THE EXTENSIONAL CONSTRAINT

Private defense

Hélène Verhaeghe

14 June 2021

ICTEAM, UCLouvain, Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium, *helene.verhaeghe@uclouvain.be*

Advisors:

- Pierre Schaus
- Christophe Lecoutre

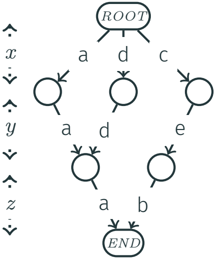
Thesis jury:

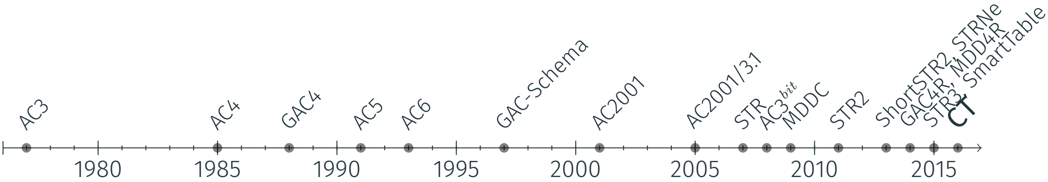
- Yves Deville
- Claude-Guy Quimper
- Jean-Charles Régin
- Peter Van Roy

	x	y	z
τ_1	a	a	a
τ_2	d	d	a
τ_3	c	e	b
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tables are one of the oldest most used CP constraints

MDDs are equivalent to tables



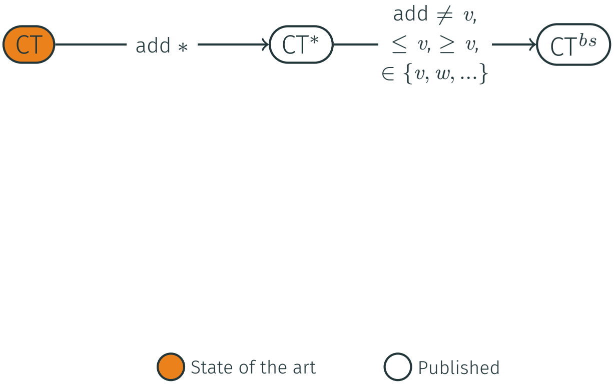


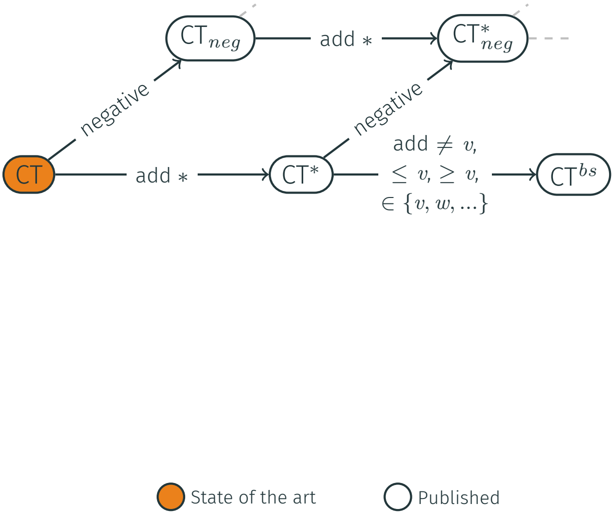
2016 : New algorithm! Compact-Table [CP2016], based on bitwise operations, completely outperformed existing algorithms

State of the art

Published

THE 3 DIMENSIONS OF EXTENSION

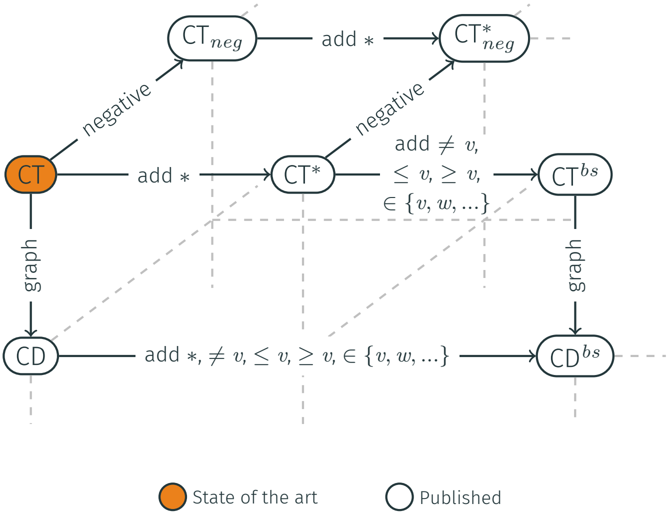




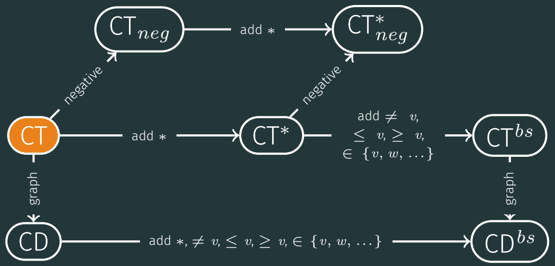
CT_{neg}
add $*$
CT^*_{neg}
negative
negative
CT
add $*$
CT^*
add $\neq v$,
$\leq v$, $\geq v$,
$\in \{v, w, \ldots\}$
CT^{bs}
State of the art
Published

THE 3 DIMENSIONS OF EXTENSION

UCLouvain



COMPACT-TABLE



Table

	x_1	x_2	x_3
τ_1	a	c	a
τ_2	b	b	b
τ_3	a	c	b
τ_4	c	a	b
τ_5	b	c	b
τ_6	c	b	c
τ_7	a	a	b
τ_8	b	b	c

`currTable` (Reversible Sparse Bitset)

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
currTable	1	1	1	1	1	1	1	1

`supports` (Precomputed Bitsets)

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
$[x_1, a]$	1	0	1	0	0	0	1	0
$[x_1, b]$	0	1	0	0	1	0	0	1
$[x_1, c]$	0	0	0	1	0	1	0	0
$[x_2, a]$	0	0	0	1	0	0	1	0
$[x_2, b]$	0	1	0	0	0	1	0	1
$[x_2, c]$	1	0	1	0	1	0	0	0
$[x_3, a]$	1	0	0	0	0	0	0	0
$[x_3, b]$	0	1	1	1	1	0	1	0
$[x_3, c]$	0	0	0	0	0	1	0	1

UCLouvain

Table

	x_1	x_2	x_3
τ_1	a	c	a
τ_2	b	b	b
τ_3	a	c	b
τ_4	c	a	b
τ_5	b	c	b
τ_6	c	b	c
τ_7	a	a	b
τ_8	b	b	c

currTable (Reversible Sparse Bitset)

τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
1	1	1	1	1	1	1	1

supports (Precomputed Bitsets)

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
$[x_1, a]$	1	0	1	0	0	0	1	0
$[x_1, b]$	0	1	0	0	1	0	0	1
$[x_1, c]$	0	0	0	1	0	1	0	0
$[x_2, a]$	0	0	0	1	0	0	1	0
$[x_2, b]$	0	1	0	0	0	1	0	1
$[x_2, c]$	1	0	1	0	1	0	0	0
$[x_3, a]$	1	0	0	0	0	0	0	0
$[x_3, b]$	0	1	1	1	1	0	1	0
$[x_3, c]$	0	0	0	0	0	1	0	1

Goal of the update

Remove invalid tuples from `currTable`

`Classical update`

Δ_x

`supports`$[x,b]$	0	0	0	1
`supports`$[x,d]$	1	0	0	0
`supports`$[x,f]$	0	1	0	0

$\sim \cup =$

`mask`	0	0	1	0

$\cap$

old `currTable`	1	1	1	0

$=$

new `currTable`	0	0	1	0

`Reset update`

$dom(x)$

`supports`$[x,a]$	1	0	0	0
`supports`$[x,c]$	0	1	0	0
`supports`$[x,e]$	0	0	0	1

$\cup =$

`mask`	1	1	0	1

$\cap$

old `currTable`	1	0	1	0

$=$

new `currTable`	1	0	0	0

- Classical update :

$$\mathcal{O}(|\Delta_x|)$$

- Reset update :

$$\mathcal{O}(|dom(x)|)$$

Goal of the update

Remove invalid tuples from `currTable`

Algorithm: Update(x)

```
foreach variable x ∈ scp where |Δx| > 0 do
    if |Δx| < |dom(x)| then
        ClassicalUpdate(x);
    else
        ResetUpdate(x);
```

currTable	0	1	1	0
		$\cap$		
supports[x,a]	1	1	0	0
		$=$		
intersection	0	1	0	0

$\Downarrow$

not empty

$\Downarrow$

$a \in dom(x)$

currTable	0	1	1	0
		$\cap$		
supports[x,b]	0	0	0	1
		$=$		
intersection	0	0	0	0

$\Downarrow$

empty

$\Downarrow$

$dom(x) \setminus \{b\}$

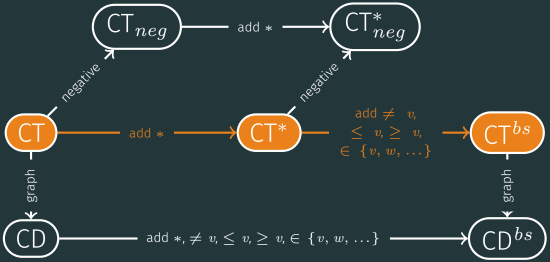
Goal of the propagation

Remove unsupported values

Algorithm: Propagate()

```
foreach variable x ∈ scp do
    foreach value a ∈ dom(x) do
        if currTable & supports[x, a] = 0 then
            dom(x) ← dom(x) \ {a} ;
```

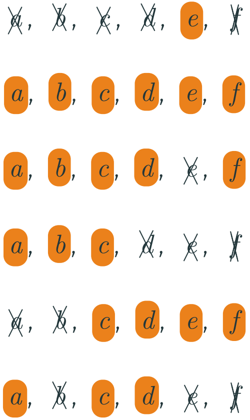
1ST DIMENSION: FROM GROUND TABLES TO SMART TABLES



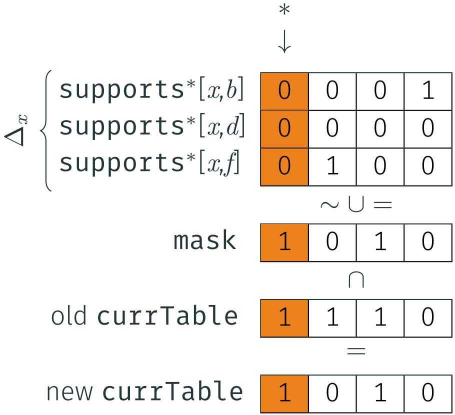
A Basic Smart Table — contains unary Smart Elements — representing multiples values

	x	y	z
τ_1	$*$	$*$	$\in \{a, b\}$
τ_2	$\neq a$	c	$\leq a$
τ_3	b	$*$	$*$
τ_4	$\geq c$	$\neq b$	a
	$\vdots$	$\vdots$	$\vdots$

- single value: e
- universal value: $*$
- exclusion: $\neq e$
- upper bound: $\leq c$
- lower bound: $\geq c$
- set: $\in \{a, c, d\}$



UCLouvain

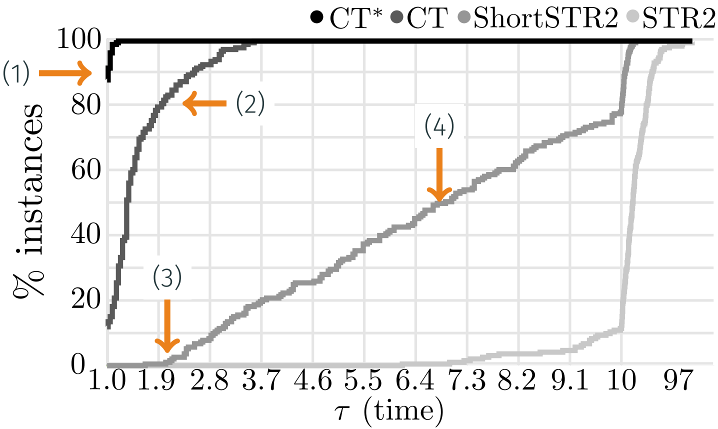


Goal of the update

Remove invalid tuples from `currTable`

Algorithm: ClassicalUpdate(x)

```
mask ← 0 ;
foreach value a ∈ Δ_x do
    mask ← mask | supports*[x, a] ;
mask ← ~ mask ;
currTable ← currTable & mask ;
```



Complexity of CT*:
same as CT ($\mathcal{O}(rd\frac{t}{w})$)

(1) CT* best 90% of the time

(2) CT requires $< 2\times$ time on 80%

(3) ShortSTR2 needs $> 2\times$ time

(4) ShortSTR2 needs $> 7\times$ time on 50%

600 instances, 20 variables, domain size from 5 to 7, 40 random tables by instances, arity of 6 or 7, tightness [0.5%;2%], 1, 5, 10 or 20 % of short tuples

UCLouvain

$$|dom(x)| == 0$$

Trivial!
Handled by variable x

$$|dom(x)| == 1$$

$|\Delta_x| \geq |dom(x)|$ always true!
`ResetUpdate(x)` used
and already working!

$$|dom(x)| > 1$$

If $|\Delta_x| < |dom(x)|$

Tuple always valid!

At least one valid value
`supports`$^*[x][\tau] = 0$

If $|\Delta_x| \geq |dom(x)|$

`ResetUpdate(x)` used
and already working!

UCLouvain

	$\leq d$ ↓	$\leq b$ ↓	$\geq e$ ↓	$\geq b$ ↓
∼mask	1	1	1	1
∩				
supportsMin[x, $\underbrace{c}_{x.min}$]	1	0	1	1
∩				
supportsMax[x, $\underbrace{d}_{x.max}$]	1	1	0	1
=				
new mask	1	0	0	1
∩				
old currTable	0	0	1	1
=				
new currTable	0	0	0	1

Goal of the update

Remove invalid tuples from `currTable`

Algorithm: ClassicalUpdate(x)

```
mask ← 0 ;
foreach value a ∈ Δ_x do
    if a ∈ [dom(x).min; dom(x).max] then
        mask ← mask | supports*[x, a] ;
mask ← ∼ mask ;
mask ← mask &
  supportsMin[x, dom(x).min] ;
mask ← mask &
  supportsMax[x, dom(x).max] ;
currTable ← currTable & mask ;
```

UCLouvain

$\lvert dom(x) \rvert$	sets		structured sets	
1	$\{a\}$	1	$*$	1
2	$\{a\}, \{b\}, \{a, b\}$	3	$a, b, *$	3
3	$\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}$	7	$a, b, c, \neq a, \neq b, \neq c, *$	7
4	$\{a\}, \{b\}, \ldots, \{a, b\}$, {a,c}, {a,d}, {b,c}, {b,d}, $\{c, d\}, \{a, b, c\}, \ldots, \{a, b, c, d\}$	15	$a, b, c, d, \leq b, \geq c, \neq a, \neq b, \neq c, \neq d, *$	11
5	$\{a\}, \{b\}, \ldots, \{a, b\}$, {a,c}, {a,d}, {a,e}, {b,c}, {b,d}, {b,e}, {c,d}, {c,e}, $\{a, b, c\}$, {a,b,d}, {a,b,e}, {a,c,d}, {a,c,e}, $\ldots, \{a, b, c, d\}, \ldots, \{a, b, c, d, e\}$	31	$a, b, c, d, e, \leq b, \leq c, \geq c, \geq d, \neq a, \neq b, \neq c, \neq d, \neq e, *$	15

CT^{bs}: CT WITH MODIFIED UPDATE

Algorithm: Update(x)

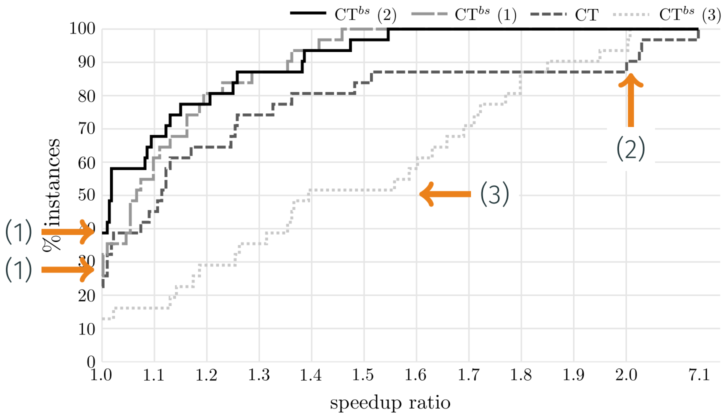
foreach variable $x \in$ scp$_{no \in S}$ **do**
  if $|\Delta_x| < |dom(x)|$ **then**
    ClassicalUpdate(x);
  else
    ResetUpdate(x);
foreach variable $x \in$ scp$_{with \in S}$ **do**
  ResetUpdate(x);

Algorithm: ClassicalUpdate(x)

mask $\leftarrow 0$;
foreach value $a \in \Delta_x$ **do**
  if $a \in [dom(x).min; dom(x).max]$ **then**
    mask $\leftarrow$ mask | supports$^*[x, a]$;
mask $\leftarrow \sim$ mask ;
mask $\leftarrow$ mask & supportsMin$[x, dom(x)$.min$]$;
mask $\leftarrow$ mask & supportsMax$[x, dom(x)$.max$]$;
currTable $\leftarrow$ currTable & mask ;

- supports$[x,v]$: supports value v
- supports$^*[x,v]$: supports only value v
- supportsMin$[x,v]$: supports at least one value $\geq v$
- supportsMax$[x,v]$: supports at least one value $\leq v$

UCLouvain



Complexity of CT^{bs}:
same as CT ($\mathcal{O}(rd\frac{t}{w})$)

(1) CT^{bs} best on 40 + 30%
(2) CT needs $< 2\times$ for 88%
(3) Overhead due to Set only

XCSP3 instances with only tables, transformed into basic smart table with at least 10% compression (1) with only $\leq v$ and $\geq v$, (2) with $\leq v$ and $\geq v$ + post processing to add $*$ and $\neq v$, (3) with elements treated as simple sets

UCLouvain

A Full Smart Table

	x	y	z
τ_1	$\leq z - b$	$*$	$\in \{a, b\}$
τ_2	$\neq a$	c	$\leq a$
τ_3	b	$= x + a$	$*$
τ_4	$\geq c$	$\neq b$	a
τ_5	$\geq y + a$	$\neq z - c$	$*$
	$\vdots$	$\vdots$	$\vdots$

unary Smart Elements

- single value: e
- universal value: $*$
- exclusion: $\neq e$
- upper bound: $\leq c$
- lower bound: $\geq c$
- set: $\in \{a, c, d\}$

binary Smart Elements

- value: $= x + v$
- exclusion: $\neq x + v$
- upper bound: $\leq x + v$
- lower bound: $\geq x + v$

$$\tau_1 = (1, *, = x_2, *)$$

$$\tau_2 = (0, *, = x_2, = x_3)$$

$\tau_1 = (1, *, = x_2, *)$ $\tau_2 = (0, *, = x_2, = x_3)$

Removal of value 1 from $dom(x_2)$

$\tau_1 = (1, *, = x_2, *)$

$\tau_2 = (0, *, = x_2, = x_3)$

Removal of value 1 from $dom(x_2)$

no impact on x_1
τ_1 doesn't allow $x_3 = 1$
no impact on x_4

no impact on x_1
τ_2 doesn't allow $x_3 = 1$
τ_2 doesn't allow $x_4 = 1$

$\tau_1 = (1, *, = x_2, *)$

$\tau_2 = (0, *, = x_2, = x_3)$

Removal of value 1 from $dom(x_2)$

no impact on x_1
τ_1 doesn't allow $x_3 = 1$
no impact on x_4

no impact on x_1
τ_2 doesn't allow $x_3 = 1$
τ_2 doesn't allow $x_4 = 1$

Removal of value 1 from $dom(x_1)$

$\tau_1 = (1, *, = x_2, *)$

$\tau_2 = (0, *, = x_2, = x_3)$

Removal of value 1 from $dom(x_2)$

no impact on x_1
τ_1 doesn't allow $x_3 = 1$
no impact on x_4

no impact on x_1
τ_2 doesn't allow $x_3 = 1$
τ_2 doesn't allow $x_4 = 1$

Removal of value 1 from $dom(x_1)$

τ_1 becomes unvalid

1 should be removed from $dom(x_4)$ since not supported by τ_2

$\tau_1 = (1, *, = x_2, *)$

$\tau_2 = (0, *, = x_2, = x_3)$

Removal of value 1 from $dom(x_2)$

no impact on x_1
τ_1 doesn't allow $x_3 = 1$
no impact on x_4

no impact on x_1
τ_2 doesn't allow $x_3 = 1$
τ_2 doesn't allow $x_4 = 1$

Removal of value 1 from $dom(x_1)$

τ_1 becomes unvalid

1 should be removed from $dom(x_4)$ since not supported by τ_2

Conflict with uniform aproach for similar smart elements

x	y	z
$*$	$= x + v$	v
$\leq y + v$	$\geq z + v$	$*$
$= y + v$	$\leq x + v$	$*$

Smart Table

$=$

x	y	z	$aux_{(x,y)}$	$aux_{(x,z)}$	$aux_{(y,z)}$
$*$	$*$	v	$-v$	$*$	$*$
$*$	$*$	$*$	$*$	$\leq v$	$\geq v$
$*$	$*$	$*$	$= v$	$*$	$*$

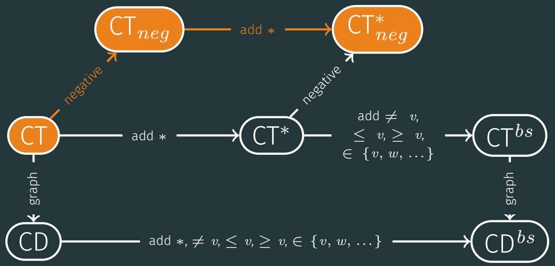
Basic Smart Table

$+$

$$aux_{(x,y)} = x - y$$
$$aux_{(x,z)} = x - z$$
$$aux_{(y,z)} = y - z$$

Auxillary constraints

2ND DIMENSION: FROM POSITIVE TO NEGATIVE TABLES



Negative Table

	x_1	x_2	x_3
τ_1	a	c	a
τ_2	b	b	b
τ_3	a	c	b
τ_4	c	a	b
τ_5	b	c	b
τ_6	c	b	c
τ_7	a	a	b
τ_8	b	b	c

currTable (Reversible Sparse Bitset)

τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
1	1	1	1	1	1	1	1

dangerous supports (Precomputed Bitsets)

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
$[x_1, a]$	1	0	1	0	0	0	1	0
$[x_1, b]$	0	1	0	0	1	0	0	1
$[x_1, c]$	0	0	0	1	0	1	0	0
$[x_2, a]$	0	0	0	1	0	0	1	0
$[x_2, b]$	0	1	0	0	0	1	0	1
$[x_2, c]$	1	0	1	0	1	0	0	0
$[x_3, a]$	1	0	0	0	0	0	0	0
$[x_3, b]$	0	1	1	1	1	0	1	0
$[x_3, c]$	0	0	0	0	0	1	0	1

Hypothesis

No duplicate!

No overlap!

currTable	1	1	1	0

$\cap$

supports[x,a]	0	0	1	1

$=$

intersection	0	0	1	0

$\Downarrow$

$$\texttt{bitCount} = 1 \neq \underbrace{|dom(y)|}_{=2} \times \underbrace{|dom(z)|}_{=1}$$

$\Downarrow$

$$a \in dom(x)$$

currTable	1	1	1	0

$\cap$

supports[x,b]	1	1	0	0

$=$

intersection	1	1	0	0

$\Downarrow$

$$\texttt{bitCount} = 2 = \underbrace{|dom(y)|}_{=2} \times \underbrace{|dom(z)|}_{=1}$$

$\Downarrow$

$$dom(x) \backslash \{b\}$$

Goal of the propagation

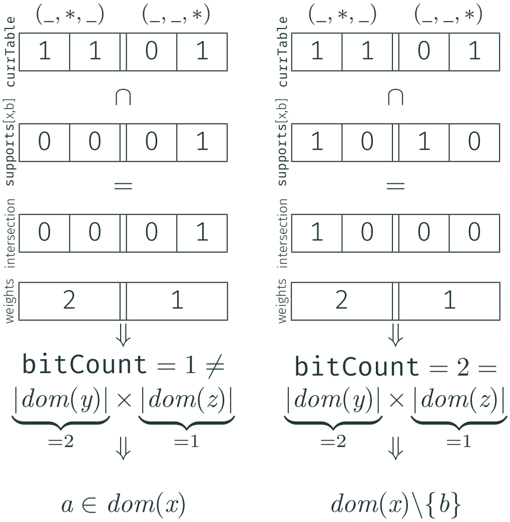
Remove unsupported values

Algorithm: Propagate()

foreach variable $x \in$ `scp` **do**
  $T \leftarrow \prod_{y \in scp: y \neq x} |dom(y)|$;
  foreach value $a \in dom(x)$ **do**
    $S \leftarrow$ `currTable` & `supports`$[x, a]$;
    if `bitCount`$(S) == T$ **then**
      $dom(x) \leftarrow dom(x) \setminus \{a\}$;

CT^*_{neg}: CT* WITH MODIFIED PROPAGATION

UCLouvain



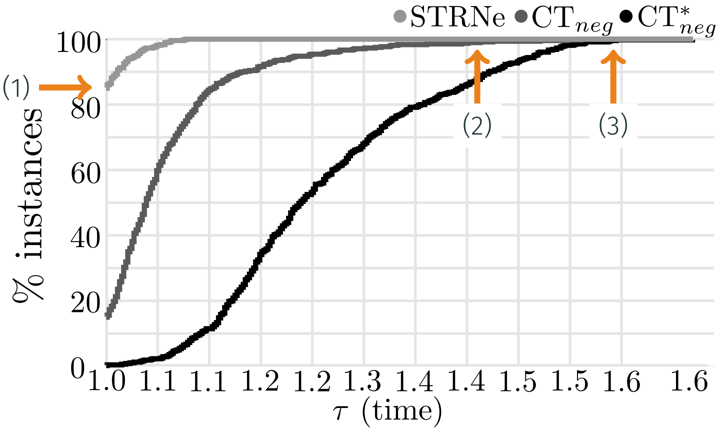
Goal of the propagation

Remove unsupported values

Algorithm: Propagate()

---

foreach variable $x \in$ `scp` **do**
- $T \leftarrow \prod_{y \in scp : y \neq x} |dom(y)|$;
- **foreach** value $a \in dom(x)$ **do**
  - $S \leftarrow$ `currTable` & `supports`$[x, a]$;
  - **if** `weightedBitCount`$(S) == T$ **then**
    - $dom(x) \leftarrow dom(x) \setminus \{a\}$;

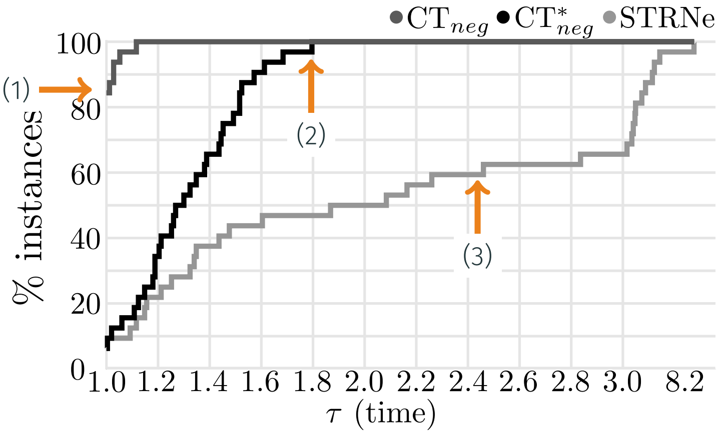


Complexity of CT_{neg}:
CT's complexity × complexity of bitcount ($\mathcal{O}(rd\frac{t}{w}k)$)

Complexity of CT^*_{neg}:
CT_{neg}'s complexity, using t' the # of tuples with dummy ones ($\mathcal{O}(rd\frac{t'}{w}k)$)

(1) STRNe best 85%
(2) CT_{neg} requiers max $1.4\times$
(3) CT_{neg} requiers max $1.6\times$

600 instances (with high number of solution), 20 variables, domain size from 5 to 7, 40 random tables by instances, arity of 6 or 7, tightness [0.5%;2%], 1, 5, 10 or 20 % of short tuples

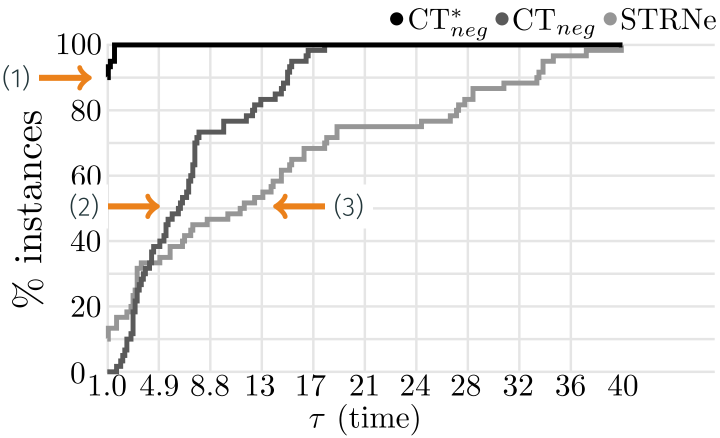


Complexity of CT_{neg}:
CT's complexity $\times$ complexity of bitcount ($\mathcal{O}(rd\frac{t}{w}k)$)

Complexity of CT^*_{neg}:
CT_{neg}'s complexity, using t' the # of tuples with dummy ones ($\mathcal{O}(rd\frac{t'}{w}k)$)

(1) CT_{neg} best 85% of the time
(2) $< 1.8\times$ for CT^*_{neg}
(3) STRNe requiers $> 2.5\times$ on 40% of instances

45 instances (with low number of solutions), 10 variables, domain size of 5, 40 random tables by instances, arity of 6, tightness 10,... 90%, no short tuples



Complexity of CT_{neg}: CT's complexity × complexity of bitcount ($\mathcal{O}(rd\frac{t}{w}k)$)

Complexity of CT^*_{neg}: CT_{neg}'s complexity, using t' the # of tuples with dummy ones ($\mathcal{O}(rd\frac{t'}{w}k)$)

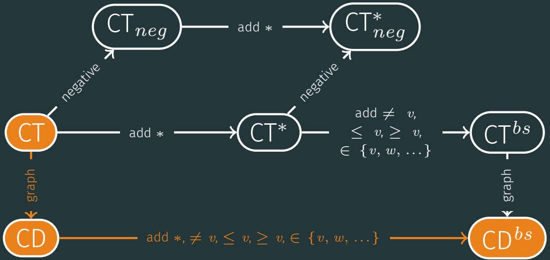
(1) CT^*_{neg} best 90% of the time

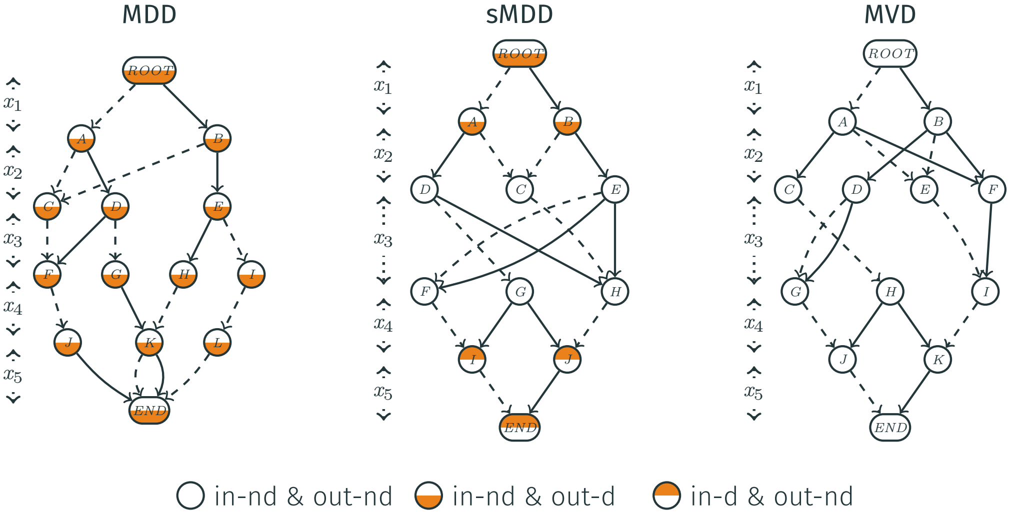
(2) $> 6\times$ for 50%

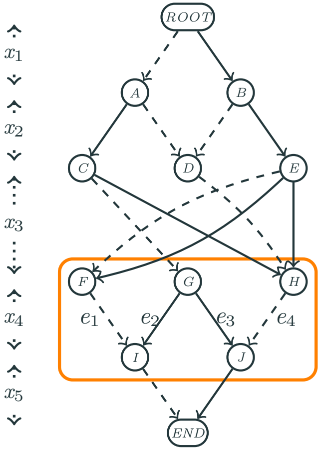
(3) $> 11\times$ for 50%

100 instances (with low number of solutions), 3 variables, domain size of 100, 40 random tables by instances, arity of 3, tightness [0.5;2%], 5, 10 or 20 % of short tuples

3RD DIMENSION: FROM TABLES TO GRAPHS




MDD
sMDD
MVD
ROOT
END
x_1
x_2
x_3
x_4
x_5
A
B
C
D
E
F
G
H
I
J
K
L
in-nd & out-nd
in-nd & out-d
in-d & out-nd



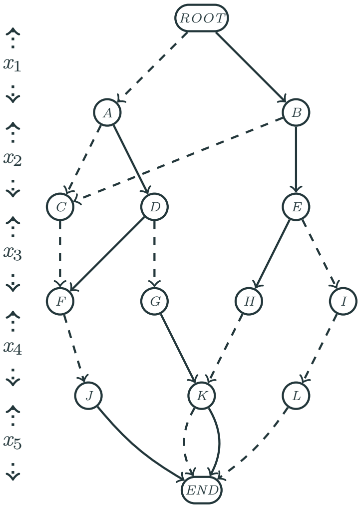
Name	Set	Bit-set
currArcs[x_4]	$\{e_1, e_2, e_3, e_4\}$	[1 1 1 1]
supports[x_4,0]	$\{e_1, \cancel{e_2}, \cancel{e_3}, e_4\}$	[1 0 0 1]
arcsT[G,x_4]	$\{\cancel{e_1}, e_2, e_3, \cancel{e_4}\}$	[0 1 1 0]
arcsH[x_4,I]	$\{e_1, e_2, \cancel{e_3}, \cancel{e_4}\}$	[1 1 0 0]

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

```
foreach variable x ∈ scp do
    mask[x] ← 0;
updateMasks();
propagateDown(x_1,false);
propagateUp(x_r,false);
```



currArcs[x_1]
[1 1]

currArcs[x_2]
[1 1 1 1]

currArcs[x_3]
[1 1 1 1 1]

currArcs[x_4]
[1 1 1 1]

currArcs[x_5]
[1 1 1 1]

UCLouvain

Goal of the update

Remove invalid edges from `currArcs`

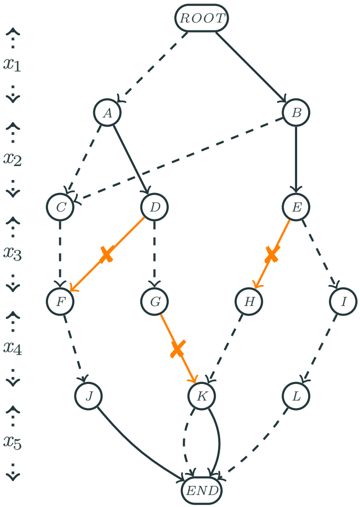
Algorithm: Update(x)

foreach variable $x \in$ `scp` **do**
    `mask`$[x] \leftarrow 0$;

updateMasks();

propagateDown(x_1,false);

propagateUp(x_r,false);



`currArcs`$[x_1]$
[1 1]

`currArcs`$[x_2]$
[1 1 1 1]

`currArcs`$[x_3]$
[1 1 1 1 1]

`currArcs`$[x_4]$
[1 1 1 1]

`currArcs`$[x_5]$
[1 1 1 1]

1st step
Direct removal

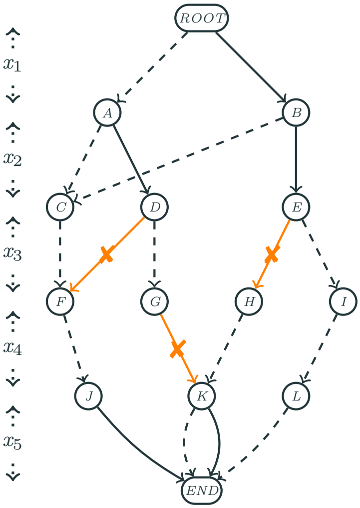
UCLouvain

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

1. **foreach** variable $x \in$ `scp` **do**
2.   `mask`$[x] \leftarrow 0$;
3. updateMasks();
4. propagateDown(x_1,false);
5. propagateUp(x_r,false);



currArcs$[x_1]$
[1 1]

currArcs$[x_2]$
[1 1 1 1]

currArcs$[x_3]$
[1 0 1 0 1]

currArcs$[x_4]$
[1 0 1 1]

currArcs$[x_5]$
[1 1 1 1]

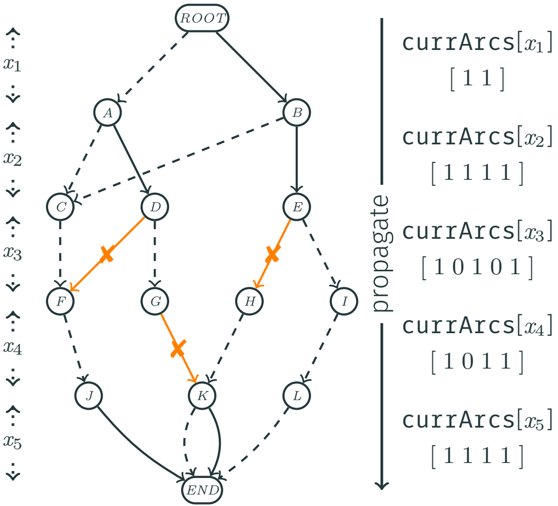
1st step
Direct removal

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

```
foreach variable x ∈ scp do
    mask[x] ← 0;
updateMasks();
propagateDown(x_1,false);
propagateUp(x_r,false);
```



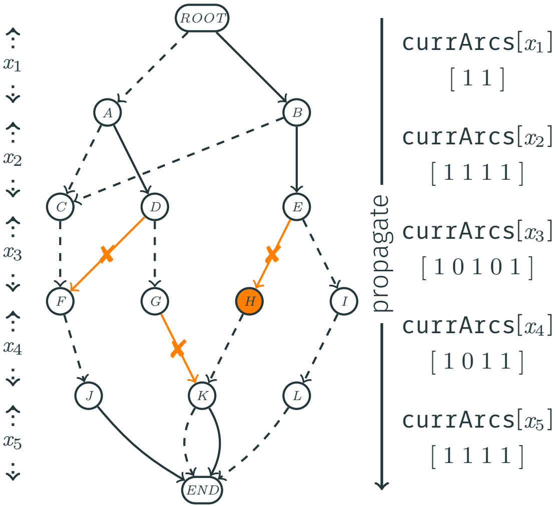
2nd step
Top down

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

```
1 foreach variable x ∈ scp do
2     mask[x] ← 0;
3 updateMasks();
4 propagateDown(x_1,false);
5 propagateUp(x_r,false);
```



2nd step
Top down

UCLouvain

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

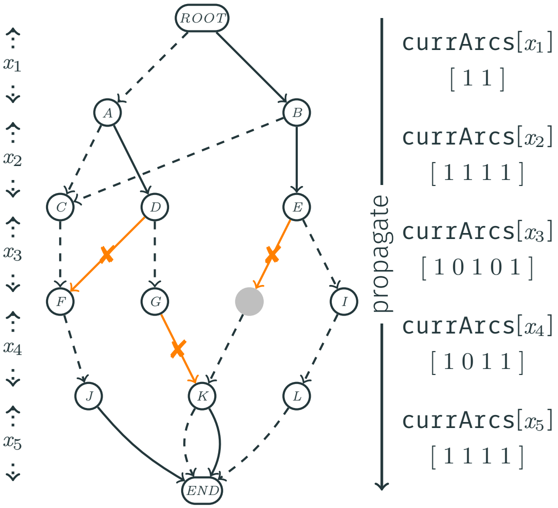
1 **foreach** variable $x \in$ `scp` **do**

2 `mask`$[x] \leftarrow 0$;

3 updateMasks();

4 propagateDown(x_1,false);

5 propagateUp(x_r,false);



2nd step
Top down

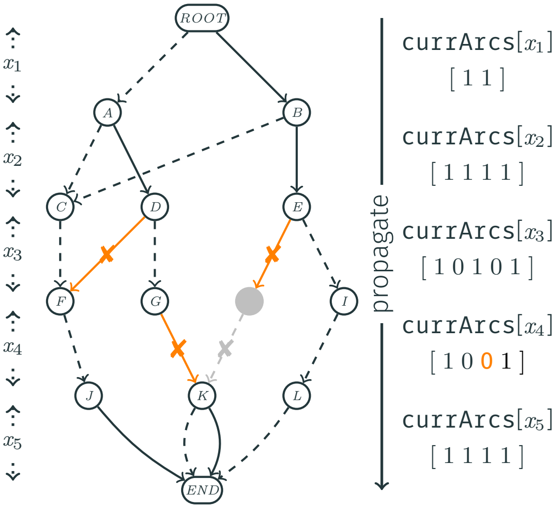
UCLouvain

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

```
foreach variable x ∈ scp do
    mask[x] ← 0;
updateMasks();
propagateDown(x_1,false);
propagateUp(x_r,false);
```



2nd step
Top down

UCLouvain

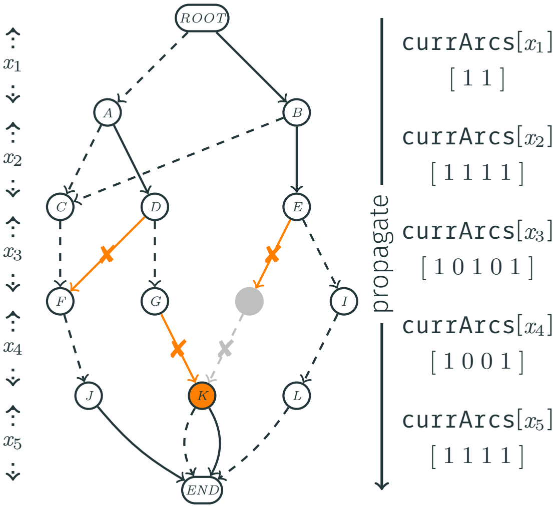
Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

1 **foreach** variable $x \in$ `scp` **do**
2   `mask`$[x] \leftarrow 0$;
3 updateMasks();
4 propagateDown(x_1,false);
5 propagateUp(x_r,false);

2nd step
Top down

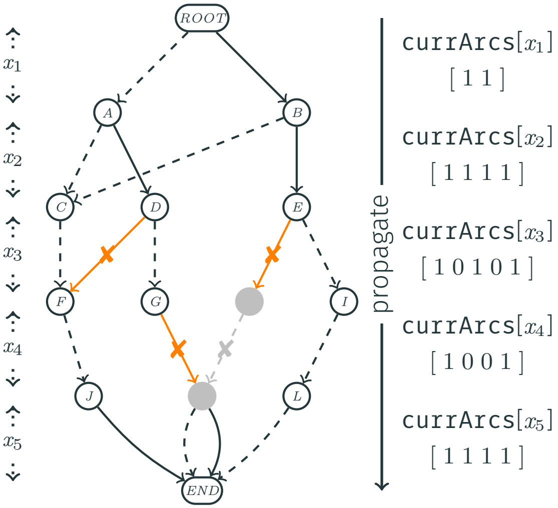


Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

1 **foreach** variable $x \in$ `scp` **do**
2   `mask`$[x] \leftarrow 0$;
3 updateMasks();
4 propagateDown(x_1,false);
5 propagateUp(x_r,false);



2nd step
Top down

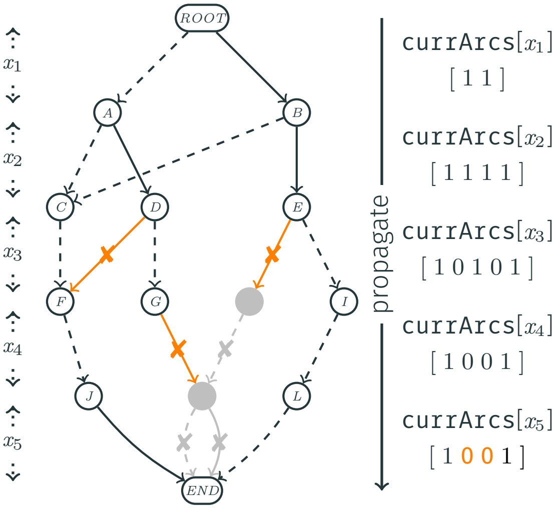
UCLouvain

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

```
foreach variable x ∈ scp do
    mask[x] ← 0;
updateMasks();
propagateDown(x_1,false);
propagateUp(x_r,false);
```



2nd step
Top down

UCLouvain

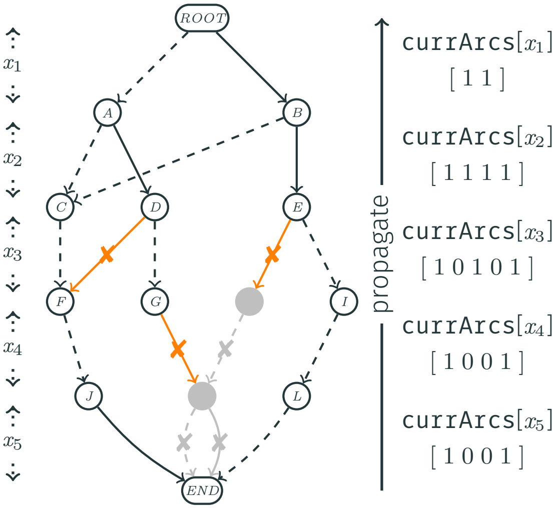
Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

1 **foreach** variable $x \in$ `scp` **do**
2 `mask`$[x] \leftarrow 0$;
3 updateMasks();
4 propagateDown(x_1,false);
5 propagateUp(x_r,false);

3rd step
Bottom up

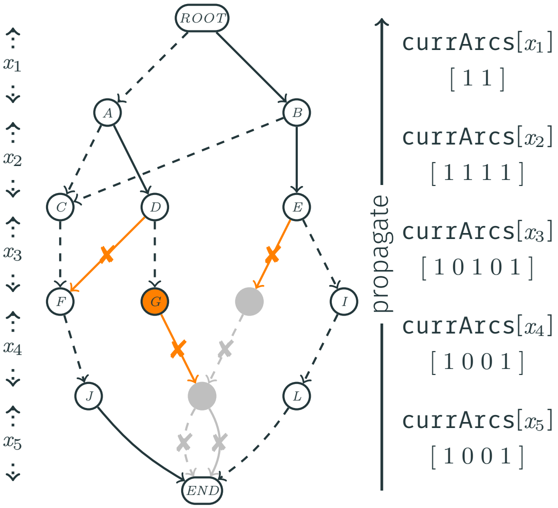


Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

1 **foreach** variable $x \in$ **scp do**
2 mask[x] $\leftarrow$ 0;
3 updateMasks();
4 propagateDown(x_1,false);
5 propagateUp(x_r,false);



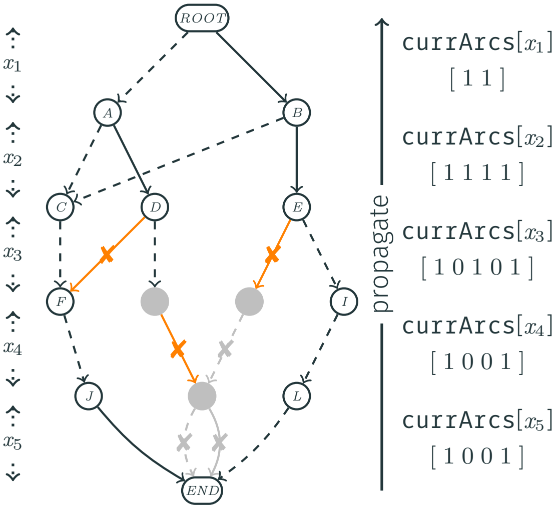
3rd step
Bottom up

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

```
foreach variable x ∈ scp do
    mask[x] ← 0;
updateMasks();
propagateDown(x_1,false);
propagateUp(x_r,false);
```



3rd step
Bottom up

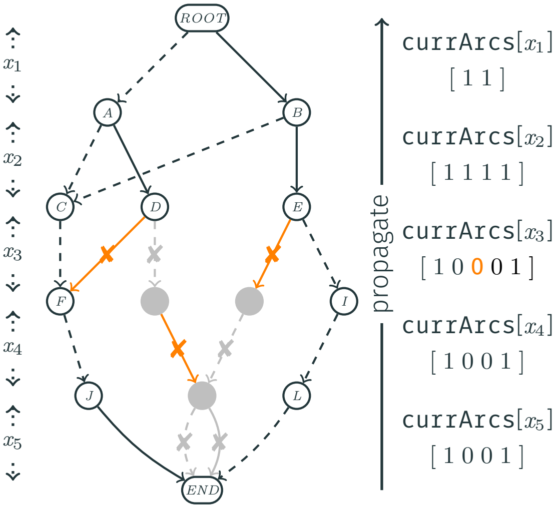
UCLouvain

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

foreach variable $x \in$ `scp` **do**
    `mask`$[x] \leftarrow 0$;
updateMasks();
propagateDown(x_1,false);
propagateUp(x_r,false);



3rd step
Bottom up

UCLouvain

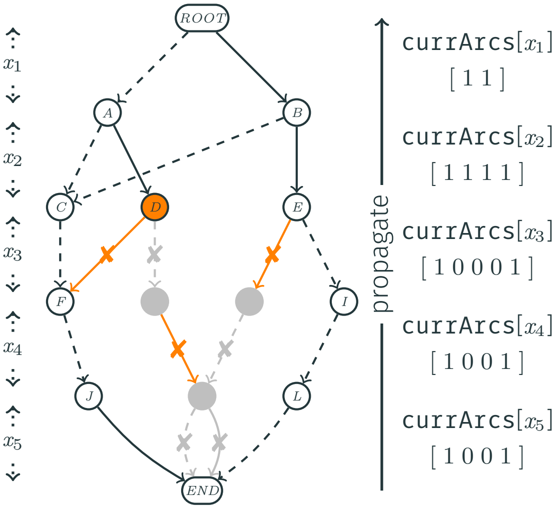
Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

1. **foreach** variable $x \in$ `scp` **do**
2. `mask`$[x] \leftarrow 0$;
3. updateMasks();
4. propagateDown(x_1,false);
5. propagateUp(x_r,false);

3rd step
Bottom up



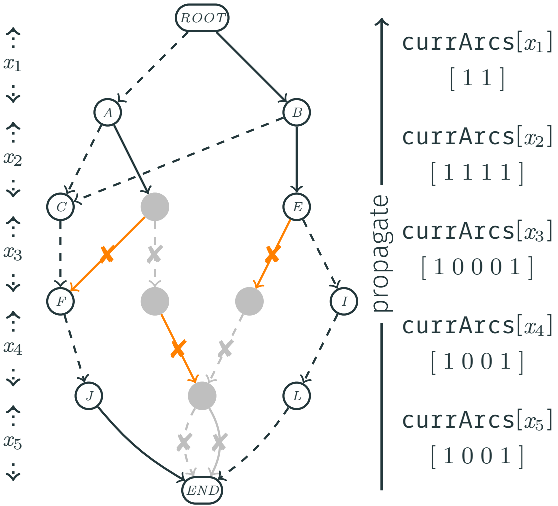
UCLouvain

Goal of the update

Remove invalid edges from `currArcs`

Algorithm: Update(x)

foreach variable $x \in$ `scp` **do**
  `mask`$[x] \leftarrow 0$;
updateMasks();
propagateDown(x_1,false);
propagateUp(x_r,false);



3rd step
Bottom up

COMPACT-DIAGRAM: UPDATE

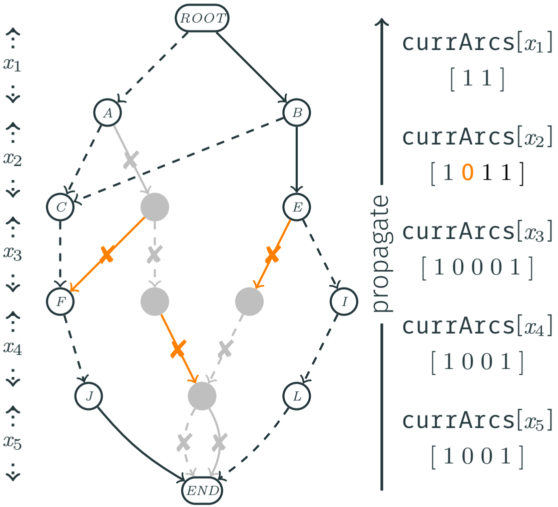
UCLouvain

Goal of the update

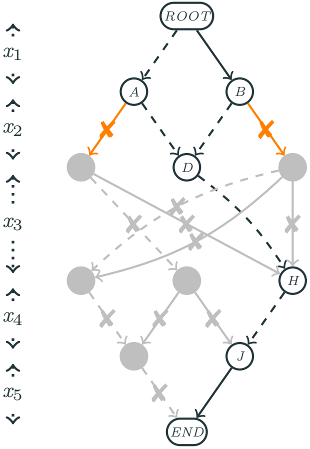
Remove invalid edges from `currArcs`

Algorithm: Update(x)

foreach variable $x \in$ `scp` **do**
  `mask`$[x] \leftarrow 0$;
updateMasks();
propagateDown(x_1,false);
propagateUp(x_r,false);



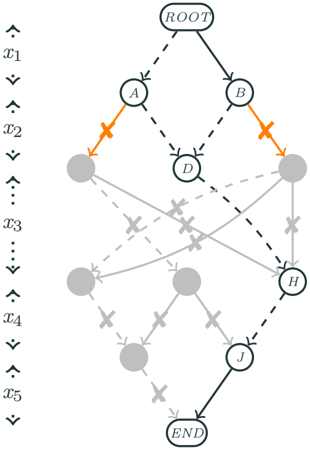
3rd step
Bottom up



$\Delta_{x_2} = \{1\}$

x_1	x_2	x_3	x_4	x_5
$\{0,1\}$	$\{0\}$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$

(x,v)	currArcs[x]	supports[x,v]	$\cap$
$(x_1,0)$	11	10	10
$(x_1,1)$	11	01	01
$(x_3,0)$	001000	101100	001000
$(x_3,1)$	001000	010011	000000
$(x_4,0)$	0001	1001	0001
$(x_4,1)$	0001	0110	0000
$(x_5,0)$	01	10	00
$(x_5,1)$	01	01	01

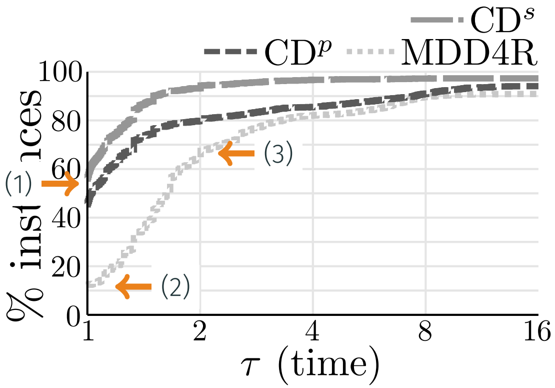


$\Delta_{x_2} = \{1\}$

x_1	x_2	x_3	x_4	x_5
$\{0, 1\}$	$\{0\}$	$\{0, \not{1}\}$	$\{0, \not{1}\}$	$\{\not{0}, 1\}$

(x,v)	currArcs[x]	supports[x,v]	$\cap$
$(x_1,0)$	11	10	10
$(x_1,1)$	11	01	01
$(x_3,0)$	001000	101100	001000
$(x_3,1)$	001000	010011	000000
$(x_4,0)$	0001	1001	0001
$(x_4,1)$	0001	0110	0000
$(x_5,0)$	01	10	00
$(x_5,1)$	01	01	01

UCLouvain



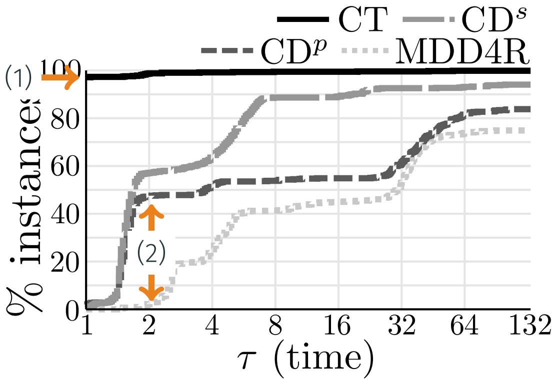
Complexity of CD: similar to CT ($\mathcal{O}(\max(n, d) r \frac{a}{w})$)

(1) CD gives best results, sMDDs better than MDDs

(2) MDD4R only best on 12%

(3) MDD4R requires $> 2\times$ on 35%

XCSP3 instances with only tables, transformed into sMDD or MDD instances only



Complexity of CD: similar to CT ($\mathcal{O}(\max(n, d) r \frac{a}{w})$)

(1) CT still best 95%

(2) Reduction of the gap: CD^s requires $< 2\times$ for 60%, CD^p requires $< 2\times$ for 50%, while MDD4R requires $< 2\times$ for 5%

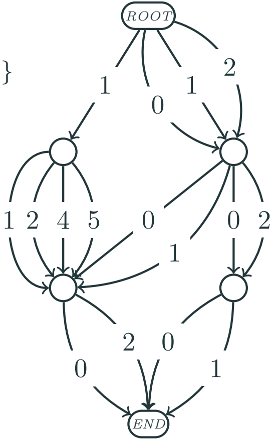
XCSP3 instances with only tables, transformed into sMDD or MDD instances only

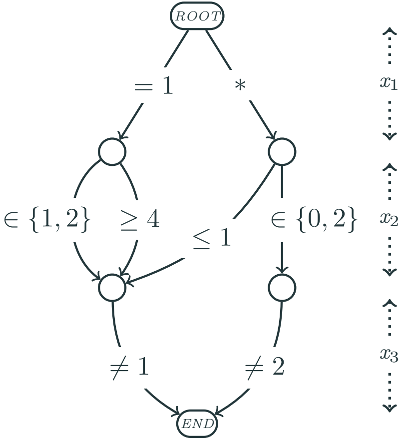
Domains

$x_0 : \{0, 1, 2\}$

$x_1 : \{0, 1, 2, 3, 4, 5\}$

$x_2 : \{0, 1, 2\}$





Algorithm: Direct removal part of the update

```
if layer without ∈ then
    if |Δ(x)| < |dom(x)| then
        Incremental update (=, ≠, ∗);
        Lower bound update (≤);
        Upper bound update (≥);
    else
        Reset update (=, ≠, ∗, ≤, ≥, ∈);
else
    Reset update (=, ≠, ∗, ≤, ≥, ∈);
```

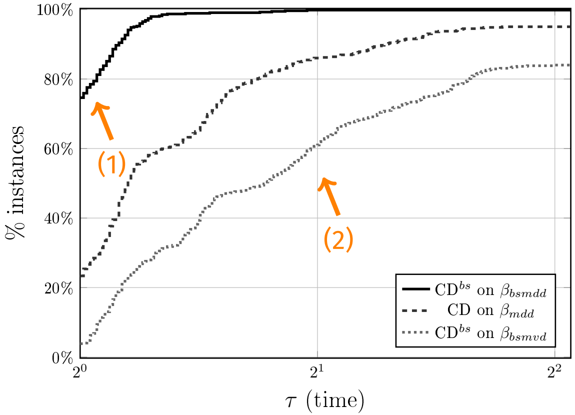
word 0				word 1				word 2			
w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	-	-
$=$	$\leq$	$\geq$	$\in$	$\neq$	$>$	$\notin$	$<$	$\neq$	$*$		

$$\Downarrow$$

word 0				word 1				word 2				word 3			
w_0	w_4	w_8	w_9	w_3	w_6	-	-	w_1	w_7	-	-	w_2	w_5	-	-
$=$	$\neq$	$\neq$	$*$	$\in$	$\notin$			$\leq$	$<$			$\geq$	$>$		

word 0				word 1				word 2				word 3			
w_0	w_4	w_8	w_9	w_3	w_6	-	-	w_1	w_7	-	-	w_2	w_5	-	-
$=$	$\neq$	$\neq$	$*$	$\in$	$\notin$			$\leq$	$<$			$\geq$	$>$		

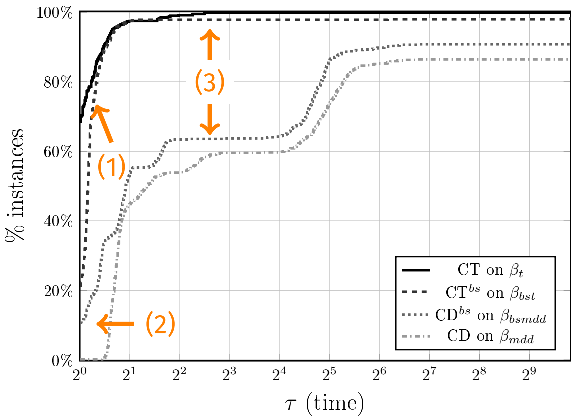
- word 0 → Incremental or Reset update (depending if $|\Delta(x)| < |dom(x)|$)
- word 1 → Reset update
- word 2 → Lower bound update
- word 3 → Upper bound update



(1) CD^{bs} on bs-MDDs (fewer arcs) best 80% of the time

(2) CD^{bs} on bs-MVDs (more nodes) worst

XCSP3 instances with only tables, transformed into MDD and bs-MDD instances only



(1) CT and CT^{bs} still dominating

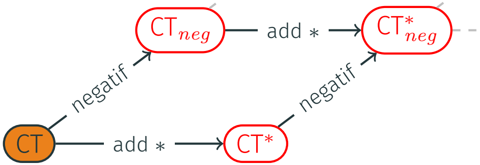
(2) CD^{bs} becomes efficient when compression is high

(3) gap reduced

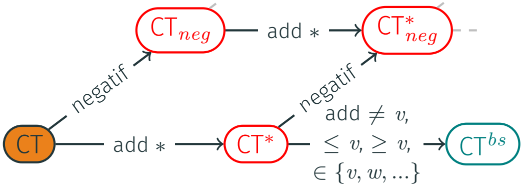
XCSP3 instances with only tables, transformed into bs-table, MDD and bs-MDD instances only

CONCLUSION

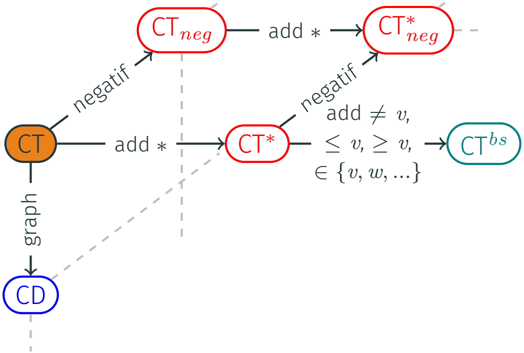
CT



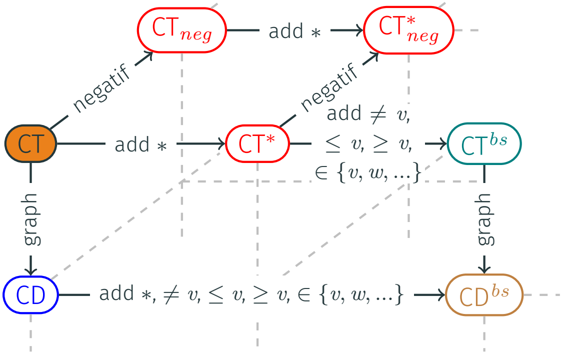
H. Verhaeghe, C. Lecoutre and P. Schaus. **Extending Compact-Table to Negative and Short Tables.** AAAI17



H. Verhaeghe, C. Lecoutre and P. Schaus. **Extending Compact-Table to Negative and Short Tables.** AAAI17
H. Verhaeghe, C. Lecoutre, Y. Deville and P. Schaus. **Extending Compact-Table to Basic Smart Tables.** CP2017



H. Verhaeghe, C. Lecoutre and P. Schaus. **Extending Compact-Table to Negative and Short Tables.** AAAI17
H. Verhaeghe, C. Lecoutre, Y. Deville and P. Schaus. **Extending Compact-Table to Basic Smart Tables.** CP2017
H. Verhaeghe, C. Lecoutre, P. Schaus. **Compact-MDD: Efficiently filtering (s)mdd constraints with Reversible Sparse Bit-Sets.** IJCAI18



H. Verhaeghe, C. Lecoutre and P. Schaus. **Extending Compact-Table to Negative and Short Tables.** AAAI17
H. Verhaeghe, C. Lecoutre, Y. Deville and P. Schaus. **Extending Compact-Table to Basic Smart Tables.** CP2017
H. Verhaeghe, C. Lecoutre, P. Schaus. **Compact-MDD: Efficiently filtering (s)mdd constraints with Reversible Sparse Bit-Sets.** IJCAI18
H. Verhaeghe, C. Lecoutre, P. Schaus. **Extending Compact-Diagram to Basic Smart Multi-Valued Variable Diagrams.** CPAIOR19

- Increasing non-determinism in diagrams
- Closing the gap between diagrams and tables propagators
- Direct use of compressed tables and non-deterministic diagrams in applications