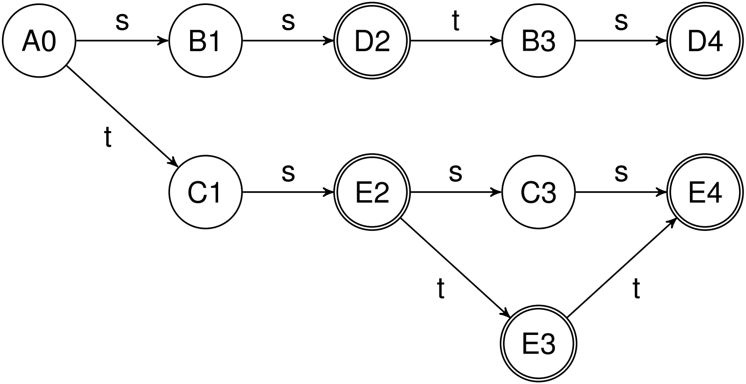
UPPSALA UNIVERSITET

- Reification
- Global Constraints
- linear
- channel
- Element
- **extensional**
- distinct
  - Naïve DC Propagator
  - Efficient DC Propagator
  - Efficient BC Propagator

Efficient DC Propagator (Pesant, 2004)

Pruning Phase: Delete unsupported values; at fixpoint.

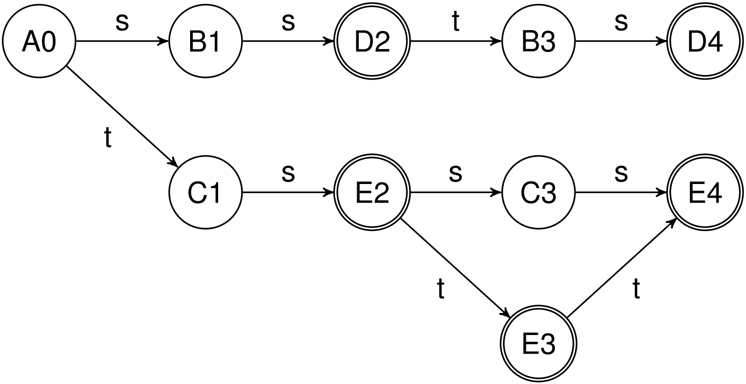
$x_1 \mapsto \{s, t\}$ $x_2 \mapsto \{s, t\}$ $x_3 \mapsto \{s, t\}$ $x_4 \mapsto \{s, t\}$



Efficient DC Propagator (Pesant, 2004)

Pruning Phase: Delete unsupported values; at fixpoint.

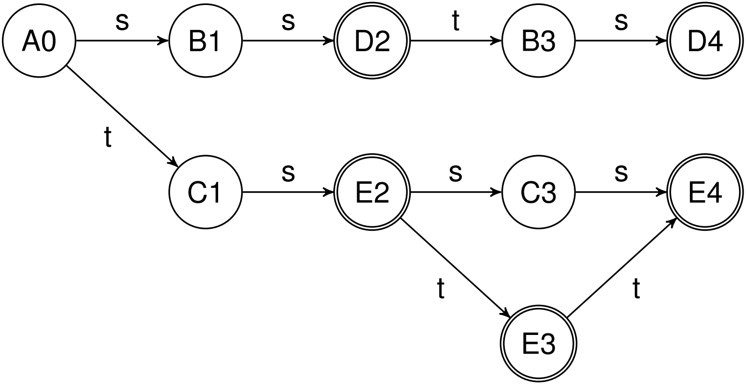
$x_1 \mapsto \{s,t\}$ $x_2 \mapsto \{s\}$ $x_3 \mapsto \{s,t\}$ $x_4 \mapsto \{s,t\}$



Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_1 = \mathrm{t}$ to fixpoint.

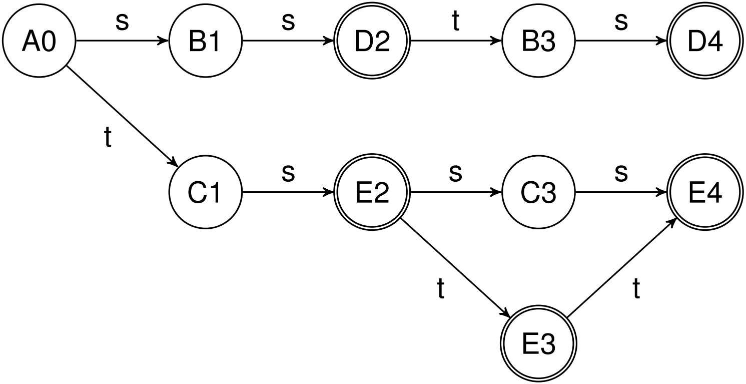
$x_1 \mapsto \{\mathrm{s}, \mathrm{t}\}$ $x_2 \mapsto \{\mathrm{s}\}$ $x_3 \mapsto \{\mathrm{s}, \mathrm{t}\}$ $x_4 \mapsto \{\mathrm{s}, \mathrm{t}\}$



Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_1 = \text{t}$ to fixpoint.

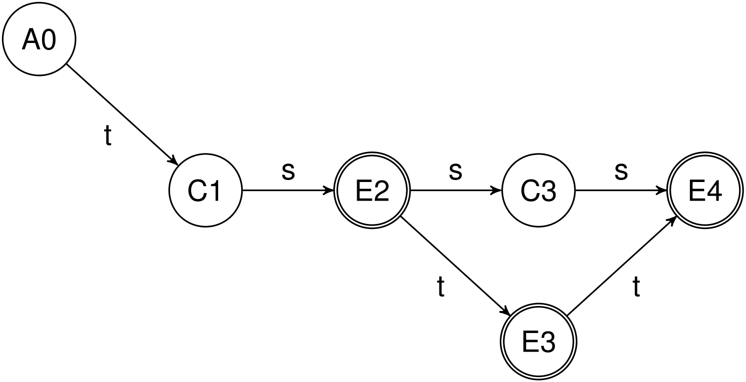
$x_1 \mapsto \{\text{t}\}$ $x_2 \mapsto \{\text{s}\}$ $x_3 \mapsto \{\text{s}, \text{t}\}$ $x_4 \mapsto \{\text{s}, \text{t}\}$



Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_1 = \mathrm{t}$ to fixpoint.

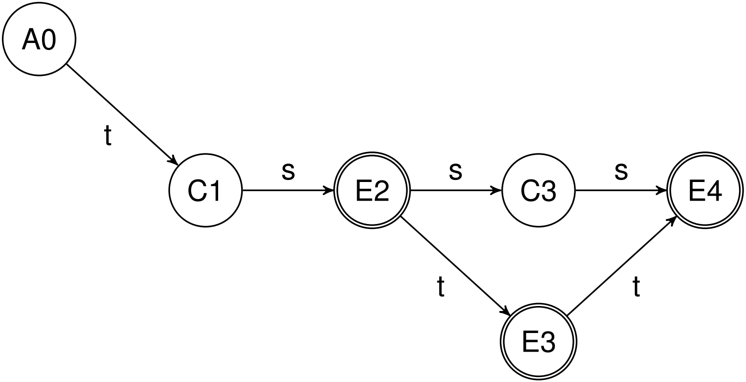
$$x_1 \mapsto \{\mathrm{t}\} \qquad x_2 \mapsto \{\mathrm{s}\} \qquad x_3 \mapsto \{\mathrm{s}, \mathrm{t}\} \qquad x_4 \mapsto \{\mathrm{s}, \mathrm{t}\}$$



Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_3 = \mathrm{s}$ to subsumption.

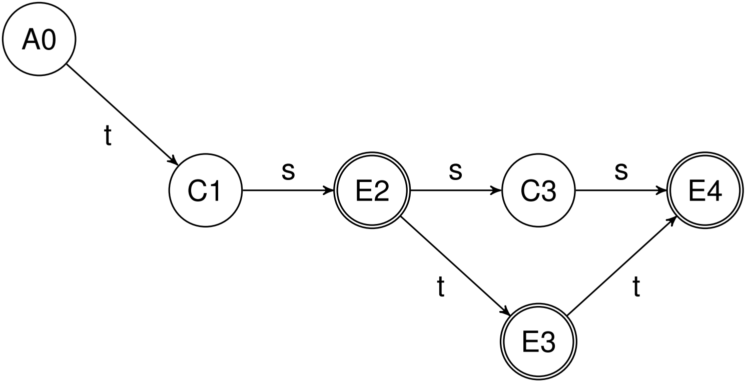
$$x_1 \mapsto \{\mathrm{t}\} \qquad x_2 \mapsto \{\mathrm{s}\} \qquad x_3 \mapsto \{\mathrm{s}, \mathrm{t}\} \qquad x_4 \mapsto \{\mathrm{s}, \mathrm{t}\}$$



Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_3 = \text{s}$ to subsumption.

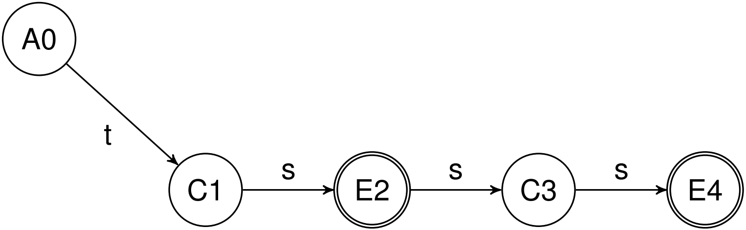
$x_1 \mapsto \{\text{t}\}$ $x_2 \mapsto \{\text{s}\}$ $x_3 \mapsto \{\text{s}\}$ $x_4 \mapsto \{\text{s}, \text{t}\}$



Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_3 = \mathrm{s}$ to subsumption.

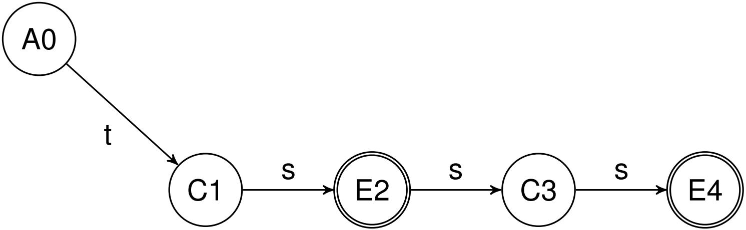
$$x_1 \mapsto \{\mathrm{t}\} \qquad x_2 \mapsto \{\mathrm{s}\} \qquad x_3 \mapsto \{\mathrm{s}\} \qquad x_4 \mapsto \{\mathrm{s}, \mathrm{t}\}$$



Efficient DC Propagator (Pesant, 2004)

Incremental propagation upon $x_3 = \text{s}$ to subsumption.

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



Complexity and Incrementality

Complexity:

The described DC propagator takes $\mathcal{O}(n \cdot m \cdot q)$ 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

Beldiceanu, Nicolas; Carlsson, Mats; Petit, Thierry.
Deriving filtering algorithms from constraint checkers.
Proceedings of CP 2004, Lecture Notes in Computer Science 3258, pages 107 – 122. Springer, 2004.

Pesant, Gilles.
A regular language membership constraint for finite sequences of variables.
Proceedings of CP 2004, Lecture Notes in Computer Science 3258, pages 482 – 495. Springer, 2004.

Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D.
Intro. to Automata Theory, Languages, & Computation.
Third edition. Addison-Wesley, 2007.

Outline

The `distinct` Predicate

Definition (Laurière, 1978)

A `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:

$$\forall i, j \in 1..n \;\; \textbf{where} \;\; i < j : x_i \neq x_j$$

Originally, the `distinct` constraint was just a wrapper for posting those $\frac{n \cdot (n-1)}{2}$ disequality constraints. The first efficient domain-consistency propagators for `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 $\texttt{distinct}([x_1, x_2, x_3])$:

- 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 ($\texttt{distinct}([u, v, w, x, y, z])$)

From the store

$$\left\{ \begin{array}{c} u \mapsto \{0,1\}\,,\ v \mapsto \{1,2\}\,,\ w \mapsto \{0,2\}\,, \\ x \mapsto \{1,3\}\,,\ y \mapsto \{2,3,4,5\}\,,\ z \mapsto \{5,6\} \end{array} \right\}$$

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.

n	value consistency	domain consistency
7	950 nodes	474 nodes
8	7,622 nodes	3,076 nodes
9	55,930 nodes	16,608 nodes
10	413,922 nodes	97,782 nodes
11	6,330,568 nodes	1,448,666 nodes

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

Outline

Variable-Value Graph:

Construct a bipartite graph from the current domains:

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

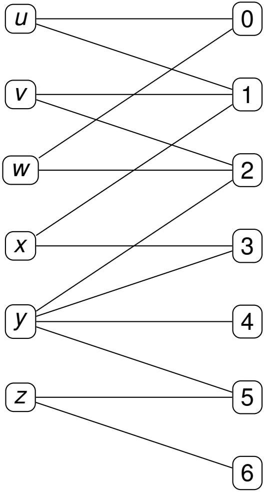
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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:

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

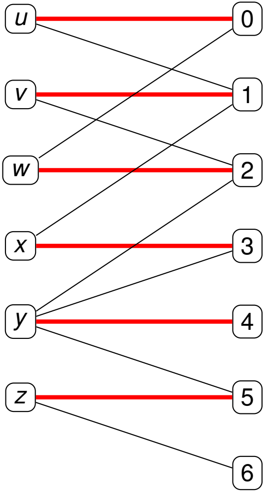
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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:

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

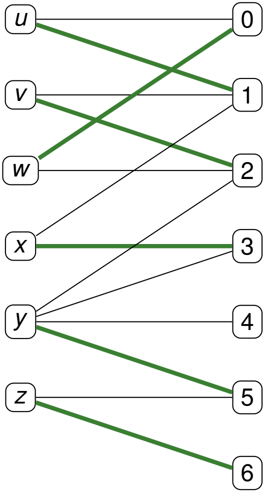
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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:

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

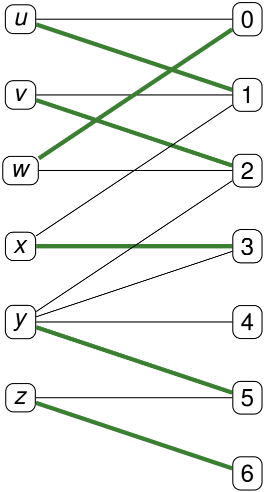
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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 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

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.

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

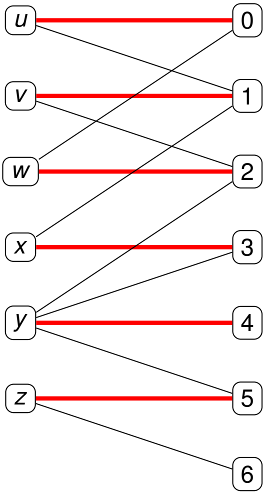
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



Efficient DC propagator (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.

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

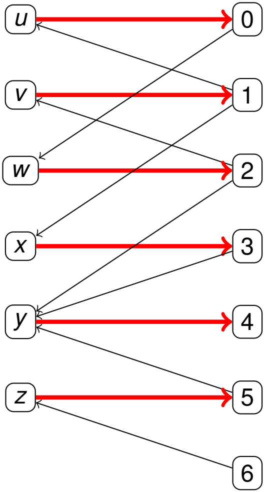
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



Efficient DC propagator (Régin, 1994) (Costa, 1994):

Start from all unmatched vertices (necessarily values here) and mark all arcs on all simple paths: arcs can be flipped.

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

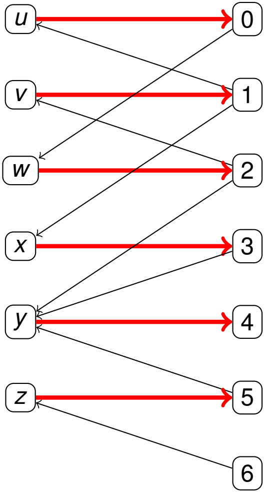
$v \mapsto \{1,2\}$

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

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

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

$z \mapsto \{5,6\}$



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.

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

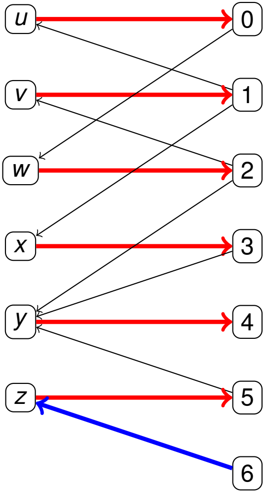
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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.

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

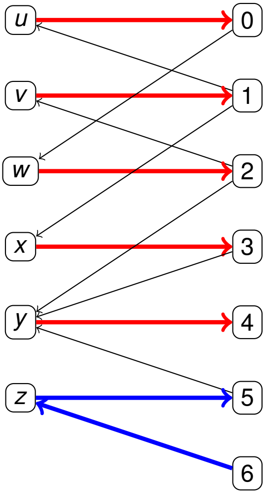
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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.

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

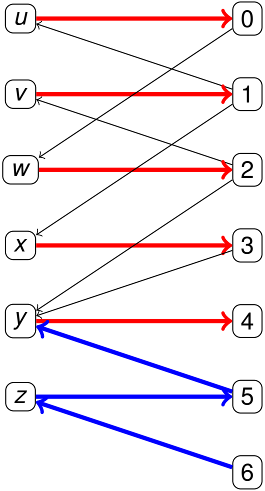
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



UPPSALA UNIVERSITET

Reification

Global Constraints

linear

channel

Element

extensional

distinct

Naïve DC Propagator

Efficient DC Propagator

Efficient BC Propagator

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.

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

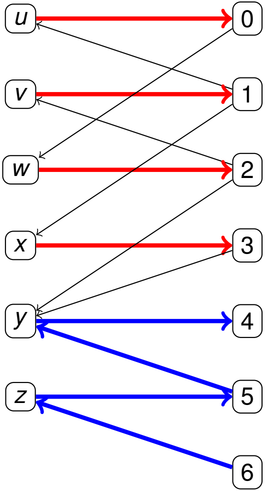
$v \mapsto \{1,2\}$

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

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

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

$z \mapsto \{5,6\}$



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.

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

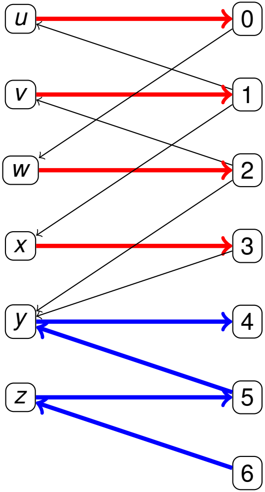
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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.

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

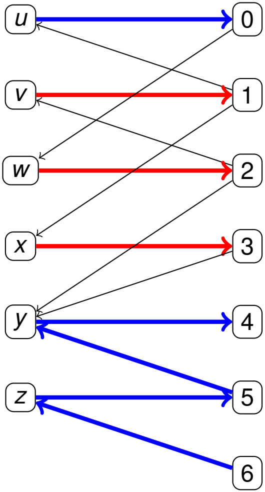
$v \mapsto \{1,2\}$

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

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

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

$z \mapsto \{5,6\}$



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.

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

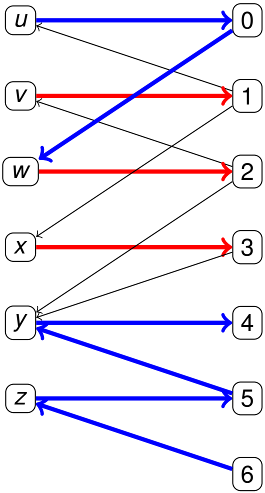
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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.

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

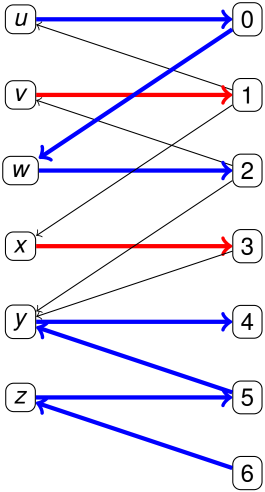
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



UPPSALA UNIVERSITET

Reification

Global Constraints

linear

channel

Element

extensional

distinct

Naïve DC Propagator

Efficient DC Propagator

Efficient BC Propagator

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.

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

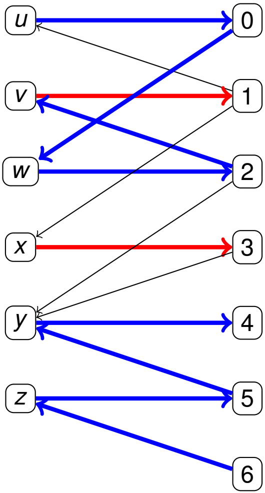
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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.

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

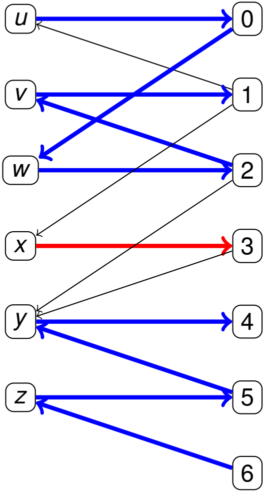
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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.

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

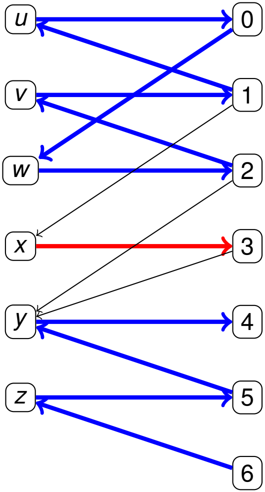
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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.

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

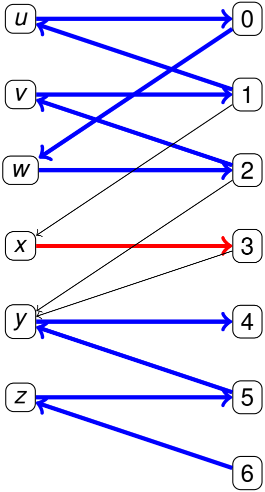
$v \mapsto \{1,2\}$

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

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

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

$z \mapsto \{5,6\}$



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.

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

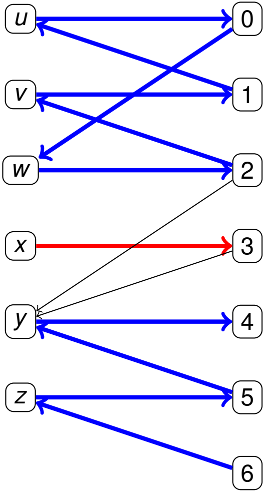
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



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.

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

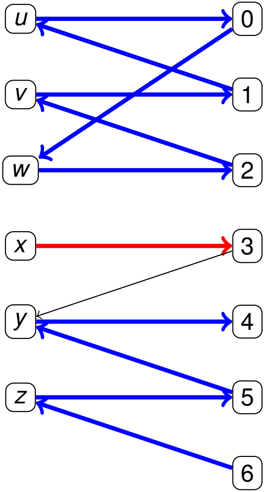
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



UPPSALA UNIVERSITET

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.

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

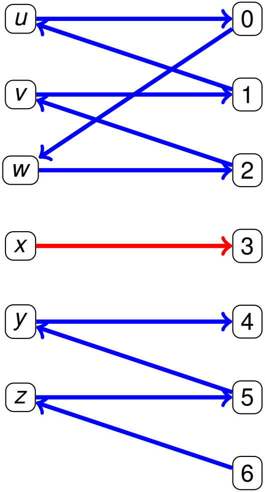
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



Efficient DC propagator (Régin, 1994) (Costa, 1994):

Every arc that is in the chosen perfect matching but not marked is in *every* perfect matching: fixed variable.

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

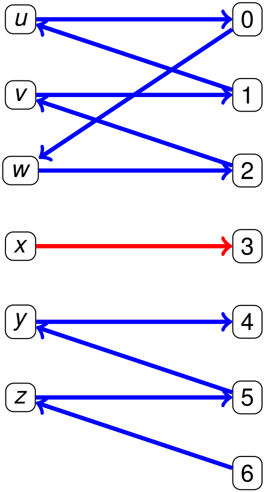
$v \mapsto \{1, 2\}$

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

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

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

$z \mapsto \{5, 6\}$



Underlying Theorem from Matching Theory

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

$$\mathcal{O}(m \cdot \sqrt{n}) \text{ time and } \mathcal{O}(m \cdot n) \text{ space}$$

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

Is BC Needed for `distinct`?

Propagation to BC often suffices for `distinct`.

Example

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

Efficient BC propagators:

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

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

The latter two run in $\mathcal{O}(n)$ 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.
AAAI 1994, pages 362–367, 1994.

Costa, Marie-Christine.
Persistency in maximum cardinality bipartite matchings.
Operations Research Letters, 15(3):143–149, 1994.

Berge, Claude.
Graphes et Hypergraphes. Dunod, 1970.

Petersen, Julius.
Die Theorie der regulären graphs.
Acta Mathematica, 15(1):193–220, 1891.

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