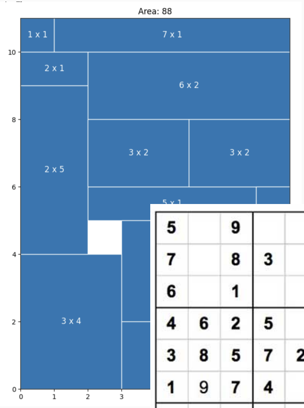
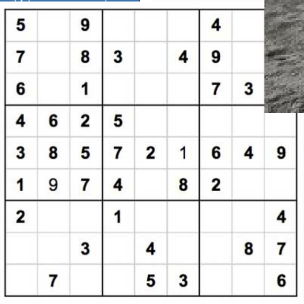
Tutorial: Optimization in CP

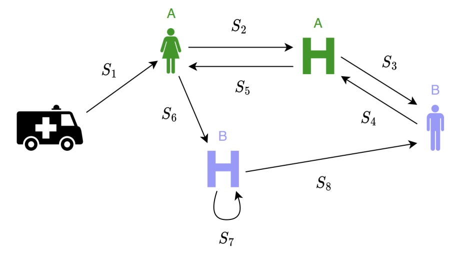
Dagstuhl Seminar: Interactions in Constraint Optimization

Hélène Verhaeghe - helene.verhaeghe@uclouvain.be

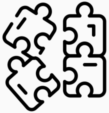
8 September 2025


Area: 88
1 x 1
7 x 1
2 x 1
6 x 2
2 x 5
3 x 2
3 x 2
5 x 1
3 x 4
10
8
6
4
2
0
0
1
2
3

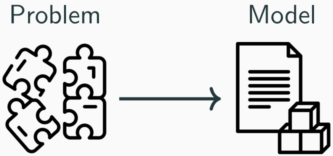

5 9 4
7 8 3 4 9
6 1 7 3
4 6 2 5
3 8 5 7 2 1 6 4 9
1 9 7 4 8 2
2 1 4
3 4 8 7
7 5 3 6

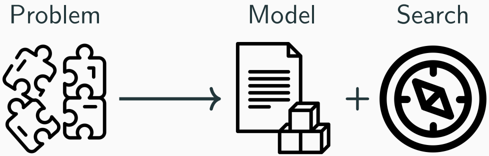

A
A
B
B
S_1
S_2
S_3
S_4
S_5
S_6
S_7
S_8

Problem

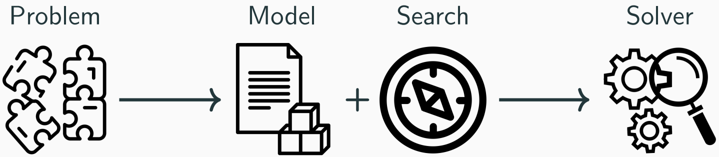


The Constraint Programming paradigm

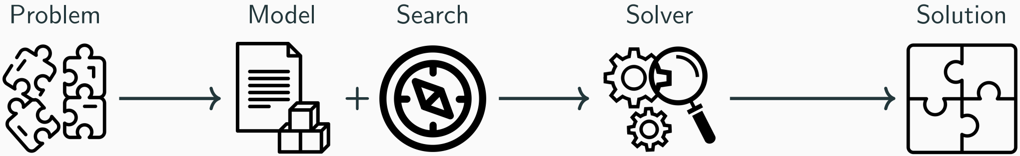



Problem
Model
+
Search

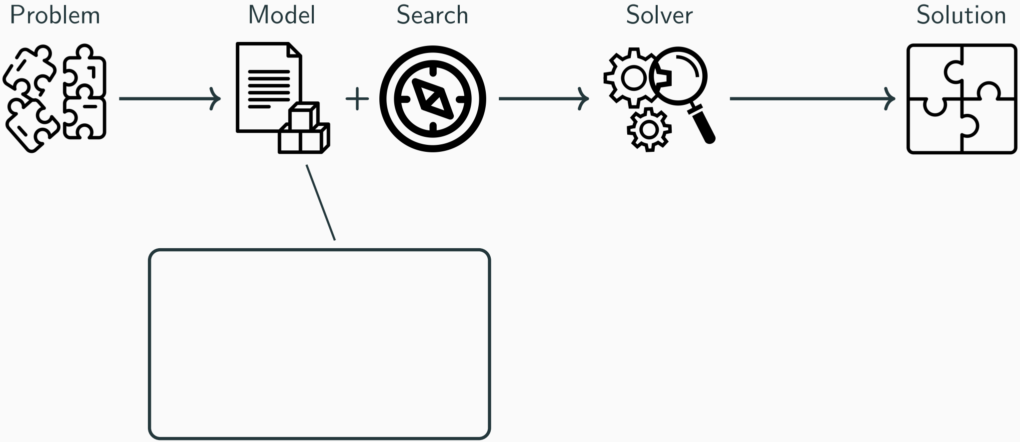
The Constraint Programming paradigm



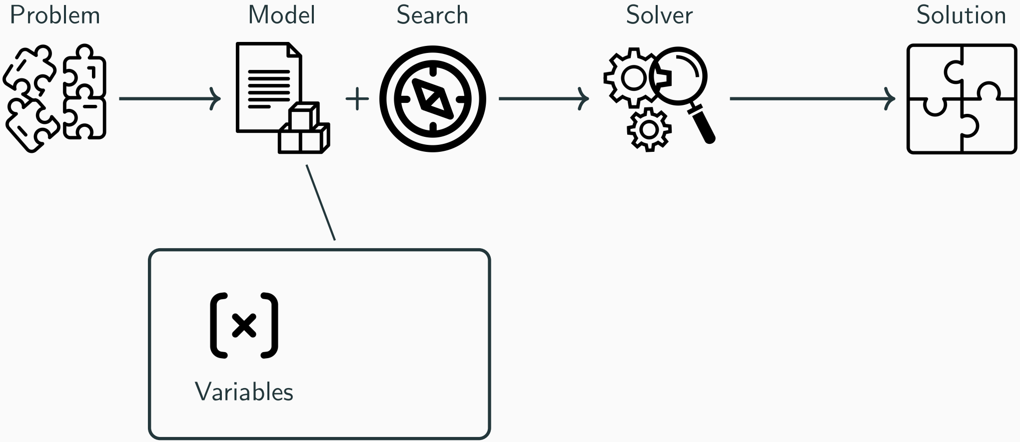
The Constraint Programming paradigm



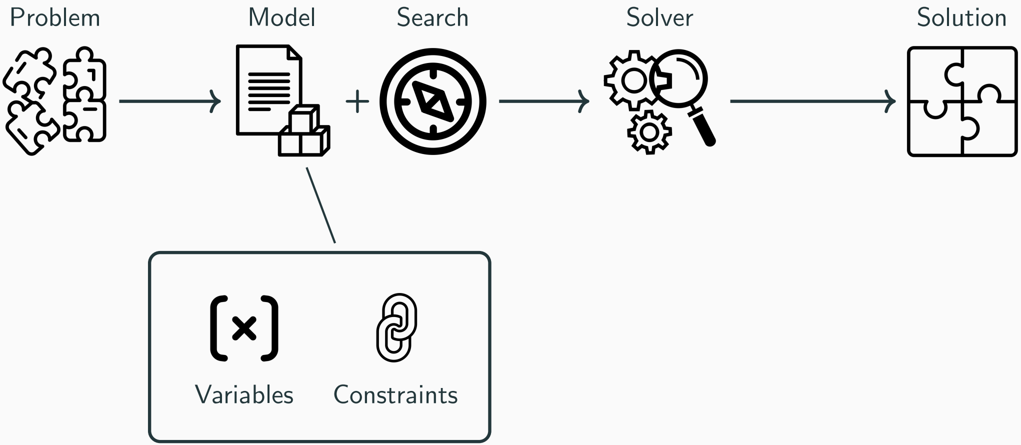
The Constraint Programming paradigm



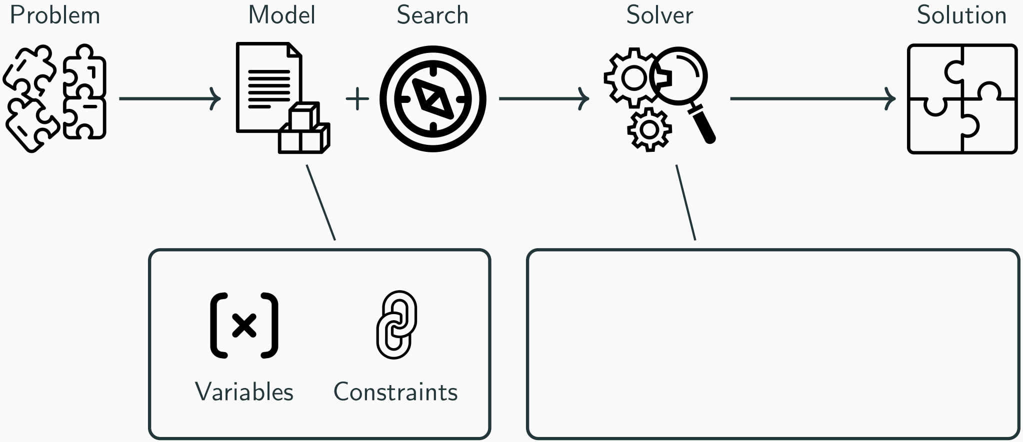
The Constraint Programming paradigm



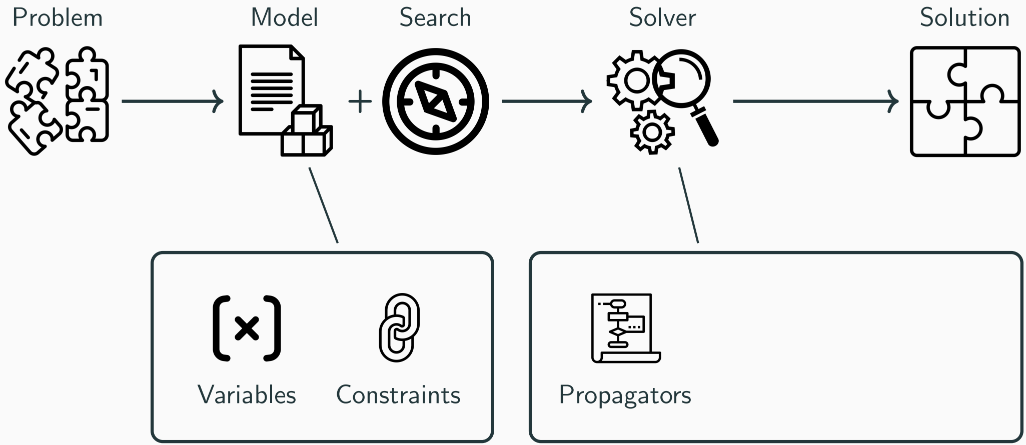
The Constraint Programming paradigm



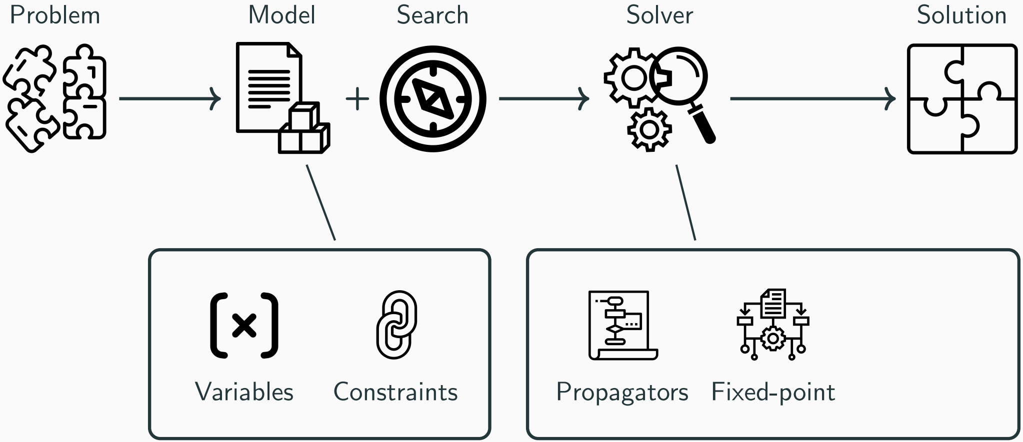
The Constraint Programming paradigm



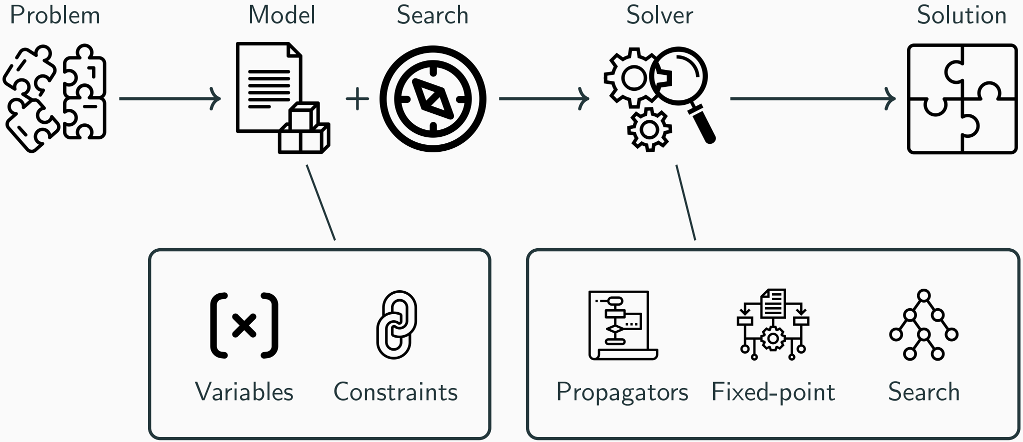
The Constraint Programming paradigm



The Constraint Programming paradigm



The Constraint Programming paradigm



Model

Variety of Variables types

X

D(X)

Goal of variables
Represent unknowns of the problem

[x]
Variable

X	D(X)
boolean	{false,true} (or {0,1})

Goal of variables
Represent unknowns of the problem

Variable

Variety of Variables types

X	D(X)
boolean	{false,true} (or {0,1})
integer	$\{\ldots,-2,-1,0,1,2,\ldots\} \subset \mathbb{Z}$

Goal of variables
Represent unknowns of the problem

Variable

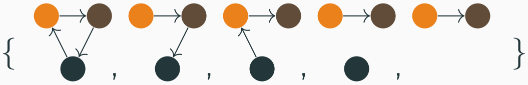
Variety of Variables types

X	D(X)
boolean	{false,true} (or {0,1})
integer	$\{\ldots,-2,-1,0,1,2,\ldots\} \subset \mathbb{Z}$
set	{{1,2,3},{1,2},{3,4},...}

Goal of variables
Represent unknowns of the problem

Variable

Variety of Variables types

X	D(X)
graph	

Goal of variables
Represent unknowns of the problem

Ref: **"CP(Graph): Introducing a Graph Computation Domain in Constraint Programming"**, by G. Dooms, Y. Deville, and P. Dupont, CP2005

Variable

X	D(X)
interval	{[1,2], [2,5], [3,5],...}

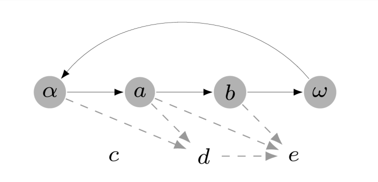
Goal of variables
Represent unknowns of the problem

Ref: **Interval constraint programming in C++**,
by E. Hyvönen, S. De Pascale, A. Lethola, Constraint Programming, 1994

Variable

X D(X)

sequence



Goal of variables
Represent unknowns of the problem

Ref: **Sequence Variables for Routing Problems**, by A. Delecluse, P. Schaus, P. Van Hentenryck, CP2022

Variable

Types of constraints

Constraint

- logical constraints: $X \vee Y$, $X \rightarrow Y$,...

- logical constraints: $X \lor Y$, $X \to Y$,...
- arithmetic constraints: $X + Y == Z$, $min(X, Y, Z) \geq 3$,...

- logical constraints: $X \vee Y$, $X \rightarrow Y$,...
- arithmetic constraints: $X + Y == Z$, $min(X, Y, Z) \geq 3$,...
- global constraints: *AllDifferent*, *Circuit*, *NValues*, *GCC*, *Cumulative*,...

Constraint

- logical constraints: $X \vee Y$, $X \rightarrow Y$,...
- arithmetic constraints: $X + Y == Z$, $min(X, Y, Z) \geq 3$,...
- global constraints: *AllDifferent*, *Circuit*, *NValues*, *GCC*, *Cumulative*,...

The goal of a global constraint is to **capture a relation** between a non-fixed number of variables

Example: Circuit constraint

Goal of global constraints

Capture a relation between a non-fixed number of variables

given $NY = 0,\ SF = 1, \ldots$

$$D(succ_i) = \{0, 1, \ldots, N-1\} \setminus \{i\}$$

$$\texttt{Circuit}([succ_{NY}, succ_{SF}, \ldots])$$

Constraint

Example: Circuit constraint

Goal of global constraints

Capture a relation between a non-fixed number of variables

given $NY = 0,\ SF = 1, \ldots$

$D(succ_i) = \{0, 1, \ldots, N-1\} \setminus \{i\}$

$\texttt{Circuit}([succ_{NY}, succ_{SF}, \ldots])$

$\min \sum \mathrm{dist}(i, succ_i)$

Constraint

minimise/maximise smth

Examples:

minimise X

maximise sum(X,Y,Z)

minimise NValue(X)

Modle

Search

- Heuristic guiding the construction of the search tree

The search, a guide for the search space exploration

- Heuristic guiding the construction of the search tree
- Goal: exploring the search space efficiently

The search, a guide for the search space exploration

- Heuristic guiding the construction of the search tree
- Goal: exploring the search space efficiently
- How: Choose a decision to perform, given the current state of the search
  - decision = a choice of variable and a value to assign it to

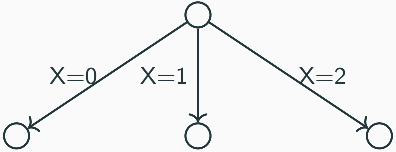
The search, a guide for the search space exploration

- Heuristic guiding the construction of the search tree
- Goal: exploring the search space efficiently
- How: Choose a decision to perform, given the current state of the search
  - decision = a choice of variable and a value to assign it to
- First-Fail principle: if a decision leads to a fail, it is better to have it early in the search tree, to allow the pruning of many decisions
  - Ref: "**Increasing tree search efficiency for constraint satisfaction problems**", by R. Haralick and G. Elliott, IJCAI 1979

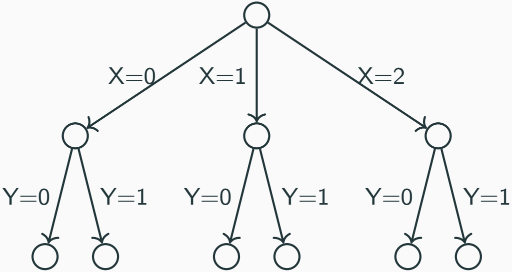
Given $D(X) = \{0, 1, 2\}$ and $D(Y) = \{0, 1\}$

Min-Dom Heuristic

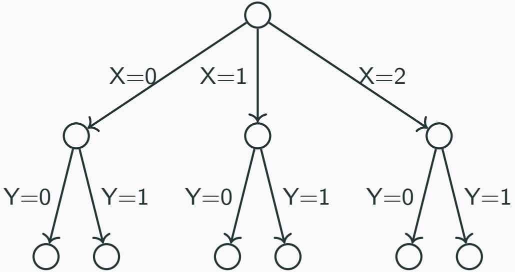
Given $D(X) = \{0, 1, 2\}$ and $D(Y) = \{0, 1\}$

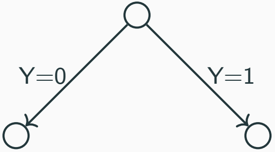


Given $D(X) = \{0, 1, 2\}$ and $D(Y) = \{0, 1\}$



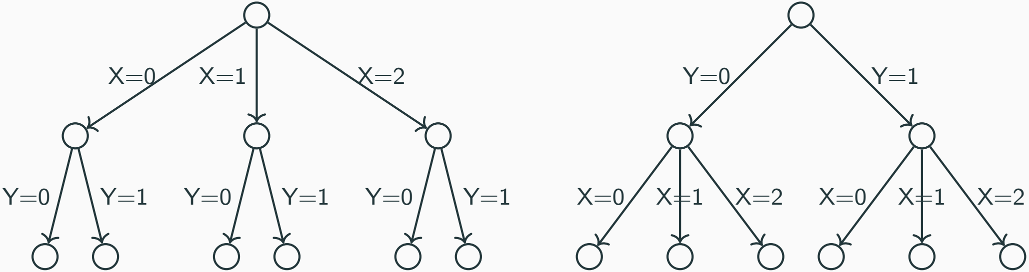
Given $D(X) = \{0, 1, 2\}$ and $D(Y) = \{0, 1\}$





Min-Dom Heuristic

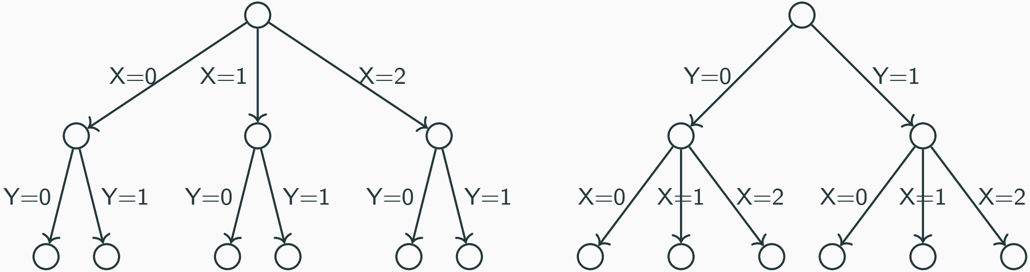
Given $D(X) = \{0, 1, 2\}$ and $D(Y) = \{0, 1\}$



Search

Min-Dom Heuristic

Given $D(X) = \{0, 1, 2\}$ and $D(Y) = \{0, 1\}$



Selecting smallest domain first leads to fewer nodes

Search

$$\min \frac{\mid D(X) \mid}{Deg(X)}$$

- $Deg(X)$ is the number of constraints where X is involved
- Idea: target small domains first (smaller tree), but also variables that might be in the center of conflicts (fail early)

Activity-Based Search (ABS) Heuristic

$$\max \frac{A(X)}{\mid D(X) \mid}$$

- $A(X)$ is the activity indicator
- At each decision:
  - $A(X) = A(X) \times \alpha$ for each unbound X (decay)
  - $A(X) = A(X) + 1$ if $D(X)$ have been modified by the decision
- Idea: Variables whose domains shrink easily are at the center of conflicts (fail early)
- Usually initialised by a bit of random search first

Search

Conflict Ordering Search (COS) Heuristic

UCLouvain

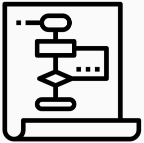
$$\max \textit{timestamp}(X)$$

- $\textit{timestamp}(X)$ is the last time a decision on X led to a dead end
- At each decision, if there is a failure (dead end):
  - $\textit{timestamp}(X) = \textit{time}$, where X is the variable at the decision
- Idea: fix variables at the heart of conflict first (fail early)

Search

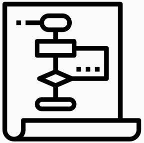
Solver

Propagators



Propagator

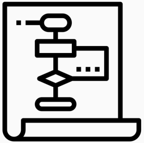
Goal: **Filter invalid values**

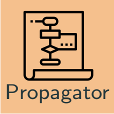


Goal: **Filter invalid values**

Input

domains

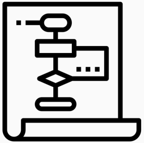




Goal: **Filter invalid values**

Input

domains →



→ updated domains

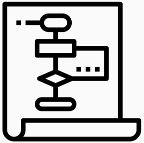
Output

Propagator

Goal: **Filter invalid values**

Input

Output

domains ⟶  ⟶ updated domains

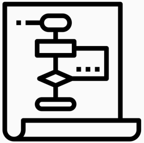
$D(X){=}D(Y){=}\{0{,}1\},$
$D(Z){=}\{0{,}1{,}2\}$

Propagator

Goal: **Filter invalid values**

Input

domains →



Output

→ updated domains

$D(X){=}D(Y){=}\{0{,}1\},$
$D(Z){=}\{0{,}1{,}2\}$

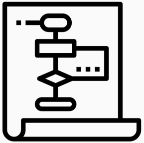
$AllDifferent(X,Y,Z)$

Propagator

Goal: **Filter invalid values**

Input

domains →



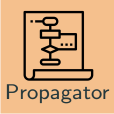
→ updated domains

Output

$D(X)=D(Y)=\{0,1\}$,
$D(Z)=\{0,1,2\}$

$AllDifferent(X,Y,Z)$

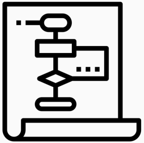
$D(X)=D(Y)=\{0,1\}$,
$D(Z)=\{2\}$



Goal: **Filter invalid values**

Input

domains →



→ updated domains

Output

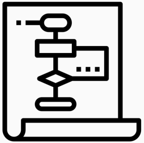
$$D(X)=D(Z)=\{0,1\},$$
$$D(Y)=\{0,1,2\}$$

Propagator

Goal: **Filter invalid values**

Input

domains $\longrightarrow$



$\longrightarrow$ updated domains

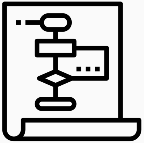
Output

$D(X)=D(Z)=\{0,1\}$,
$D(Y)=\{0,1,2\}$

$X+Y=Z$

Propagator

Goal: **Filter invalid values**

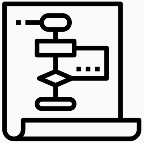
Input		Output
domains →		→ updated domains
$D(X){=}D(Z){=}\{0{,}1\}$, $D(Y){=}\{0{,}1{,}2\}$	$X{+}Y{=}Z$	$D(X){=}D(Z){=}\{0{,}1\}$, $D(Y){=}\{0{,}1\}$


Propagator

Goal: **Filter invalid values**

Input

domains $\longrightarrow$



$\longrightarrow$ updated domains

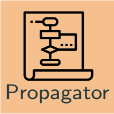
Output

$D(X)=D(Z)=\{0,1\}$,
$D(Y)=\{0,1,2\}$

$X+Y=Z$

$D(X)=D(Z)=\{0,1\}$,
$D(Y)=\{0,1\}$

The propagator algorithm is tailored to the semantics of the constraint



Strength of propagator

Given $D(X) = \{0,1,3,5\}$, $D(Y) = \{0,2,4\}$ and $D(Z) = \{3,5,6,8\}$, and $X + Y == Z$

Propagator

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

Propagator

Strength of propagator

UCLouvain

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$$D(X) = [0, 5]$$
$$D(Y) = [0, 4]$$
$$D(Z) = [3, 8]$$

Propagator

Strength of propagator

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0, 5]$
$D(Y) = [0, 4]$
$D(Z) = [3, 8]$

X	Y	Z
0	3	3
5	0	5

Strength of propagator

Given $D(X) = \{0,1,3,5\}$, $D(Y) = \{0,2,4\}$ and $D(Z) = \{3,5,6,8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0,5]$
$D(Y) = [0,4]$
$D(Z) = [3,8]$

X	Y	Z
0	3	3
5	0	5
4	**0**	4
3	**4**	7

Propagator

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0, 5]$
$D(Y) = [0, 4]$
$D(Z) = [3, 8]$

X	Y	Z
0	3	3
5	0	5
4	**0**	4
3	**4**	7
2	1	**3**
4	4	**8**

Propagator

Strength of propagator

UCLouvain

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0, 5]$
$D(Y) = [0, 4]$
$D(Z) = [3, 8]$

Support for every bound, no filtering

X	Y	Z
0	3	3
5	0	5
4	**0**	4
3	**4**	7
2	1	**3**
4	4	**8**

Propagator

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0, 5]$
$D(Y) = [0, 4]$
$D(Z) = [3, 8]$

Support for every bound, no filtering

X	Y	Z
0	3	3
5	0	5
4	**0**	4
3	**4**	7
2	1	**3**
4	4	**8**

Global Arc Consistant (GAC)

Propagator

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0, 5]$
$D(Y) = [0, 4]$
$D(Z) = [3, 8]$

Support for every bound, no filtering

X	Y	Z
0	3	3
5	0	5
4	**0**	4
3	**4**	7
2	1	**3**
4	4	**8**

Global Arc Consistant (GAC)

X	Y	Z
1	2	3
3	0	3
5	0	5

Propagator

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0, 5]$
$D(Y) = [0, 4]$
$D(Z) = [3, 8]$

Support for every bound, no filtering

X	Y	Z
0	3	3
5	0	5
4	**0**	4
3	**4**	7
2	1	**3**
4	4	**8**

Global Arc Consistant (GAC)

X	Y	Z
1	2	3
3	0	3
5	0	5
3	**0**	3

X	Y	Z
1	**2**	3
1	**4**	5

Propagator

Strength of propagator

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0, 5]$
$D(Y) = [0, 4]$
$D(Z) = [3, 8]$

Support for every bound, no filtering

X	Y	Z
0	3	3
5	0	5
4	**0**	4
3	**4**	7
2	1	**3**
4	4	**8**

Global Arc Consistant (GAC)

X	Y	Z
1	2	3
3	0	3
5	0	5
3	**0**	3

X	Y	Z
1	**2**	3
1	**4**	5
3	0	**3**
1	4	**5**

Propagator

Strength of propagator

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0, 5]$
$D(Y) = [0, 4]$
$D(Z) = [3, 8]$

Support for every bound, no filtering

X	Y	Z
0	3	3
5	0	5
4	**0**	4
3	**4**	7
2	1	**3**
4	4	**8**

Global Arc Consistant (GAC)

X	Y	Z
1	2	3
3	0	3
5	0	5
3	**0**	3

X	Y	Z
1	**2**	3
1	**4**	5
3	0	**3**
1	4	**5**

No support for X=0, Z=6, Z=8!

Propagator

Strength of propagator

UCLouvain

Given $D(X) = \{0, 1, 3, 5\}$, $D(Y) = \{0, 2, 4\}$ and $D(Z) = \{3, 5, 6, 8\}$, and $X + Y == Z$

Bound Consistant (BC(Z))

$D(X) = [0, 5]$
$D(Y) = [0, 4]$
$D(Z) = [3, 8]$

Support for every bound, no filtering

X	Y	Z
0	3	3
5	0	5
4	**0**	4
3	**4**	7
2	1	**3**
4	4	**8**

Global Arc Consistant (GAC)

X	Y	Z
1	2	3
3	0	3
5	0	5
3	**0**	3

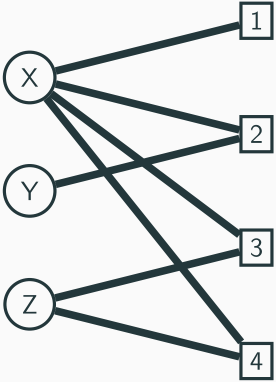
X	Y	Z
1	**2**	3
1	**4**	5
3	0	**3**
1	4	**5**

No support for X=0, Z=6, Z=8!

Side note: There exists another bound consistancy (BC(D)), slightly stronger than BC(Z)

Propagator

Example of propagator: AllDifferent



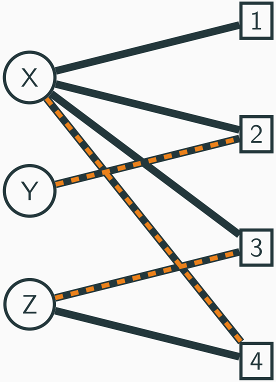
Goal of propagators
Filter invalid values

Propagator

Reference: **"A filtering algorithm for constraints of difference in CSPs"**, by J.-C. Régin, AAAI94

Example of propagator: AllDifferent

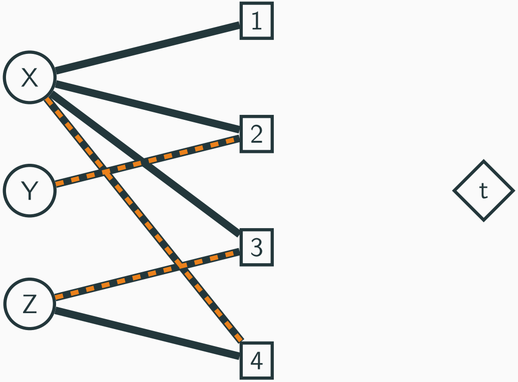
Goal of propagators
Filter invalid values



Reference: **"A filtering algorithm for constraints of difference in CSPs"**, by J.-C. Régin, AAAI94

Example of propagator: AllDifferent

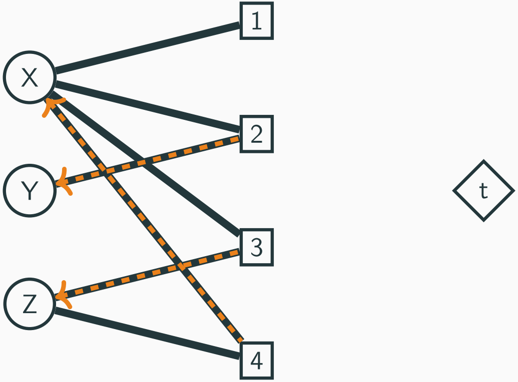
Goal of propagators
Filter invalid values



Propagator

Reference: **"A filtering algorithm for constraints of difference in CSPs"**, by J.-C. Régin, AAAI94

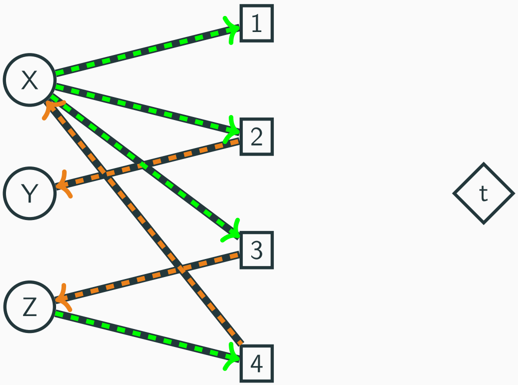
Example of propagator: AllDifferent



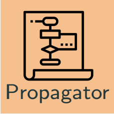
Goal of propagators
Filter invalid values

Reference: **"A filtering algorithm for constraints of difference in CSPs"**, by J.-C. Régin, AAAI94

Example of propagator: AllDifferent



Goal of propagators
Filter invalid values

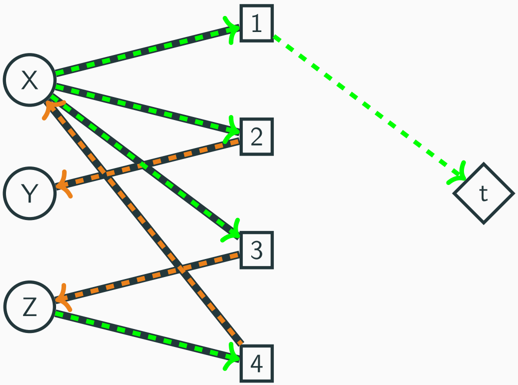


Reference: **"A filtering algorithm for constraints of difference in CSPs"**, by J.-C. Régin, AAAI94

Example of propagator: AllDifferent

UCLouvain

Goal of propagators
Filter invalid values

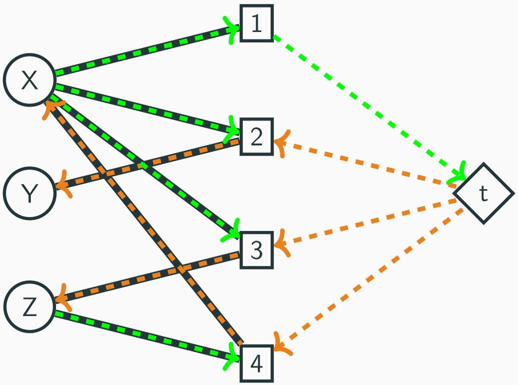


Propagator

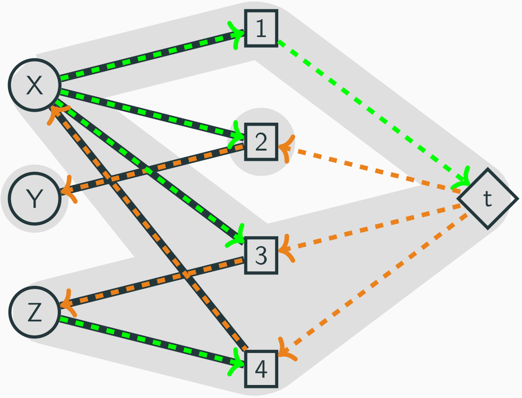
Reference: **"A filtering algorithm for constraints of difference in CSPs"**, by J.-C. Régin, AAAI94

Example of propagator: AllDifferent

Goal of propagators
Filter invalid values



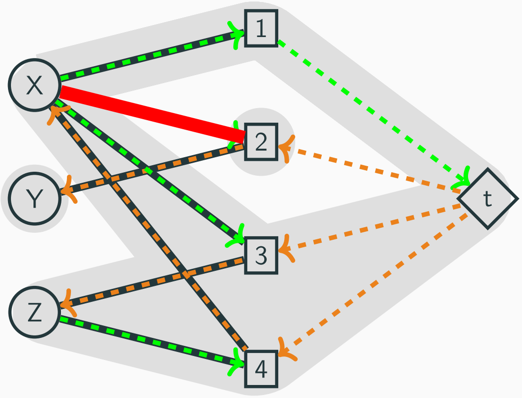
Reference: **"A filtering algorithm for constraints of difference in CSPs"**, by J.-C. Régin, AAAI94



Goal of propagators
Filter invalid values

Reference: **"A filtering algorithm for constraints of difference in CSPs"**, by J.-C. Régin, AAAI94

Example of propagator: AllDifferent

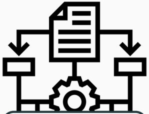


Goal of propagators
Filter invalid values

Reference: **"A filtering algorithm for constraints of difference in CSPs"**, by J.-C. Régin, AAAI94

The fixed-point algorithm: scheduling the calls to constraint

Goal of fixed-point
Schedule constraints until no more values can be removed

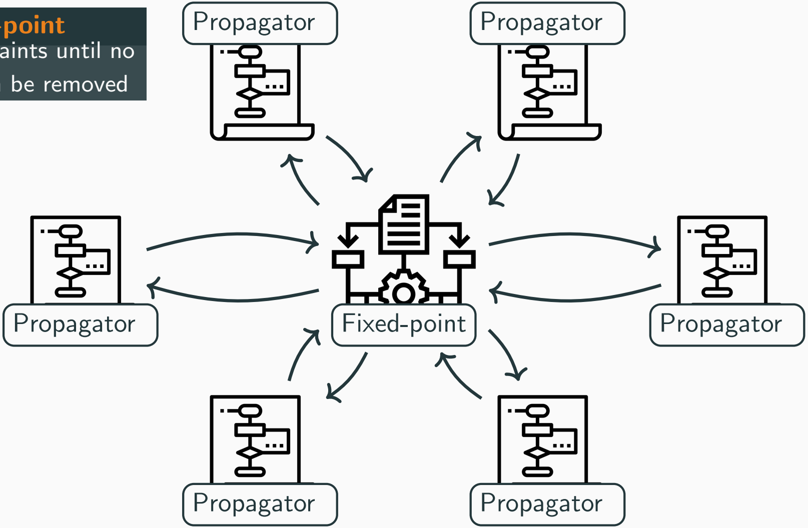


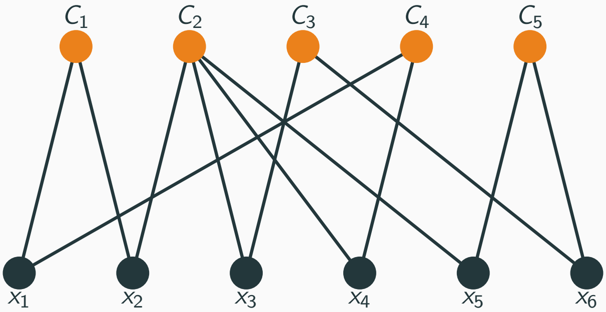
Fixed-point

Fixed-point

Goal of fixed-point

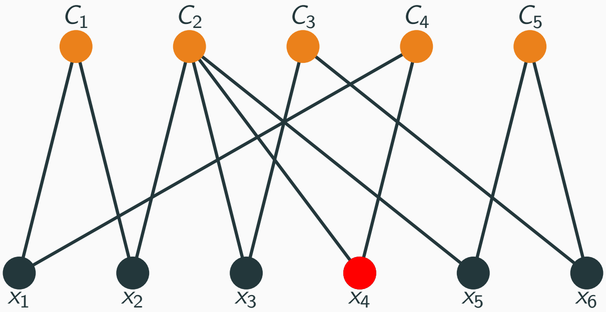
Schedule constraints until no more values can be removed





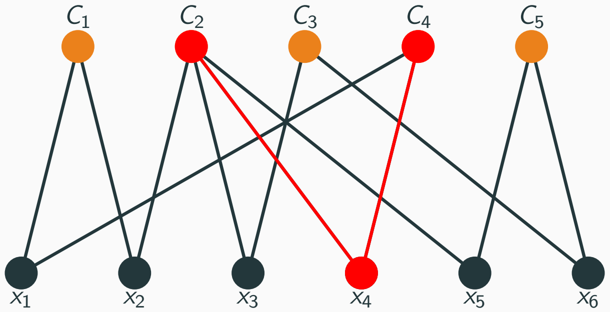
Propagating: $\square$ Queue:

Fixed-point

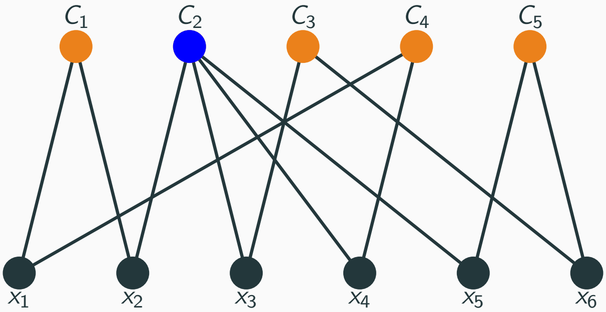


Propagating: $\boxed{}$ Queue:

The fixed-point algorithm: scheduling the calls to constraint

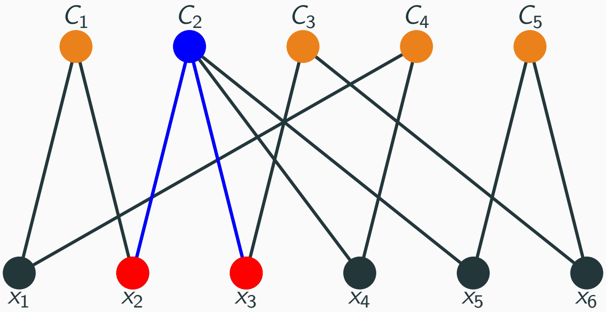


Propagating: $\boxed{}$ Queue: C_2, C_4



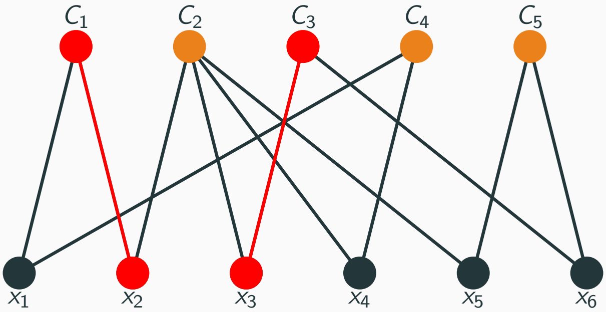
Propagating: $\boxed{C_2}$ Queue: C_4

Fixed-point



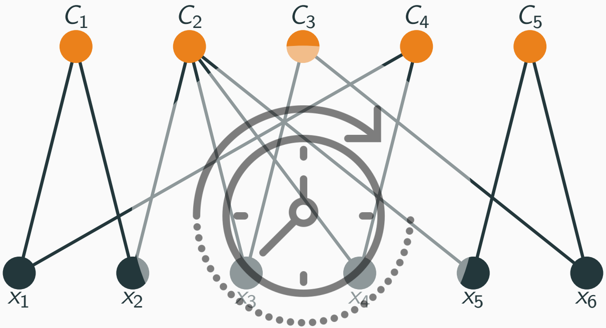
Propagating: $\boxed{C_2}$ Queue: C_4

The fixed-point algorithm: scheduling the calls to constraint

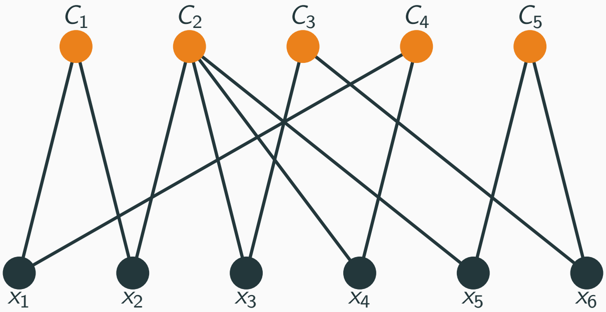


Propagating: $\boxed{}$ Queue: C_4, C_1, C_3

Fixed-point



Propagating: $\square$ Queue: C_4, C_1, C_3

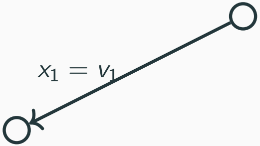


Propagating: $\boxed{}$ Queue:

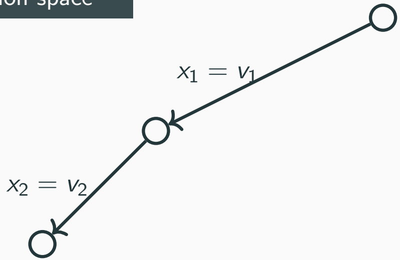
The search tree exploration

Goal of the search
Explore the solution space

Goal of the search
Explore the solution space

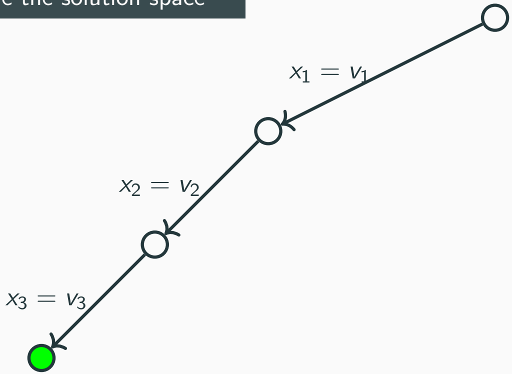


Goal of the search
Explore the solution space



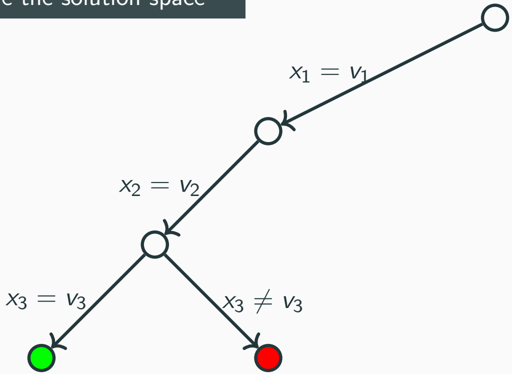
The search tree exploration

Goal of the search
Explore the solution space



Search

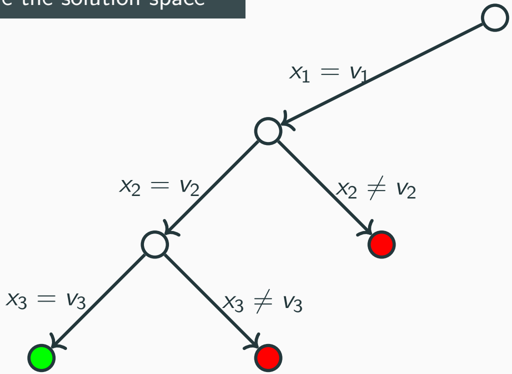
Goal of the search
Explore the solution space



Search

Goal of the search

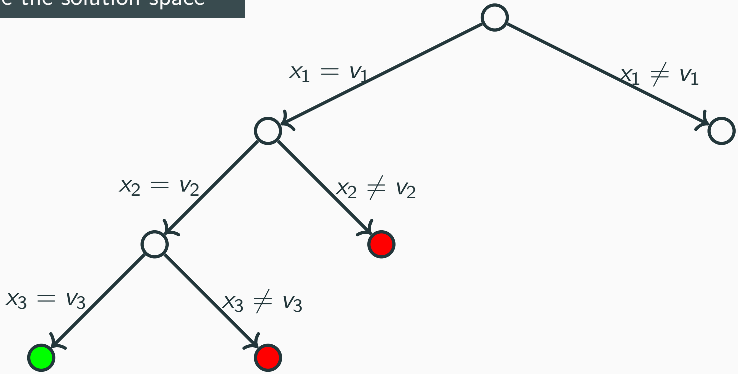
Explore the solution space



Search

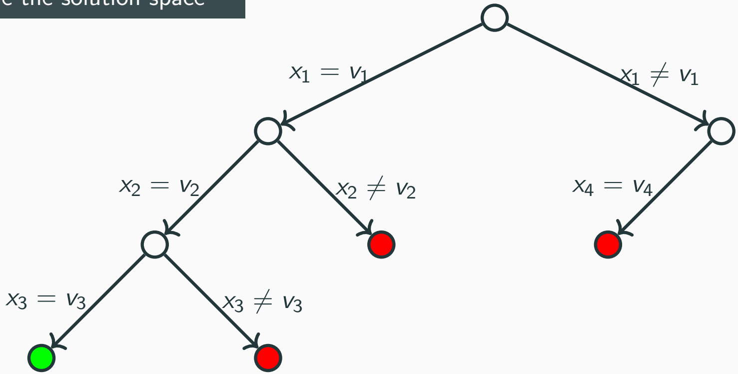
UCLouvain

Goal of the search
Explore the solution space



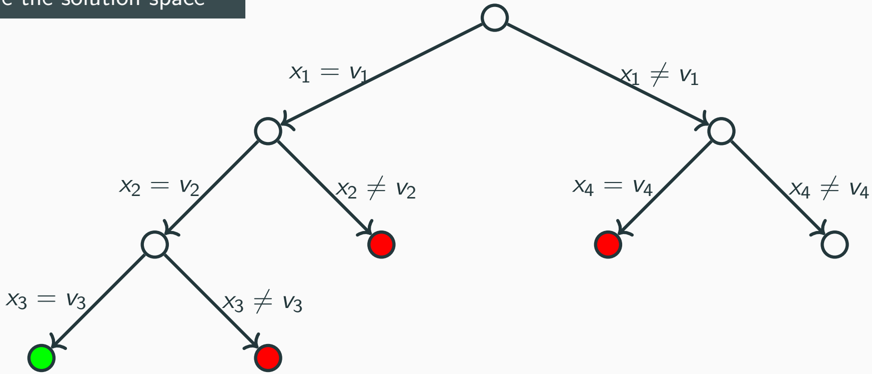
Search

Goal of the search
Explore the solution space

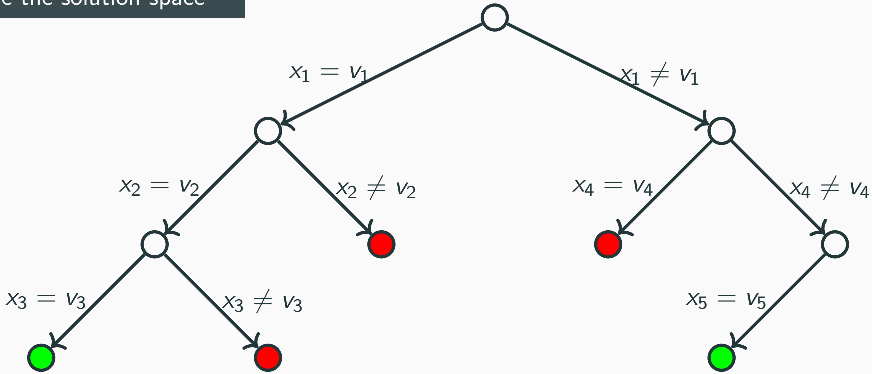


UCLouvain

Goal of the search
Explore the solution space



Goal of the search
Explore the solution space

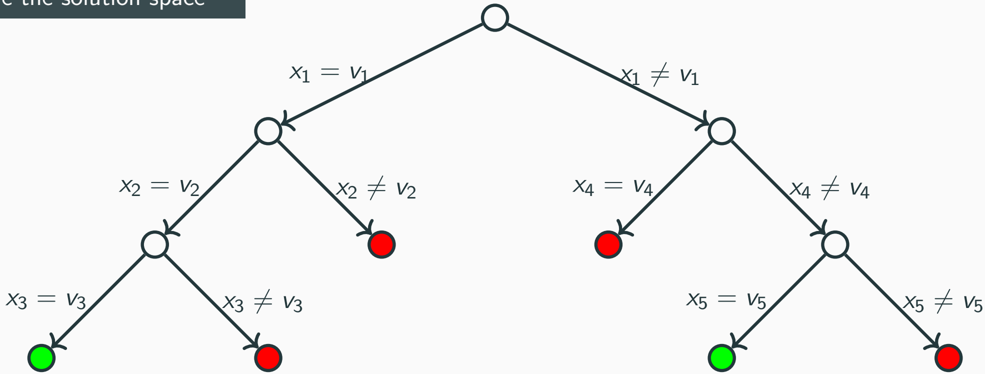


The search tree exploration

UCLouvain

Goal of the search

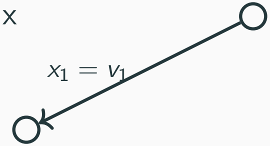
Explore the solution space



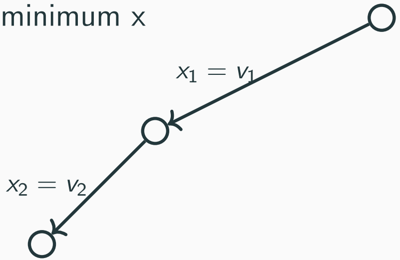
Search

obj: minimum x

obj: minimum x

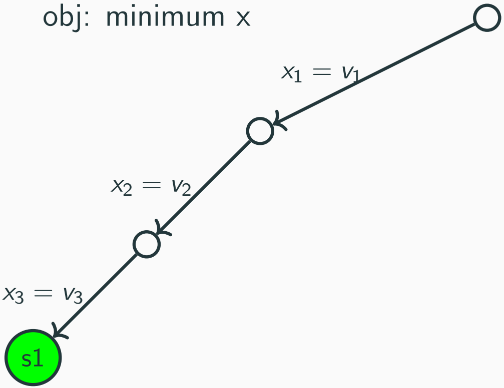


obj: minimum x

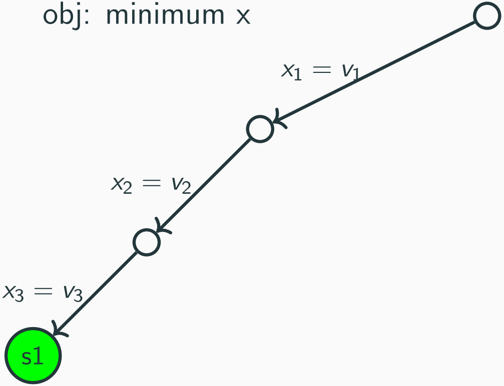


Search tree for an optimal value

obj: minimum x



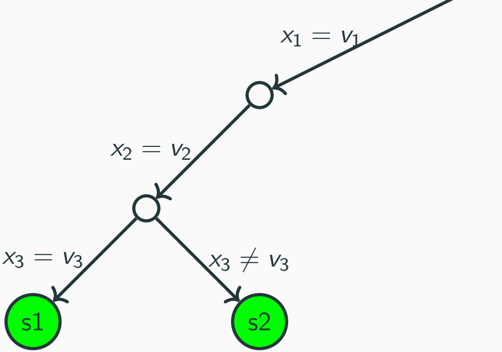
obj: minimum x



add cst: $x < obj(s1)$

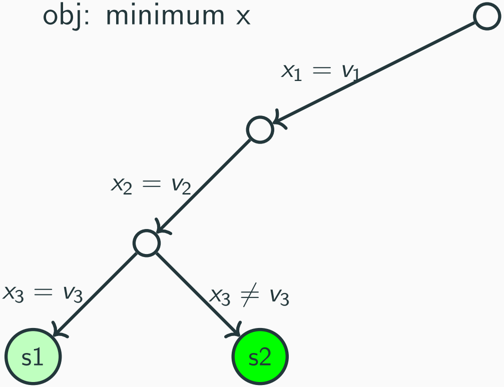
Search tree for an optimal value

obj: minimum x



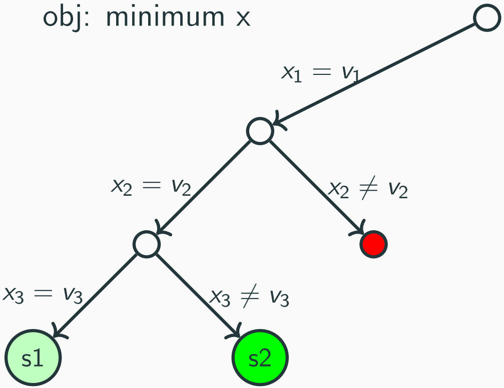
add cst: $x < obj(s1)$

obj: minimum x



add cst: $x < obj(s2)$

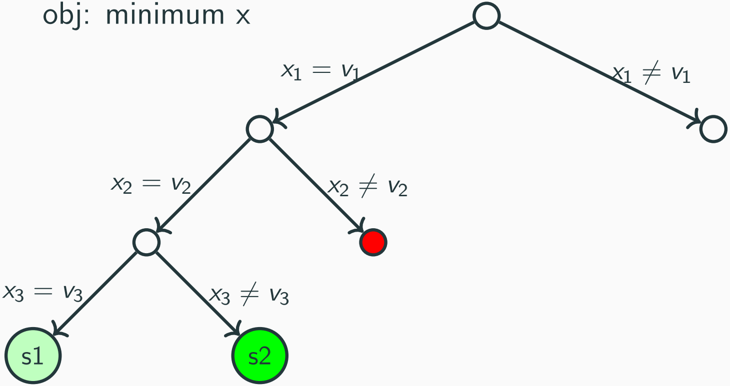
obj: minimum x



add cst: $x < obj(s2)$

Search tree for an optimal value

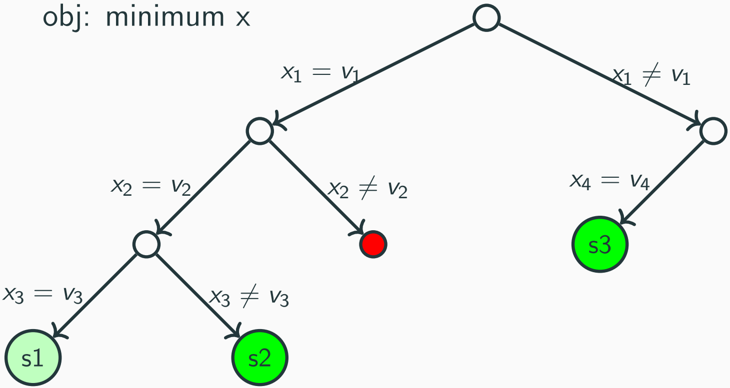
obj: minimum x



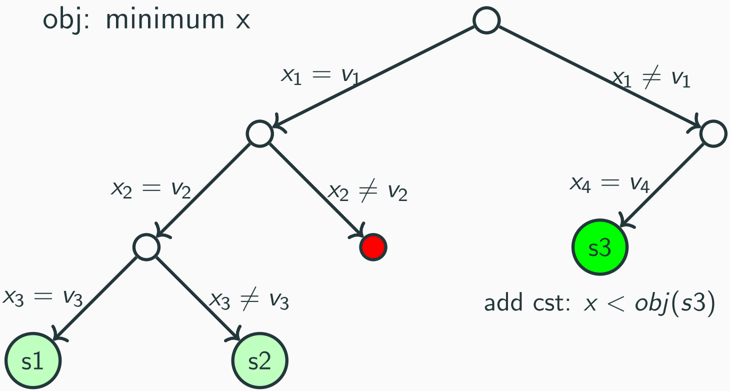
add cst: $x < obj(s2)$

Search

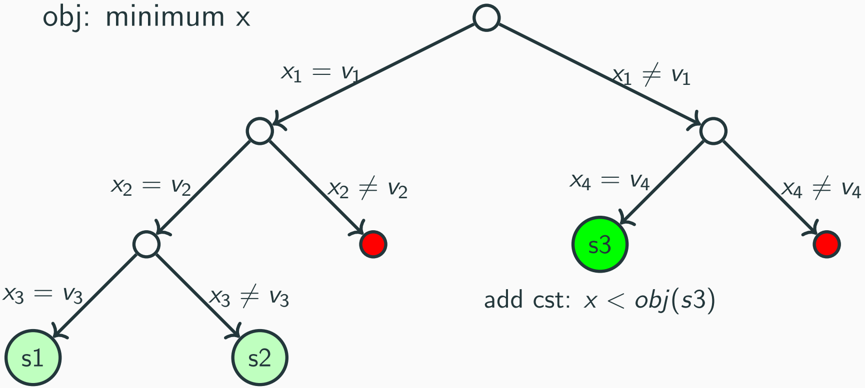
obj: minimum x



add cst: $x < obj(s2)$

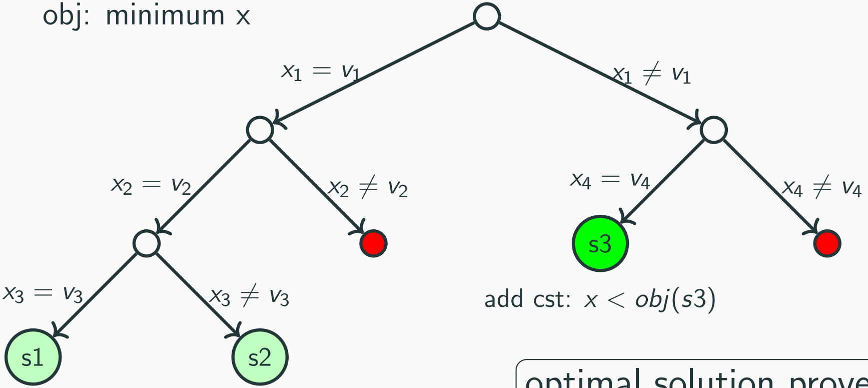

obj: minimum x
$x_1 = v_1$
$x_1 \neq v_1$
$x_2 = v_2$
$x_2 \neq v_2$
$x_4 = v_4$
s3
add cst: $x < obj(s3)$
$x_3 = v_3$
$x_3 \neq v_3$
s1
s2

Search


obj: minimum x
$x_1 = v_1$
$x_1 \neq v_1$
$x_2 = v_2$
$x_2 \neq v_2$
$x_4 = v_4$
$x_4 \neq v_4$
$x_3 = v_3$
$x_3 \neq v_3$
s1
s2
s3
add cst: $x < obj(s3)$

Search tree for an optimal value

obj: minimum x



add cst: $x < obj(s3)$

optimal solution proven!

Backtracking structures

How to go back in the search?

Before decision, set "state save points" you can go back to

Copying

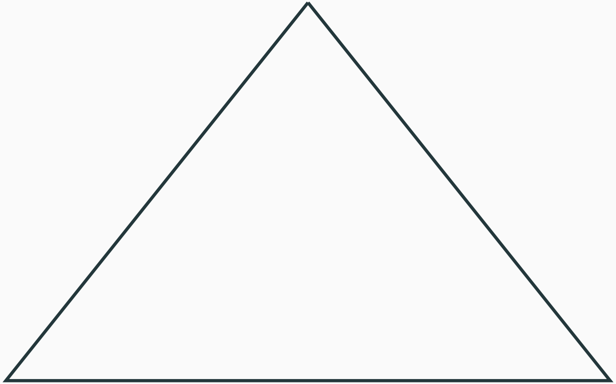
- save: copy the required part of the state
- restore: replace the state with the copy

Trailing

- save: create restoring operation for each modification since last save
- restore: revert the modifications one by one

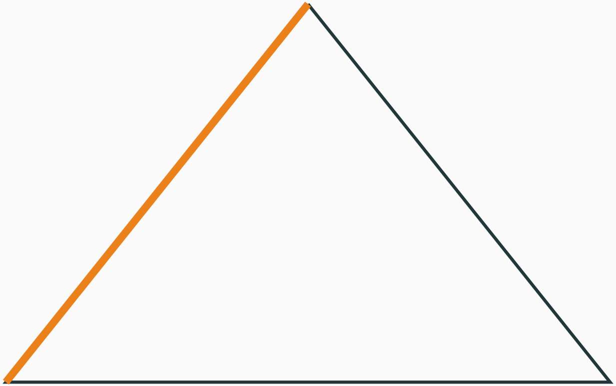
Propagators can benefit by having incremental algorithms

Huge search space



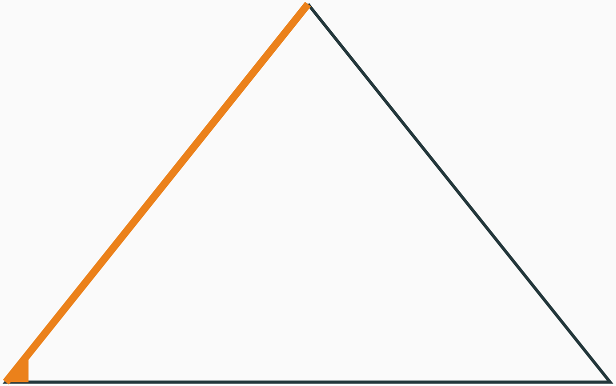
Search

Huge search space



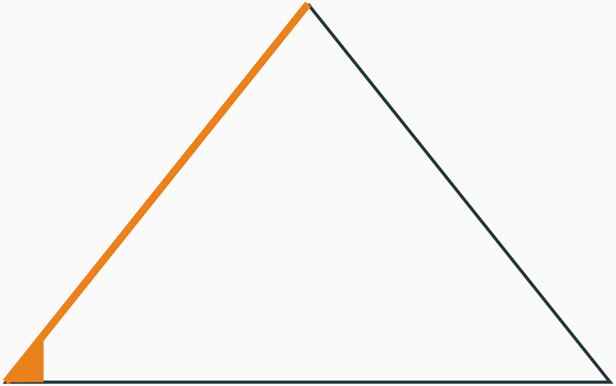
Search

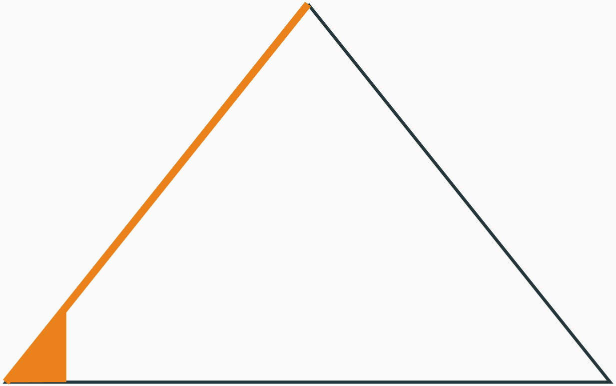
Huge search space



Search

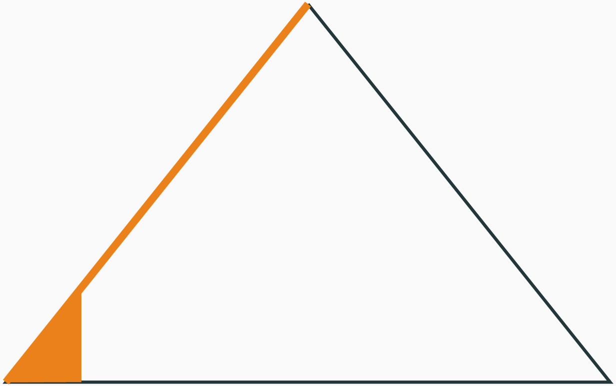
Huge search space



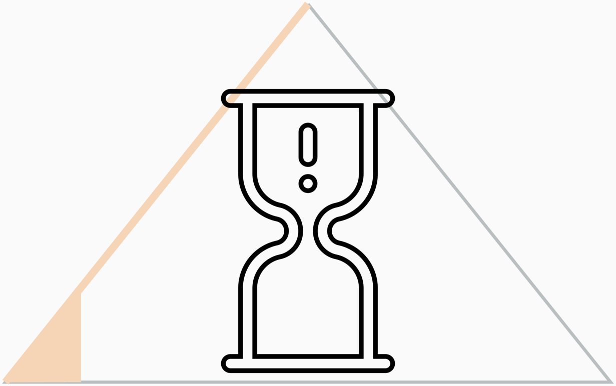


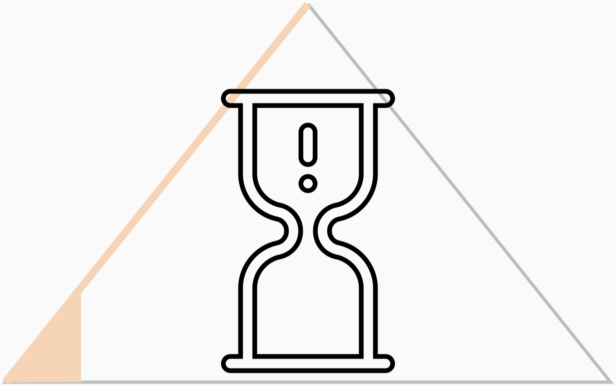
Search

Huge search space



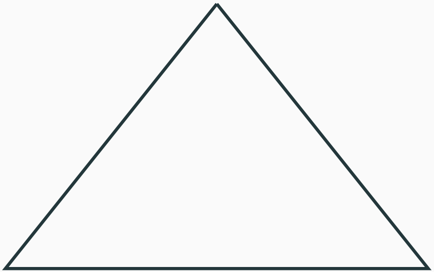
Huge search space



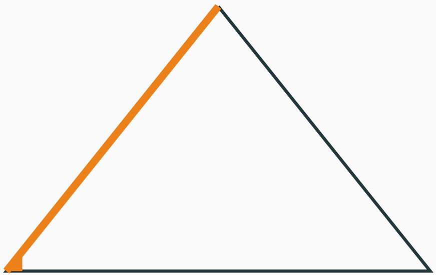


What to do when the search space is too big?

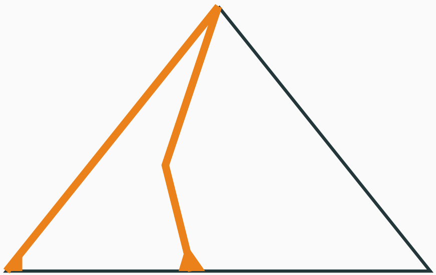
Incomplete search



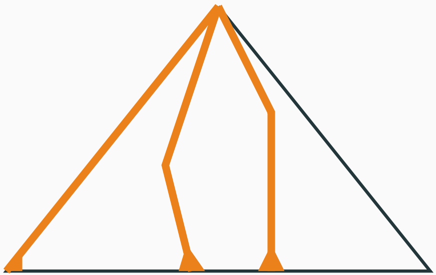
Search

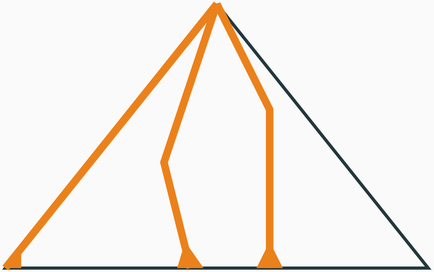


Incomplete search

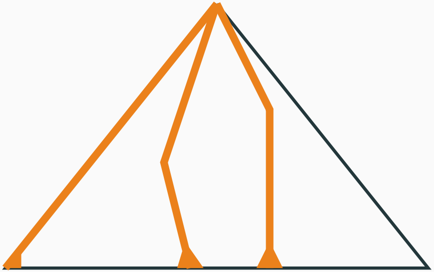


Search



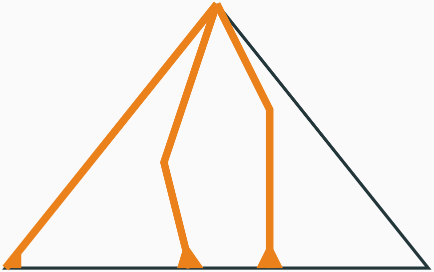


Meta-heuristics:



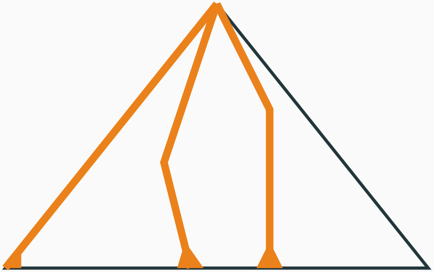
Meta-heuristics:

- Resets with non-deterministic searches



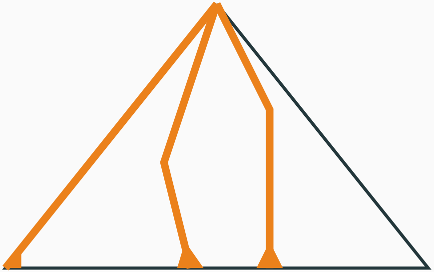
Meta-heuristics:

- Resets with non-deterministic searches
- Large Neighborhood search (LNS): search for a good solution, relax part of the solution, restart from the partial solution



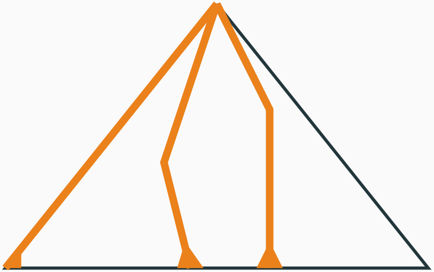
Meta-heuristics:

- Resets with non-deterministic searches
- Large Neighborhood search (LNS): search for a good solution, relax part of the solution, restart from the partial solution
- Portfolio searches: try multiple search strategies for a bit of time



Meta-heuristics:

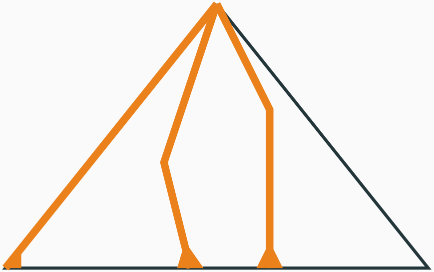
- Resets with non-deterministic searches
- Large Neighborhood search (LNS): search for a good solution, relax part of the solution, restart from the partial solution
- Portfolio searches: try multiple search strategies for a bit of time
- ...



Meta-heuristics:

- Resets with non-deterministic searches
- Large Neighborhood search (LNS): search for a good solution, relax part of the solution, restart from the partial solution
- Portfolio searches: try multiple search strategies for a bit of time
- ...

Idea: try a diversity of smaller subspace



Meta-heuristics:

- Resets with non-deterministic searches
- Large Neighborhood search (LNS): search for a good solution, relax part of the solution, restart from the partial solution
- Portfolio searches: try multiple search strategies for a bit of time
- ...

Idea: try a diversity of smaller subspace

No guarantee of optimality!

Hybrids of techniques

MDD propagators (with DD input)

Table constraint

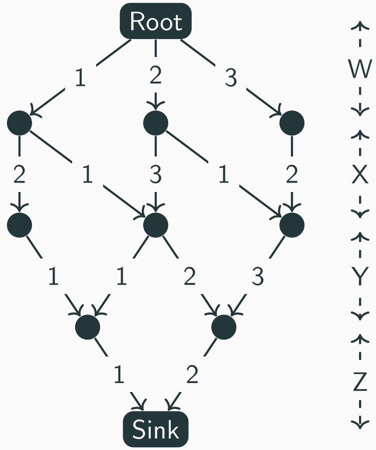
W	X	Y	Z
1	2	1	1
1	1	1	1
1	1	2	2
2	3	1	1
2	3	2	2
2	1	3	2
3	2	3	2

MDD propagators (with DD input)

Table constraint

W	X	Y	Z
1	2	1	1
1	1	1	1
1	1	2	2
2	3	1	1
2	3	2	2
2	1	3	2
3	2	3	2

MDD constraint

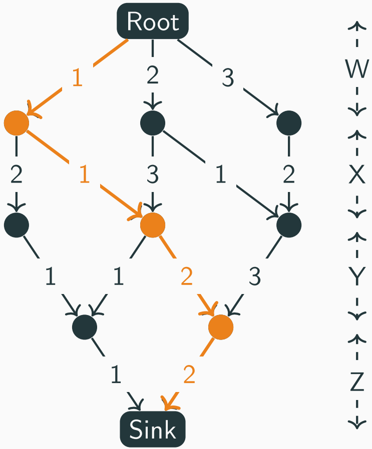


Reference: **"Compact-MDD: Efficiently Filtering (s) MDD Constraints with Reversible Sparse Bit-sets"**, by H. Verhaeghe, C. Lecoutre, P. Schaus, IJCAI18

Table constraint

W	X	Y	Z
1	2	1	1
1	1	1	1
1	1	2	2
2	3	1	1
2	3	2	2
2	1	3	2
3	2	3	2

MDD constraint



Reference: **"Compact-MDD: Efficiently Filtering (s) MDD Constraints with Reversible Sparse Bit-sets"**, by H. Verhaeghe, C. Lecoutre, P. Schaus, IJCAI18

Lazy Clause Generation (with SAT clauses)

UCLouvain

Constraints (2009) 14:357–391
DOI 10.1007/s10601-008-9064-x

Propagation via lazy clause generation

Olga Ohrimenko · Peter J. Stuckey · Michael Codish

Published online: 13 January 2009

Abstract Finite domain propagation solvers effectively represent the possible values of variables by a set of choices which can be naturally modelled as Boolean variables. In this paper we describe how to mimic a finite domain propagation engine, by mapping propagators into clauses in a SAT solver. This immediately results in strong

Proof Loging Solvers (with PB proofs)

Is the problem solvable?

Easy to prove, here is a solution!

Is the problem solvable?

Easy to prove, here is a solution!

Is the problem unsolvable?

Having no solution is not really a proof... maybe we did not find it?

Is the problem solvable?

Easy to prove, here is a solution!

Is the problem unsolvable?

Having no solution is not really a proof... maybe we did not find it?

Is the optimal really the optimal?

Well, I have my best so far... but again, maybe we did not find the best?

Proof Loging Solvers (with PB proofs)

Is the problem solvable?

s a solution!

Is the probl

not find it?

Is the optim

Well,

d the best?

An Auditable Constraint Programming Solver

Stephan Gocht
Lund University, Sweden
University of Copenhagen, Denmark

Ciaran McCreesh
University of Glasgow, UK

Jakob Nordström
University of Copenhagen, Denmark
Lund University, Sweden

Abstract

We describe the design and implementation of a new constraint programming solver that can produce an auditable record of what problem was solved and how the solution was reached. As well as a solution, this solver provides an independently verifiable proof log demonstrating that the solution is correct. This proof log uses the VeriPB proof system, which is based upon cutting planes reasoning with extension variables. We explain how this system can support global constraints, variables with

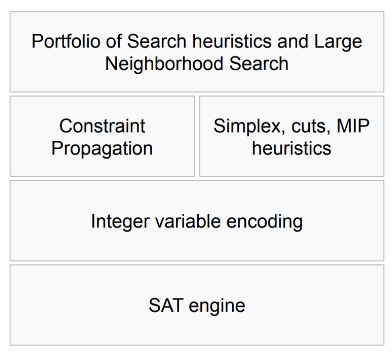
Portfolio solvers combine multiple strategies in parallel and share information (bounds, learned clauses,...) between the threads

Portfolio solvers combine multiple strategies in parallel and share information (bounds, learned clauses,...) between the threads

- Portfolio searches: Variety of searches at the same time

Portfolio solvers combine multiple strategies in parallel and share information (bounds, learned clauses,...) between the threads

- Portfolio searches: Variety of searches at the same time
- Hybrid CP-SAT-LP Or-Tools solver: specialised solvers in parallel



Or-Tools CP-SAT-LP

Conclusion

Conclusion

Conclusion

Solving a problem using CP:

Solving a problem using CP:

- **Step 1**: model the problem, using variables, constraints, and objective (if one)

Solving a problem using CP:

- **Step 1**: model the problem, using variables, constraints, and objective (if one)
- **Step 2**: select a search heuristic based on the knowledge of the problem (or use solver's default)

Solving a problem using CP:

- **Step 1**: model the problem, using variables, constraints, and objective (if one)
- **Step 2**: select a search heuristic based on the knowledge of the problem (or use solver's default)
- **Step 3**: solver searches for solution(s) using search strategies, fixed-point and propagators

Solving a problem using CP:

- **Step 1**: model the problem, using variables, constraints, and objective (if one)
- **Step 2**: select a search heuristic based on the knowledge of the problem (or use solver's default)
- **Step 3**: solver searches for solution(s) using search strategies, fixed-point and propagators

Constraint programming

Solving a problem using CP:

- **Step 1**: model the problem, using variables, constraints, and objective (if one)
- **Step 2**: select a search heuristic based on the knowledge of the problem (or use solver's default)
- **Step 3**: solver searches for solution(s) using search strategies, fixed-point and propagators

Constraint programming

- is **modular** and **versatile**

Solving a problem using CP:

- **Step 1**: model the problem, using variables, constraints, and objective (if one)
- **Step 2**: select a search heuristic based on the knowledge of the problem (or use solver's default)
- **Step 3**: solver searches for solution(s) using search strategies, fixed-point and propagators

Constraint programming

- is **modular** and **versatile**
- **can adapt** to one's needs

Thank you for listening!

Any questions?

`https://hverhaeghe.bitbucket.io/`