"CHOCO with a trace": Set-labeled Diagrams for CSP Compilation

Alexandre Niveau[1] Hélène Fargier[2] Cédric Pralet[3]

[1]CRIL, Lens, France — niveau@cril.fr

[2]IRIT, Toulouse, France — fargier@irit.fr

[3]Onera, Toulouse, France — cpralet@onera.fr

STAIRS
August 27–28 2012

Introductory Example

- a product configuration problem: customized tee-shirts. [Had08]

[Had08] T. Hadzic, E.R. Hansen, and Barry B. O'Sullivan. "On Automata, MDDs and BDDs in Constraint Satisfaction". In: ECAI Workshop on Inference methods based on Graphical Structures of Knowledge (WIGSK). 2008

Introductory Example

- a product configuration problem: customized tee-shirts. [Had08]
- parameters:
  - print – "Men in Black" (MiB) or "Save the Whales" (StW)
  - color – black, blue, or red
  - size – small, medium, or large
  - sleeves – with or without

[Had08] T. Hadzic, E.R. Hansen, and Barry B. O'Sullivan. "On Automata, MDDs and BDDs in Constraint Satisfaction". In: ECAI Workshop on Inference methods based on Graphical Structures of Knowledge (WIGSK). 2008

Introductory Example

- a product configuration problem: customized tee-shirts. [Had08]
- parameters:
  - print – "Men in Black" (MiB) or "Save the Whales" (StW)
  - color – black, blue, or red
  - size – small, medium, or large
  - sleeves – with or without
- rules:
  - "MiB" tee-shirts must be black
  - "StW" picture does not fit on small-sized tee-shirts
  - large-sized tee-shirts can't be sleeveless

[Had08] T. Hadzic, E.R. Hansen, and Barry B. O'Sullivan. "On Automata, MDDs and BDDs in Constraint Satisfaction". In: ECAI Workshop on Inference methods based on Graphical Structures of Knowledge (WIGSK). 2008

Introductory Example

- a product configuration problem: customized tee-shirts. [Had08]
- parameters:
  - print – "Men in Black" (MiB) or "Save the Whales" (StW)
  - color – black, blue, or red
  - size – small, medium, or large
  - sleeves – with or without
- rules:
  - "MiB" tee-shirts must be black
  - "StW" picture does not fit on small-sized tee-shirts
  - large-sized tee-shirts can't be sleeveless
- the program must be able to tell whether each choice respects the rules

---

[Had08] T. Hadzic, E.R. Hansen, and Barry B. O'Sullivan. "On Automata, MDDs and BDDs in Constraint Satisfaction". In: ECAI Workshop on Inference methods based on Graphical Structures of Knowledge (WIGSK). 2008

Problem

- configurable product $\rightarrow$ Constraint Satisfaction Problem (CSP)

$$\begin{cases} print = \text{MiB} & \rightarrow & color = \text{black} \\ print = \text{StW} & \rightarrow & size > \text{small} \\ size \leqslant \text{med} & \vee & sleeves = \text{with} \end{cases}$$

  - each variable corresponds to a choice
  - each solution is a possible configuration

Problem

- configurable product $\rightarrow$ Constraint Satisfaction Problem (CSP)

$$\begin{cases} print = \text{MiB} & \rightarrow & color = \text{black} \\ print = \text{StW} & \rightarrow & size > \text{small} \\ size \leqslant \text{med} & \vee & sleeves = \text{with} \end{cases}$$

  - each variable corresponds to a choice
  - each solution is a possible configuration

- configuration process:
  - each choice fixes a value for some variable
  - is the CSP still consistent?

Problem

- configurable product $\rightarrow$ Constraint Satisfaction Problem (CSP)

$$\begin{cases} print = \text{MiB} & \rightarrow & color = \text{black} \\ print = \text{StW} & \rightarrow & size > \text{small} \\ size \leqslant \text{med} & \vee & sleeves = \text{with} \end{cases}$$

  - each variable corresponds to a choice
  - each solution is a possible configuration
- configuration process:
  - each choice fixes a value for some variable
  - is the CSP still consistent?
- NP-complete problem... but the user doesn't want to wait too long after each choice

A Solution: Knowledge Compilation

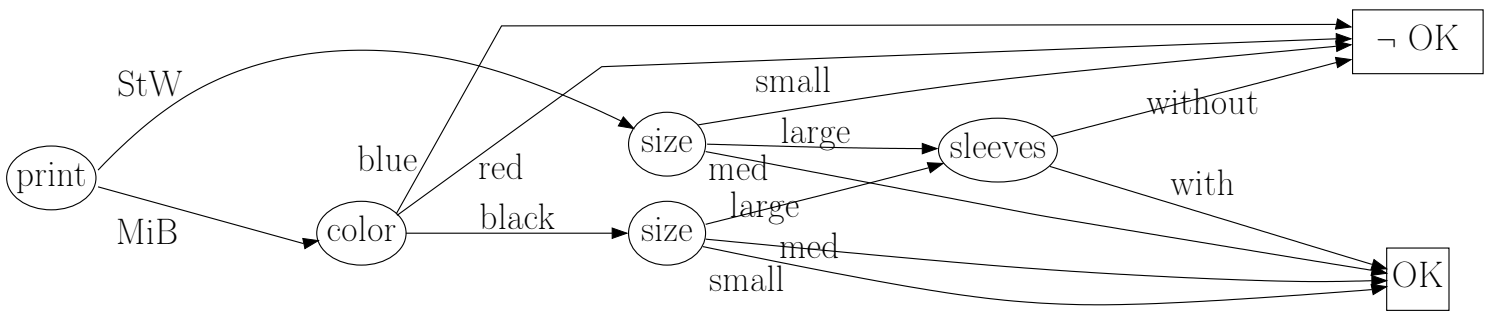
- the CSP is a fixed part of the problem

[Sri90] Arvind Srinivasan, Timothy Kam, Sharad Malik, and Robert K. Brayton. "Algorithms for Discrete Function Manipulation". In: ICCAD. Nov. 1990, pp. 92–95

A Solution: Knowledge Compilation

- the CSP is a fixed part of the problem

→ compile this CSP as a Multivalued Decision Diagram (MDD) [Sri90]

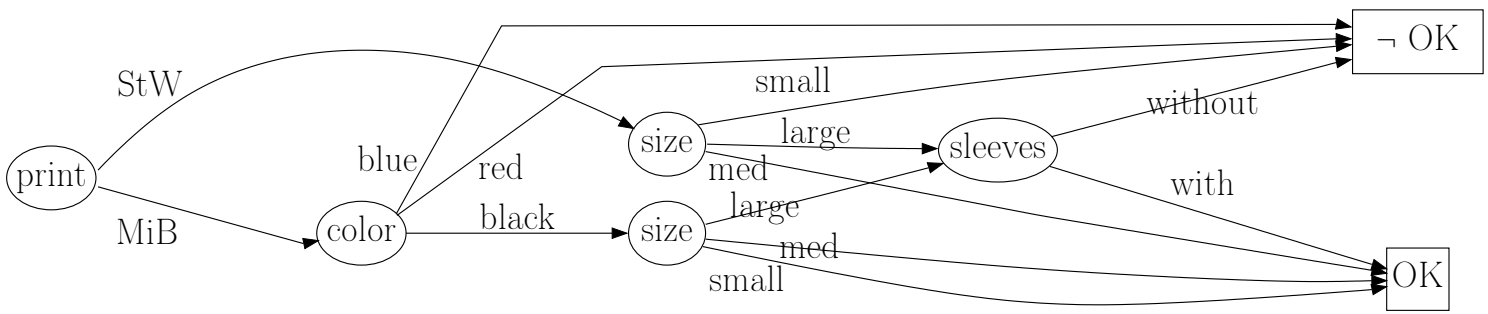


[Sri90] Arvind Srinivasan, Timothy Kam, Sharad Malik, and Robert K. Brayton. "Algorithms for Discrete Function Manipulation". In: ICCAD. Nov. 1990, pp. 92–95

A Solution: Knowledge Compilation

- the CSP is a fixed part of the problem

→ compile this CSP as a Multivalued Decision Diagram (MDD) [Sri90]



- assigning values to variables (conditioning) and checking consistency are tractable operations on MDDs

→ the user's wait is (hopefully) reduced!

[Sri90] Arvind Srinivasan, Timothy Kam, Sharad Malik, and Robert K. Brayton. "Algorithms for Discrete Function Manipulation". In: ICCAD. Nov. 1990, pp. 92–95

Set-labeled Diagrams

- MDDs meet strong requirements: determinism, read-once, ordering
- succinctness loss

  yet, requirements not always necessary

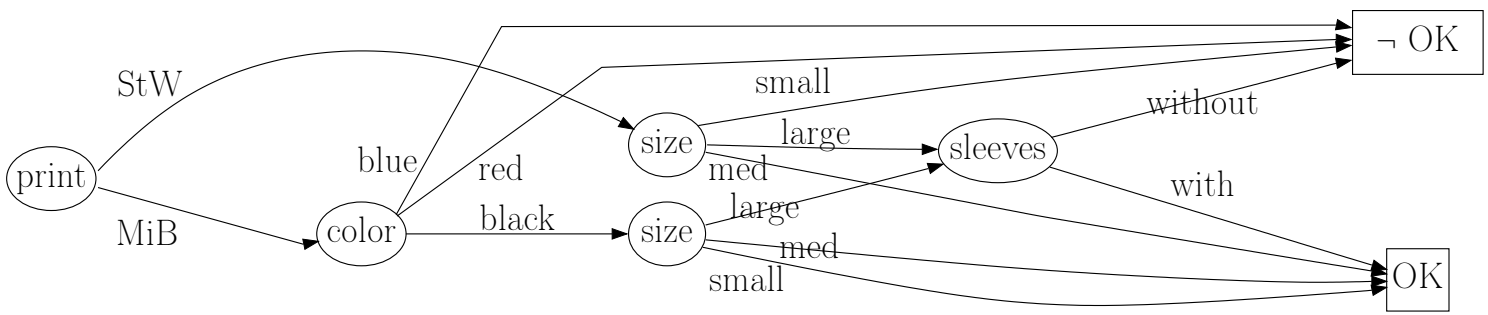
[Niv11] Alexandre Niveau, Hélène Fargier, and Cédric Pralet. "Representing CSPs with Set-labeled Diagrams: A Compilation Map". In: Proc. of the 2nd International Workshop on Graph Structures for Knowledge Representation and Reasoning (GKR). 2011

Set-labeled Diagrams

- MDDs meet strong requirements: determinism, read-once, ordering
- succinctness loss

  yet, requirements not always necessary

→ we use a more general structure: Set-labeled Diagrams (SDs) [Niv11]



- non-deterministic, non-read-once, and non-ordered
- edges labeled by sets of values

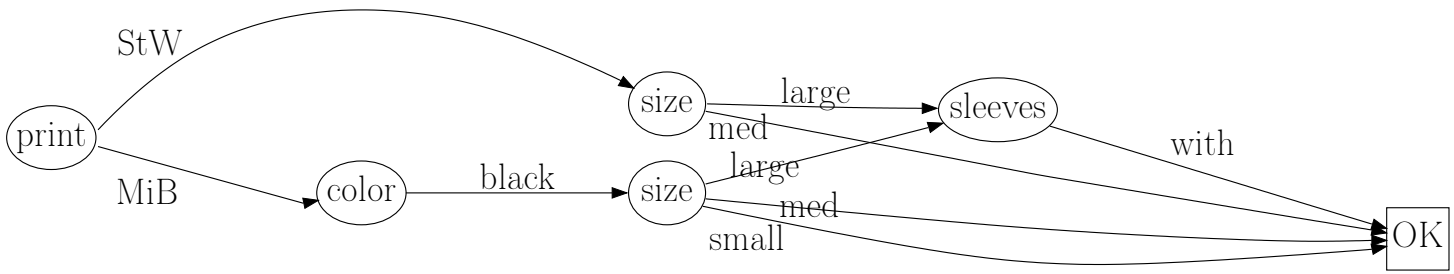
[Niv11] Alexandre Niveau, Hélène Fargier, and Cédric Pralet. "Representing CSPs with Set-labeled Diagrams: A Compilation Map". In: Proc. of the 2nd International Workshop on Graph Structures for Knowledge Representation and Reasoning (GKR). 2011

Set-labeled Diagrams

- MDDs meet strong requirements: determinism, read-once, ordering
- succinctness loss

  yet, requirements not always necessary

→ we use a more general structure: Set-labeled Diagrams (SDs) [Niv11]



- non-deterministic, non-read-once, and non-ordered
- edges labeled by sets of values

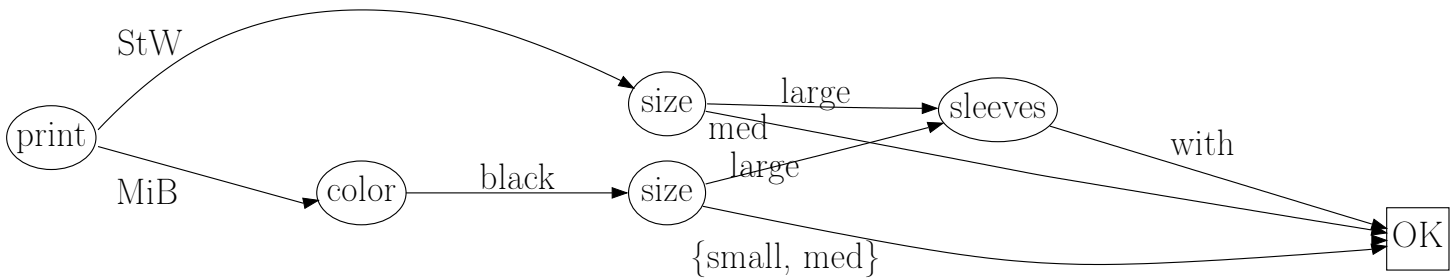
[Niv11] Alexandre Niveau, Hélène Fargier, and Cédric Pralet. "Representing CSPs with Set-labeled Diagrams: A Compilation Map". In: Proc. of the 2nd International Workshop on Graph Structures for Knowledge Representation and Reasoning (GKR). 2011

Set-labeled Diagrams

- MDDs meet strong requirements: determinism, read-once, ordering
- succinctness loss

  yet, requirements not always necessary

→ we use a more general structure: Set-labeled Diagrams (SDs) [Niv11]



- non-deterministic, non-read-once, and non-ordered
- edges labeled by sets of values

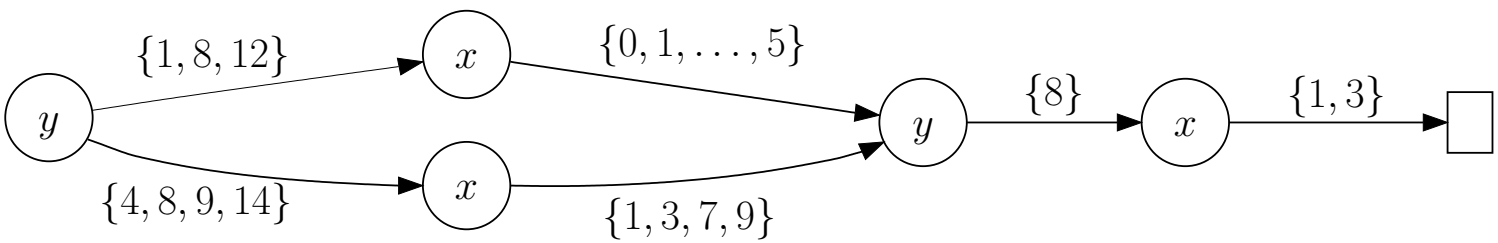
[Niv11] Alexandre Niveau, Hélène Fargier, and Cédric Pralet. "Representing CSPs with Set-labeled Diagrams: A Compilation Map". In: Proc. of the 2nd International Workshop on Graph Structures for Knowledge Representation and Reasoning (GKR). 2011

SD Family: Focusingness

- various kinds of SDs, more or less adapted to various requests

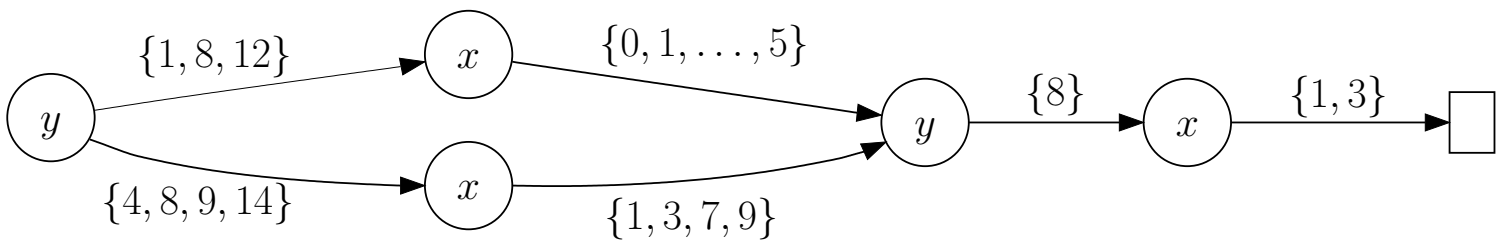
SD Family: Focusingness

- various kinds of SDs, more or less adapted to various requests
- FSDs: focusing SDs (structural restriction made possible by the set labeling)



SD Family: Focusingness

- various kinds of SDs, more or less adapted to various requests
- FSDs: focusing SDs (structural restriction made possible by the set labeling)



$\rightarrow$ conditioning and consistency checking are polytime on FSDs

SD Family

- dFSDs : deterministic FSDs
  $\rightarrow$ succinctness loss, but validity request becomes polytime

SD Family

- dFSDs : deterministic FSDs
  $\rightarrow$ succinctness loss, but validity request becomes polytime

- dRSDs : read-once dFSDs.
  $\rightarrow$ polytime negation and model counting

SD Family

- dFSDs : deterministic FSDs
  $\rightarrow$ succinctness loss, but validity request becomes polytime
- dRSDs : read-once dFSDs.
  $\rightarrow$ polytime negation and model counting
- MDDs : ordered dRSDs.
  $\rightarrow$ succinctness loss, but polytime equivalence, conjunction et disjunction

Compiling a CSP into an SD

- classical method: bottom-up compilation, combination of small SDs representing elementary constraints
- drawback: intermediary graphs, being possibly exponentially larger than the final SD
- another idea is to build SDs by following the search tree of a solver, a la "DPLL with a trace" [Hua05]

[Hua05] J. Huang and A. Darwiche. "DPLL with a Trace: From SAT to Knowledge Compilation". In: IJCAI. 2005, pp. 156–162

CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:

SD :

CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

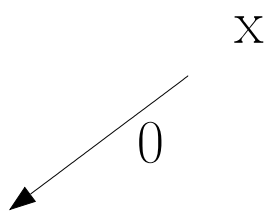
Search tree:

SD :

CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:

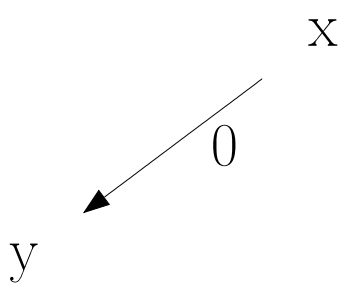


SD :

CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:

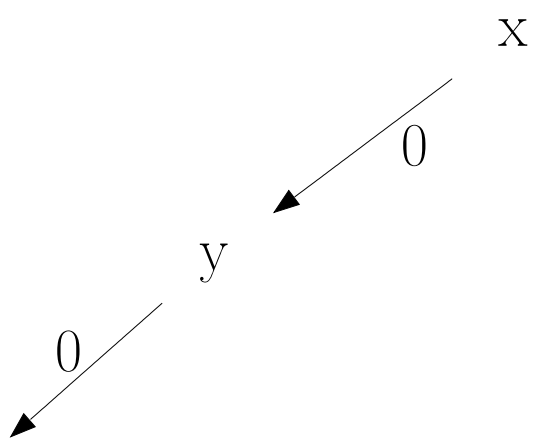


SD :

CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:

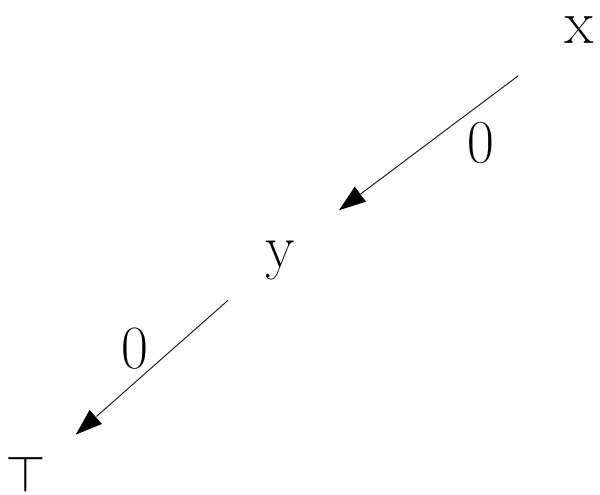


SD :

CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



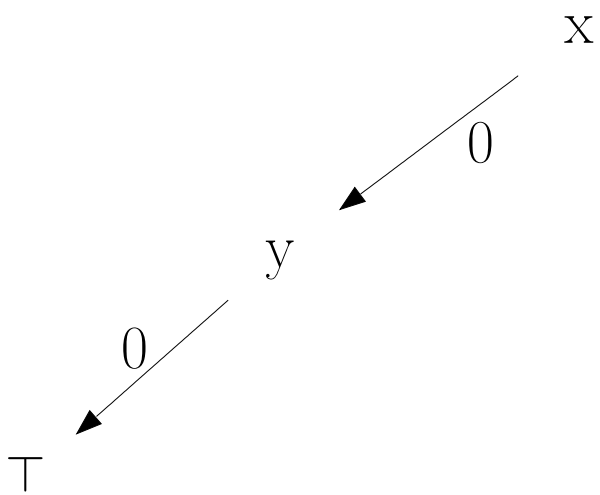
a solution has been found

SD :

CHOCO with a Trace : Step by Step

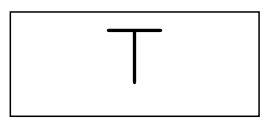
Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



a solution has been found
$\rightarrow$ the sink node is added to the SD

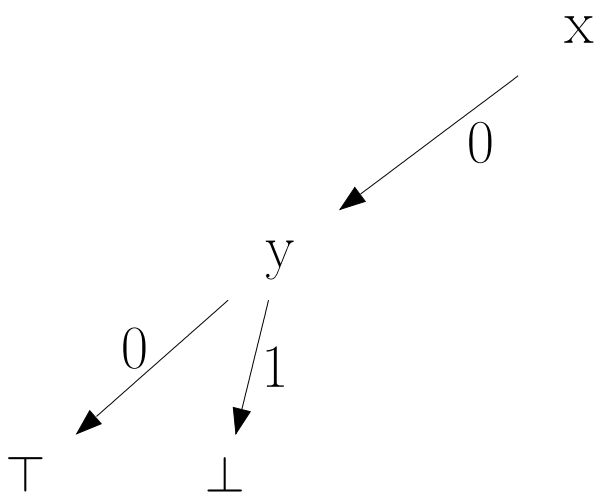
SD :



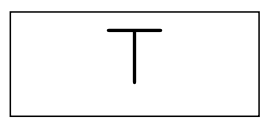
CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



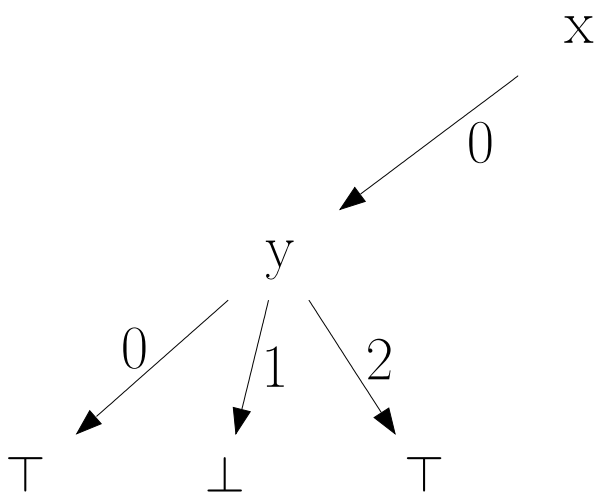
SD :



CHOCO with a Trace : Step by Step

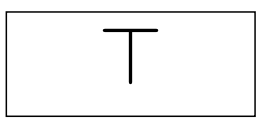
Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



closing node y

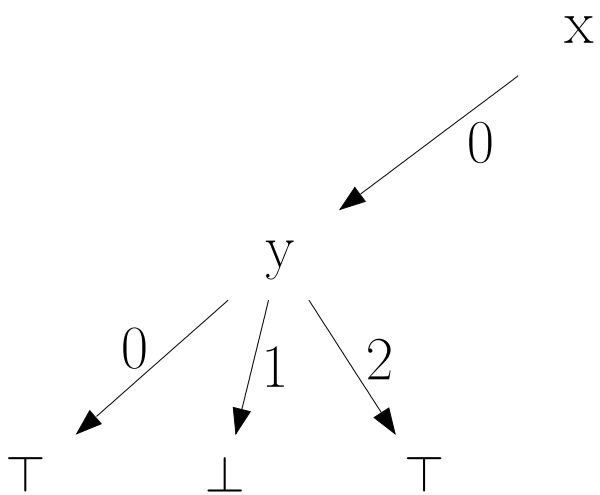
SD :



CHOCO with a Trace : Step by Step

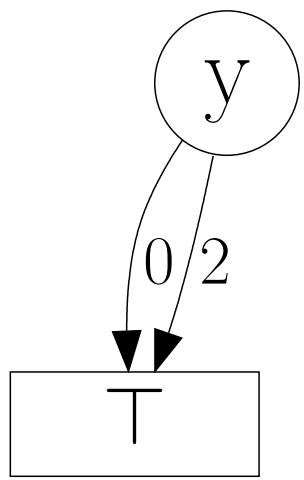
Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



closing node y
$\rightarrow$ it is added to the SD

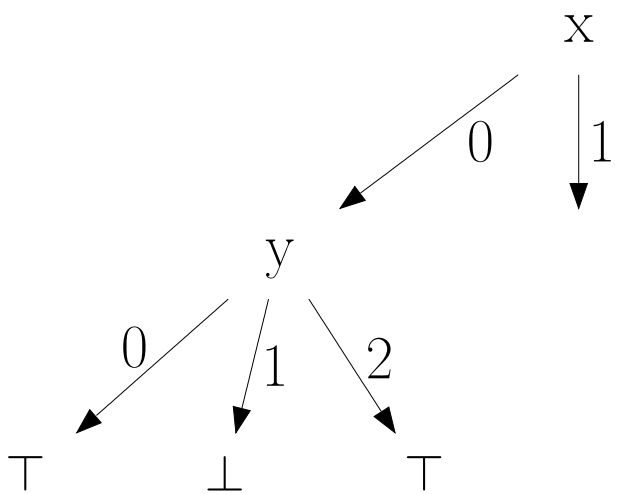
SD :



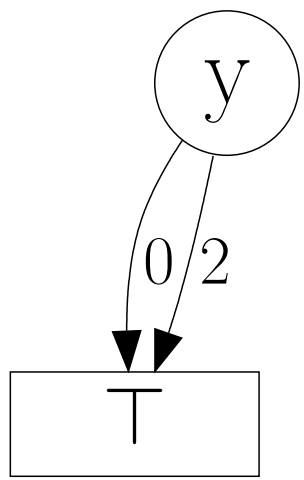
CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



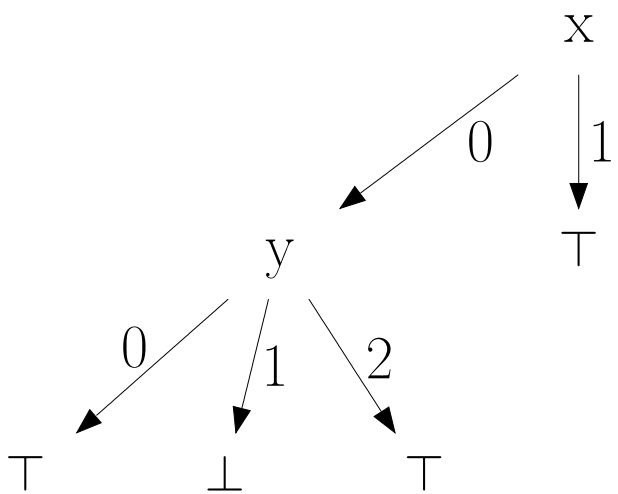
SD :



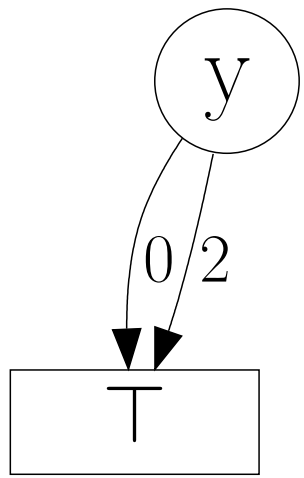
CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



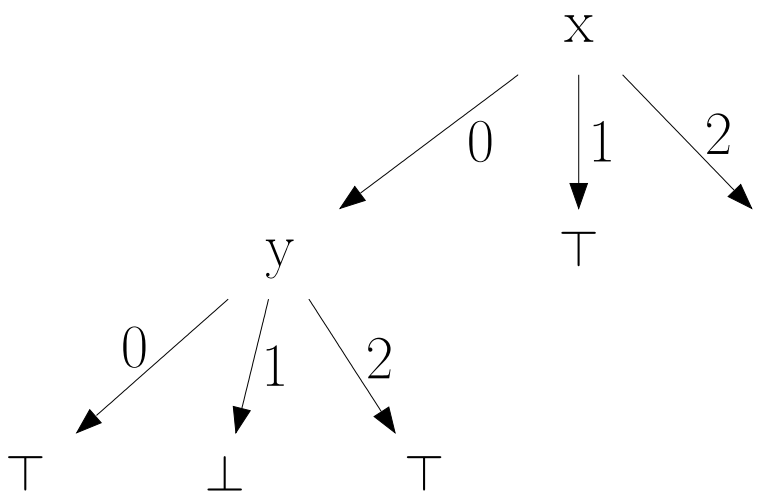
SD :



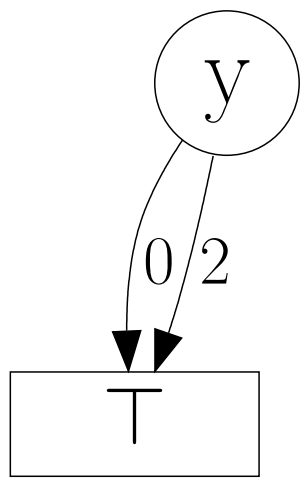
CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



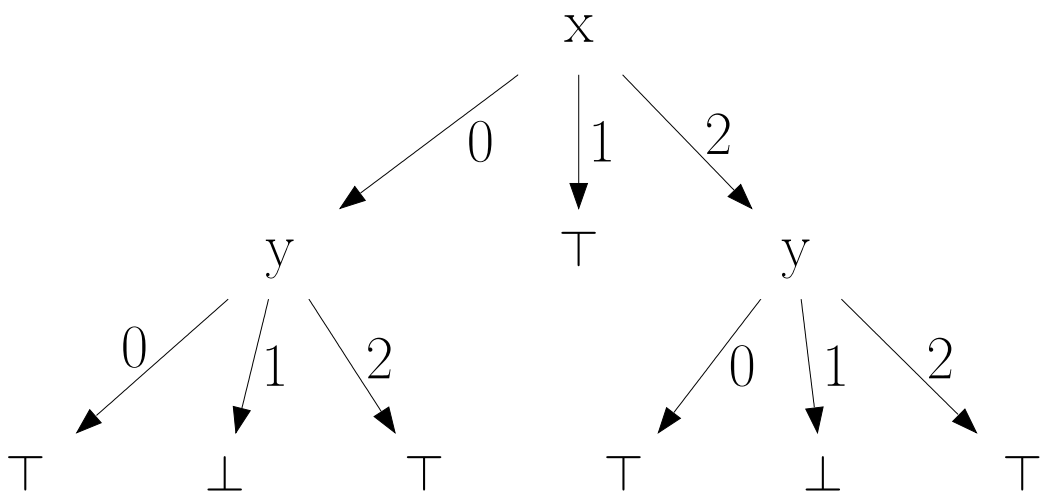
SD :



CHOCO with a Trace : Step by Step

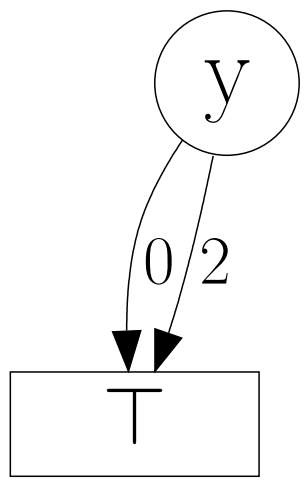
Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



closing node y

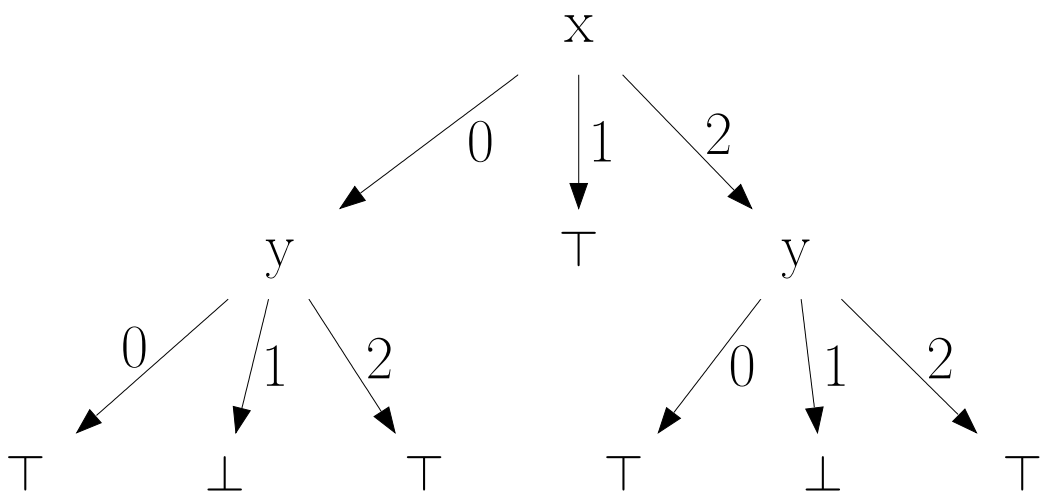
SD :



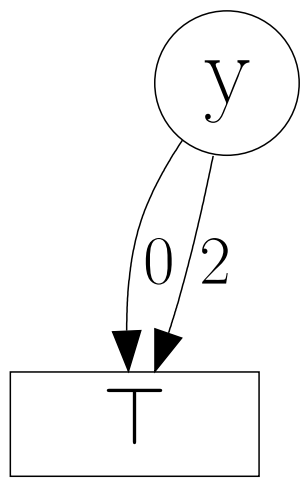
CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



SD :

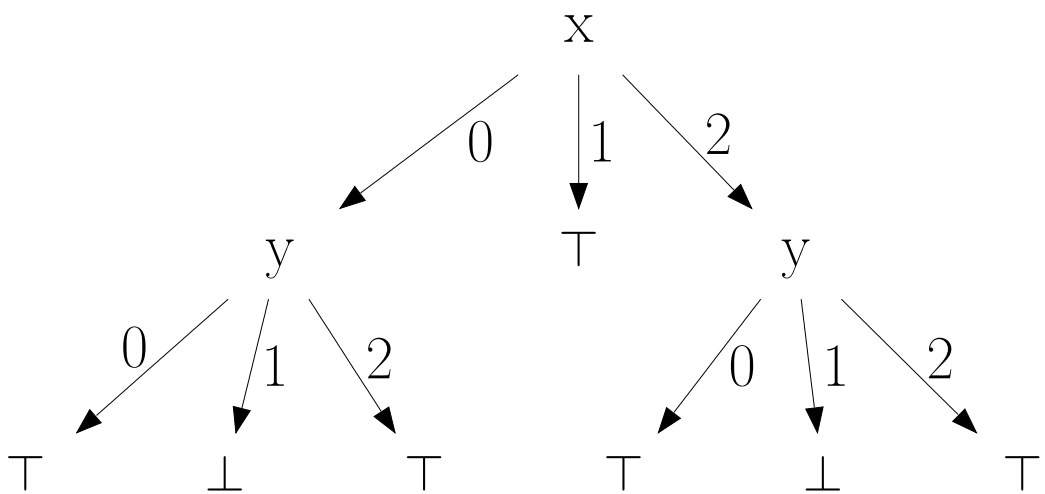


closing node y
$\rightarrow$ an equivalent node already exists in the SD:
it is not added

CHOCO with a Trace : Step by Step

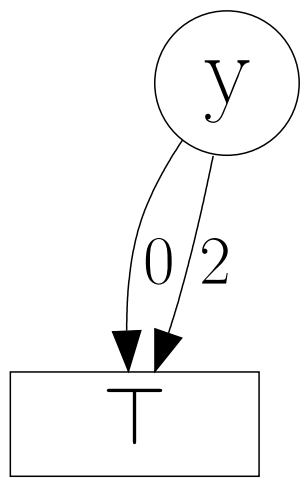
Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



closing node x

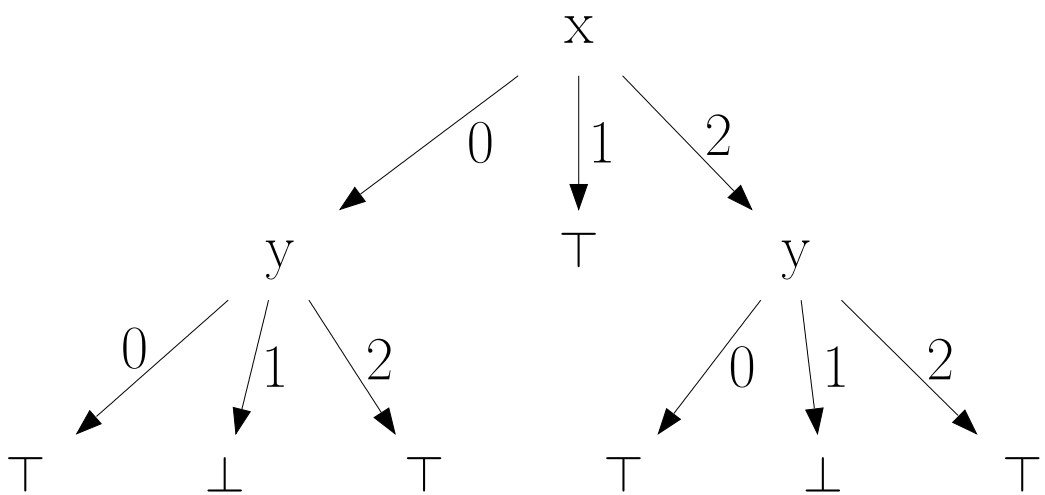
SD :



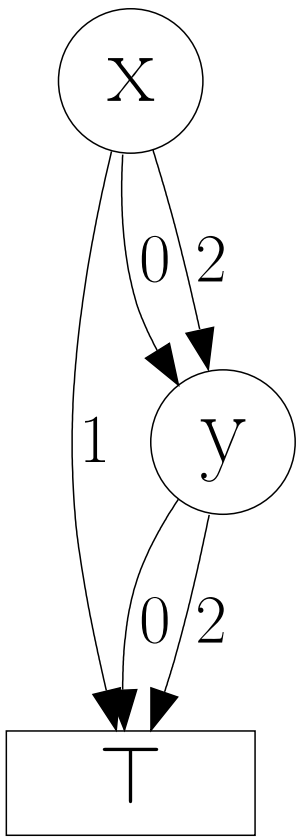
CHOCO with a Trace : Step by Step

Compilation of $x = y \vee y \neq 1$,
for x and y of domain $\{0, 1, 2\}$

Search tree:



SD :



closing node x
$\rightarrow$ it is added to the SD

Caching

- table of unique nodes: the compiler never builds two identical ("isomorphic") subgraphs
  $\rightarrow$ the final SD can be exponentially more compact than the search tree
- however the solver actually visited the whole search tree...

[Lec07] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. "Transposition Tables for Constraint Satisfaction". In: AAAI. 2007, pp. 243–248

Caching

- table of unique nodes: the compiler never builds two identical ("isomorphic") subgraphs
  $\rightarrow$ the final SD can be exponentially more compact than the search tree
- however the solver actually visited the whole search tree...

$\rightarrow$ caching: hash table of encountered subproblems

- hash key: domains of variables which are either
  - not yet assigned, or
  - used in a constraint not yet satisfied [Lec07]

[Lec07] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. "Transposition Tables for Constraint Satisfaction". In: AAAI. 2007, pp. 243–248

Resulting Compiled Form

- resulting SDs are always focusing and deterministic, but :
  - they are read-once if the solver branches on singletons
  - they are ordered if variables are considered in a static order

$\rightarrow$ "CHOCO with a Trace" compiles dFSDs, dRSDs or MDDs

Dynamic Order

- "CHOCO with a Trace" allows us to use variable choice heuristics
- how well are classical heuristics doing?
- which heuristics for compact compiled forms?

Heuristics Depending on the CSP

- ordering heuristics based on the contraint graph [Ami99]
- for a current partial order $\langle x_1, x_2, \ldots, x_k \rangle$
  - **HBW**: chooses a neighbor of x_1 first, then a neighbor of x_2, etc.
  - **HSBW**: chooses a neighbor of the first chosen variables
  - **MCSInv**: chooses the variable most linked to those already chosen

[Ami99] Jérôme Amilhastre. "Représentation par automate d'ensemble de solutions de problèmes de satisfaction de contraintes". PhD thesis. Université Montpellier II, 1999

Heuristics Depending on the Solver

- heuristics based on the cache: **MaxHashUse**
- goal: limiting the number of newly opened nodes
- checks the number of nodes that each variable choice opens
- chooses the one that opens the least

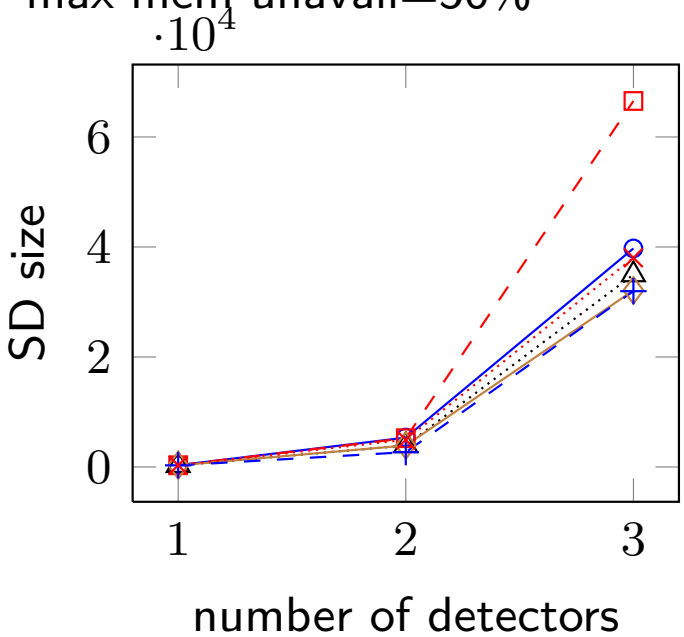
Benchmarks

- *Drone* : mission planning for a MAV in a competition (various goals to fulfill)
  $\rightarrow$ compilation of a transition table

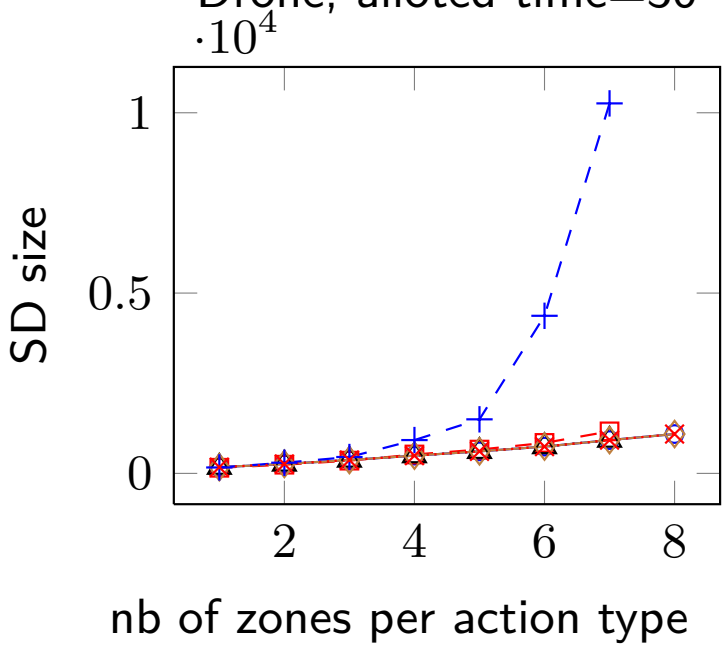
- *ObsToMem* : controller synthesis for the maintenance of links between detection instruments and mass memory of an observation satellite
  $\rightarrow$ compilation of a decision policy

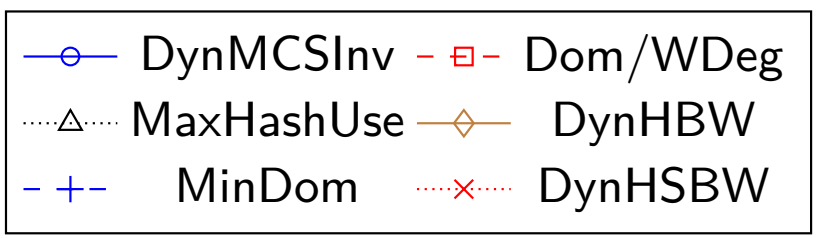
Comparison of Heuristics

ObsToMem, max com fail=50%,
max mem unavail=50%

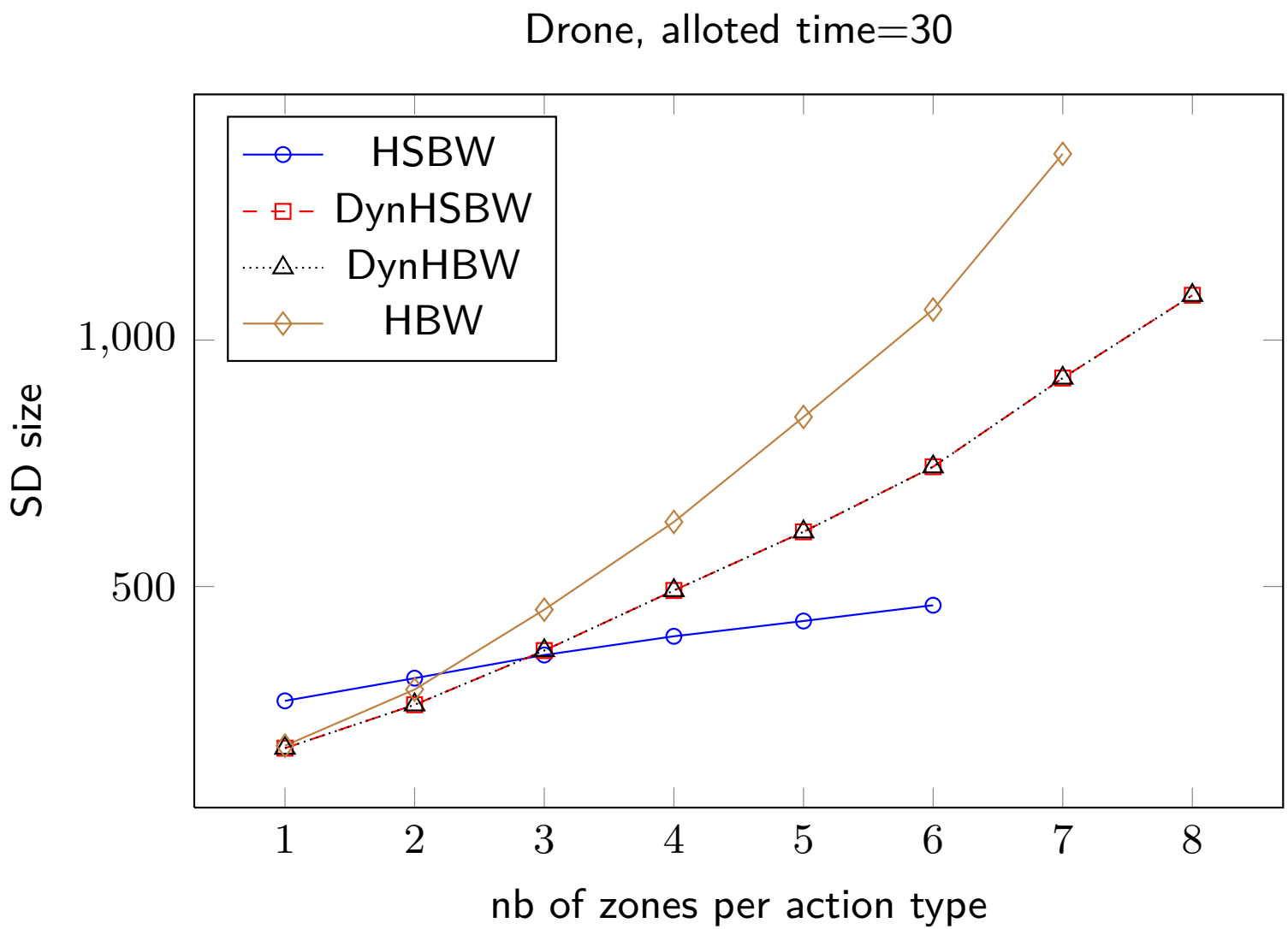


Drone, alloted time=30

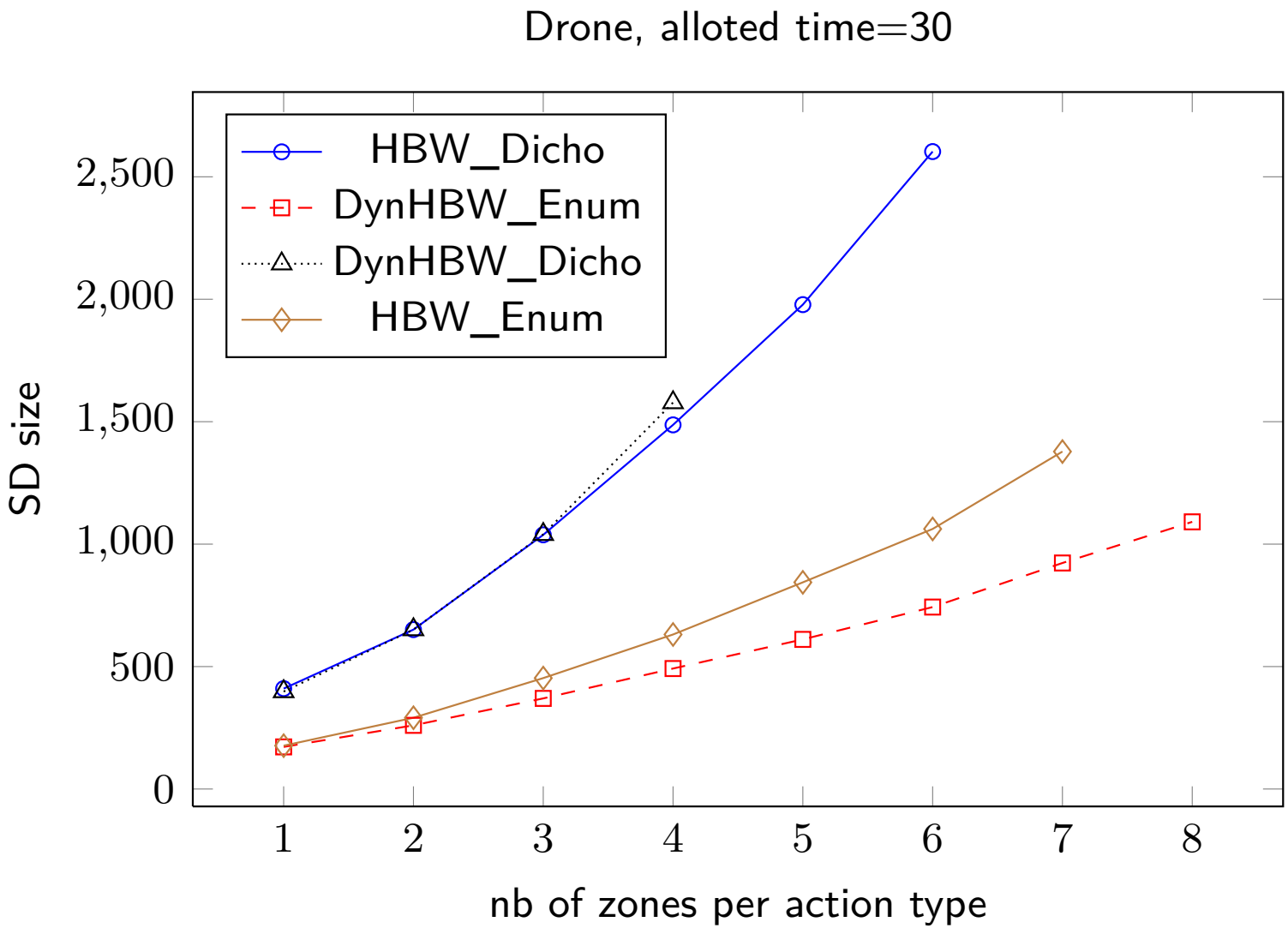




Dynamic vs Static



Branching Functions



Conclusion and Perspectives

- Contributions :
  - "CHOCO with a trace" algorithm: compiles CSPs into different variants of MDD
  - limits : compiler intrinsically deterministic
  - started studying compilation heuristics
- Perspectives :
  - more cache-based heuristics
  - how could the read-once requirement be efficiently relaxed?
  - adding "AND"-nodes to SDs

CHOCO with a trace

1: $\texttt{Prune}(\Pi)$
2: **if** Π is proven inconsistent **then**
3: **return** $\varnothing$
4: **if** $X_a = X$ **then** *// all variables are assigned*
5: **return** $\{\vec{x}\}$, with $\vec{x}$ the solution found
6: $x := \texttt{Choose\_var}(\Pi)$
7: $R := \texttt{Split}(\Pi, x)$
8: $S := \varnothing$
9: **for all** $r \in R$ **do**
10: soit $X'_a := X_a$
11: **if** r is reduced to a singleton **then** *// x is assigned*
12: $X'_a := X'_a \cup \{x\}$
13: $S_r := \texttt{CHOCO}(\Pi_{|x \in r}, X'_a)$
14: $S := S \cup S_r$
15: **return** S

CHOCO with a trace

1: $\texttt{Prune}(\Pi)$
2: **if** Π is proven inconsistent **then**
3: **return** the empty SD
4: **if** $X_a = X$ **then** *// all variables are assigned*
5: **return** $\{\vec{x}\}$, with $\vec{x}$ the solution found
6: $x := \texttt{Choose\_var}(\Pi)$
7: $R := \texttt{Split}(\Pi, x)$
8: $S := \varnothing$
9: **for all** $r \in R$ **do**
10: soit $X'_a := X_a$
11: **if** r is reduced to a singleton **then** *// x is assigned*
12: $X'_a := X'_a \cup \{x\}$
13: $S_r := \texttt{CHOCO}(\Pi_{|x \in r}, X'_a)$
14: $S := S \cup S_r$
15: **return** S

CHOCO with a trace

1: $\texttt{Prune}(\Pi)$
2: **if** Π is proven inconsistent **then**
3: **return** the empty SD
4: **if** $X_a = X$ **then** *// all variables are assigned*
5: **return** the sink-only SD
6: $x := \texttt{Choose\_var}(\Pi)$
7: $R := \texttt{Split}(\Pi, x)$
8: $S := \varnothing$
9: **for all** $r \in R$ **do**
10: soit $X'_a := X_a$
11: **if** r is reduced to a singleton **then** *// x is assigned*
12: $X'_a := X'_a \cup \{x\}$
13: $S_r := \texttt{CHOCO}(\Pi_{|x \in r}, X'_a)$
14: $S := S \cup S_r$
15: **return** S

CHOCO with a trace

1: $\texttt{Prune}(\Pi)$
2: **if** Π is proven inconsistent **then**
3: **return** the empty SD
4: **if** $X_a = X$ **then** *// all variables are assigned*
5: **return** the sink-only SD
6: $x := \texttt{Choose\_var}(\Pi)$
7: $R := \texttt{Split}(\Pi, x)$
8: $\Psi := \varnothing$
9: **for all** $r \in R$ **do**
10: soit $X'_a := X_a$
11: **if** r is reduced to a singleton **then** *// x is assigned*
12: $X'_a := X'_a \cup \{x\}$
13: $S_r := \texttt{CHOCO}(\Pi_{|x \in r}, X'_a)$
14: $S := S \cup S_r$
15: **return** S

CHOCO with a trace

1: $\texttt{Prune}(\Pi)$
2: **if** Π is proven inconsistent **then**
3: **return** the empty SD
4: **if** $X_a = X$ **then** *// all variables are assigned*
5: **return** the sink-only SD
6: $x := \texttt{Choose\_var}(\Pi)$
7: $R := \texttt{Split}(\Pi, x)$
8: $\Psi := \varnothing$
9: **for all** $r \in R$ **do**
10: soit $X'_a := X_a$
11: **if** r is reduced to a singleton **then** *// x is assigned*
12: $X'_a := X'_a \cup \{x\}$
13: $\psi_r := \texttt{CHOCO\_to\_SD}(\Pi_{|x \in r}, X'_a)$
14: $S := S \cup S_r$
15: **return** S

CHOCO with a trace

1: $\texttt{Prune}(\Pi)$
2: **if** Π is proven inconsistent **then**
3: **return** the empty SD
4: **if** $X_a = X$ **then** *// all variables are assigned*
5: **return** the sink-only SD
6: $x := \texttt{Choose\_var}(\Pi)$
7: $R := \texttt{Split}(\Pi, x)$
8: $\Psi := \varnothing$
9: **for all** $r \in R$ **do**
10: soit $X'_a := X_a$
11: **if** r is reduced to a singleton **then** *// x is assigned*
12: $X'_a := X'_a \cup \{x\}$
13: $\psi_r := \texttt{CHOCO\_to\_SD}(\Pi_{|x \in r}, X'_a)$
14: $\Psi := \Psi \cup \psi_r$
15: **return** S

CHOCO with a trace

1: $\texttt{Prune}(\Pi)$
2: **if** Π is proven inconsistent **then**
3: **return** the empty SD
4: **if** $X_a = X$ **then** *// all variables are assigned*
5: **return** the sink-only SD
6: $x := \texttt{Choose\_var}(\Pi)$
7: $R := \texttt{Split}(\Pi, x)$
8: $\Psi := \varnothing$
9: **for all** $r \in R$ **do**
10: soit $X'_a := X_a$
11: **if** r is reduced to a singleton **then** *// x is assigned*
12: $X'_a := X'_a \cup \{x\}$
13: $\psi_r := \texttt{CHOCO\_to\_SD}(\Pi_{|x \in r}, X'_a)$
14: $\Psi := \Psi \cup \psi_r$
15: **return** the SD rooted at $\texttt{Get\_node}(x, R, \Psi)$

CHOCO with a Trace + Caching

1: $\texttt{Prune}(\Pi)$

5: **if** Π is proven inconsistent **then**
6: **return** the empty SD
7: **if** $X_a = X$ **then**
8: **return** the sink-only SD
9: $x := \texttt{Split\_var}(\Pi)$
10: $R := \texttt{Split}(\Pi, x)$
11: $\Psi := \varnothing$
12: **for all** $r \in R$ **do**
13: let $X'_a := X_a$
14: **if** r is reduced to a singleton **then**
15: $X'_a := X'_a \cup \{x\}$
16: let $\psi_r := \texttt{SD\_builder}(\Pi_{|x \in r}, X'_a)$
17: $\psi_r := \texttt{CHOCO\_to\_SD}(\Pi_{|x \in r}, X'_a)$
18: let ψ be the SD rooted at $\texttt{Get\_node}(x, R, \Psi)$

20: **return** ψ

CHOCO with a Trace + Caching

1: $\texttt{Prune}(\Pi)$
2: $k := \texttt{Compute\_key}(\Pi, X_a)$
3: **if** there is an entry for the key k in the cache **then**
4: **return** the SD corresponding to key k in the cache
5: **if** Π is proven inconsistent **then**
6: **return** the empty SD
7: **if** $X_a = X$ **then**
8: **return** the sink-only SD
9: $x := \texttt{Split\_var}(\Pi)$
10: $R := \texttt{Split}(\Pi, x)$
11: $\Psi := \varnothing$
12: **for all** $r \in R$ **do**
13: let $X'_a := X_a$
14: **if** r is reduced to a singleton **then**
15: $X'_a := X'_a \cup \{x\}$
16: let $\psi_r := \texttt{SD\_builder}(\Pi_{|x \in r}, X'_a)$
17: $\psi_r := \texttt{CHOCO\_to\_SD}(\Pi_{|x \in r}, X'_a)$
18: let ψ be the SD rooted at $\texttt{Get\_node}(x, R, \Psi)$

20: **return** ψ

CHOCO with a Trace + Caching

1: $\texttt{Prune}(\Pi)$
2: $k := \texttt{Compute\_key}(\Pi, X_a)$
3: **if** there is an entry for the key k in the cache **then**
4: **return** the SD corresponding to key k in the cache
5: **if** Π is proven inconsistent **then**
6: **return** the empty SD
7: **if** $X_a = X$ **then**
8: **return** the sink-only SD
9: $x := \texttt{Split\_var}(\Pi)$
10: $R := \texttt{Split}(\Pi, x)$
11: $\Psi := \varnothing$
12: **for all** $r \in R$ **do**
13: let $X'_a := X_a$
14: **if** r is reduced to a singleton **then**
15: $X'_a := X'_a \cup \{x\}$
16: let $\psi_r := \texttt{SD\_builder}(\Pi_{|x \in r}, X'_a)$
17: $\psi_r := \texttt{CHOCO\_to\_SD}(\Pi_{|x \in r}, X'_a)$
18: let ψ be the SD rooted at $\texttt{Get\_node}(x, R, \Psi)$
19: store ψ at the key k in the cache
20: **return** ψ