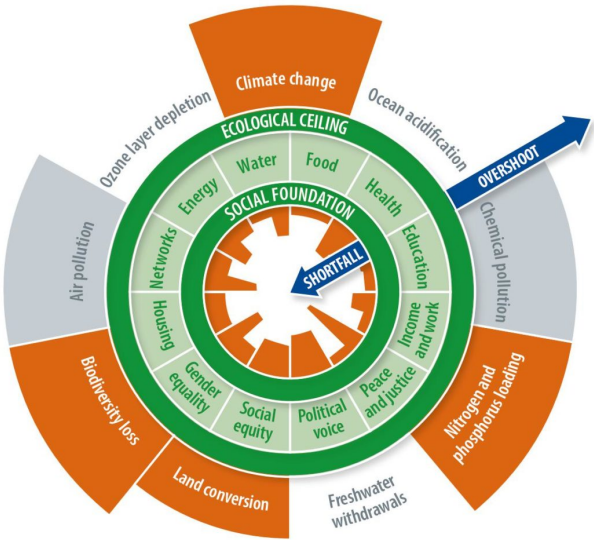
Constraint Programming:
What is it?
Can it help us meet everyone's needs within the planet's boundaries?

Christine Solnon
CITI, INSA Lyon / Inria

IA^2 2025 - Aussois

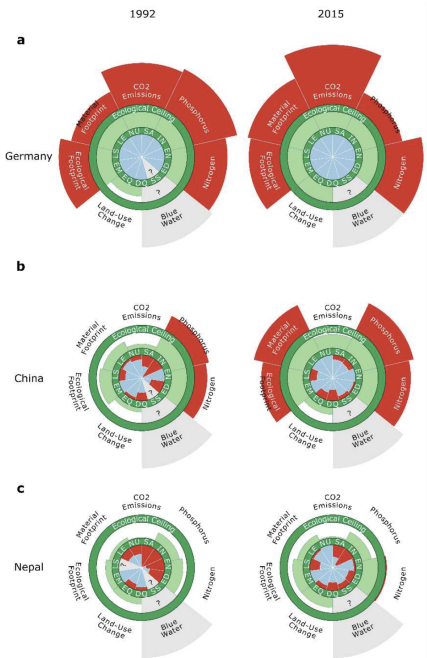
The doughnut of social and planetary boundaries



Source: Kate Raworth

Kate Raworth's Doughnut:

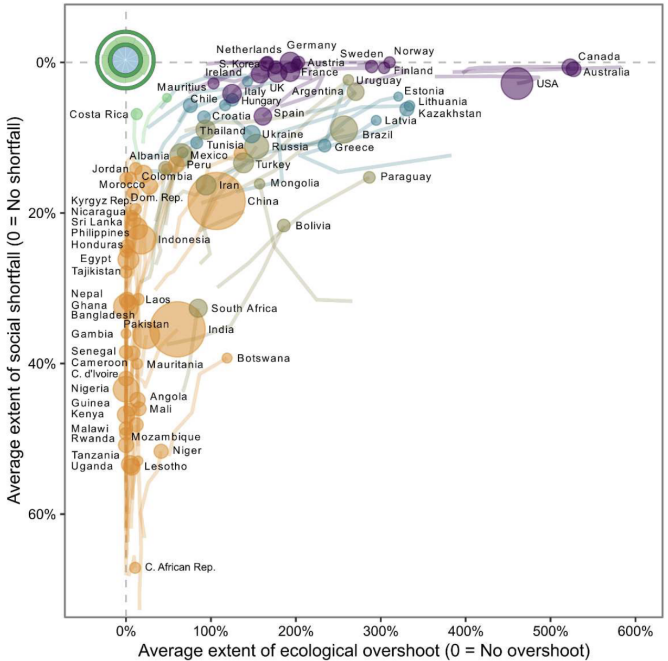
- 9 planet's boundaries
  $\rightsquigarrow$ 7 are crossed according to (Planetary health check, 2025)
- 12 social boundaries



Doughnuts depend on:

- Countries
  $\rightsquigarrow$ Germany vs China vs Nepal
- Time
  $\rightsquigarrow$ 1992 vs 2015

See (Fanning et al, 2022) for more details



Dynamics of countries from 1992 to 2015 (Fanning et al, 2022)

Can CP help us meet everyone's needs within the planet's boundaries?

OK: Our planet has limits, and some people are lacking access to life's essentials

(This was already well stated by Meadows et al, 1972)

- We need to ensure that planet and social limits are not overpassed
  $\rightsquigarrow$ Maximise efficiency and welfare
- This is a Constrained Optimization Problem!
  $\rightsquigarrow$ Can we use Constraint Programming (CP) to model and solve this problem?

Warning: I assume you already know ICT has huge impacts on planet's boundaries

What is Constraint Programming (CP)?

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

Eugene C. Freuder

In other words: CP = Model + Search

- Model $\rightsquigarrow$ Define a mathematical model by means of variables and constraints
- Search $\rightsquigarrow$ Use a generic solver to search for a solution

What is the difference with ILP (Integer Linear Programming) or SAT?

- Richer modelling language
  - ILP: Model = Linear inequations + Linear objective function
  - SAT: Model = Boolean formula
  - CP: Model = Conjunction of various kinds of constraints
    - $\rightsquigarrow$ Each constraint comes with its own propagation algorithm
    - $\rightsquigarrow$ If your favorite constraint does not exist, you can create it!
- Different solving approach: Branch & Propagate (vs Branch & Bound/Cut/Price for ILP or CDCL for SAT)

What is Constraint Programming (CP)?

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

Eugene C. Freuder

In other words: CP = Model + Search

- Model $\rightsquigarrow$ Define a mathematical model by means of variables and constraints
- Search $\rightsquigarrow$ Use a generic solver to search for a solution

What is the difference with ILP (Integer Linear Programming) or SAT?

- Richer modelling language
  - ILP: Model = Linear inequations + Linear objective function
  - SAT: Model = Boolean formula
  - CP: Model = Conjunction of various kinds of constraints
    $\rightsquigarrow$ Each constraint comes with its own propagation algorithm
    $\rightsquigarrow$ If your favorite constraint does not exist, you can create it!
- Different solving approach: Branch & Propagate (vs Branch & Bound/Cut/Price for ILP or CDCL for SAT)

Overview of the talk

Modelling with CP

CP model = $(X, D, C)[+F]$

- X = Set of variables (unknowns)
- For each variable $x_i \in X$, $D(x_i)$ = domain of x_i
  $\rightsquigarrow$ Set of values that may be assigned to x_i
- C = Constraints (relations between variables of X)
- [Optionally] $F : X \to \mathbb{R}$ = objective function to optimize

Solution of a CP model:

Assignment of a value to every variable of X such that:

- Each variable $x_i \in X$ is assigned to a value that belongs to $D(x_i)$
- Every constraint of C is satisfied
- [Optionally] F is maximized (or minimized)

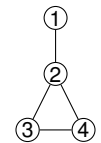
Remark: A problem may have several different models...

Example: Subgraph Isomorphism Problem (SIP)

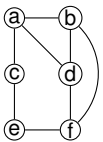
Definition of the SIP:

Given $G_p = (V_p, E_p)$ and $G_t = (V_t, E_t)$:
find an injective mapping $f : V_p \to V_t$ such that
$\forall (i, j) \in E_p, (f(i), f(j)) \in E_t$

Example of SIP instance:



$G_p = (V_p, E_p)$



$G_t = (V_t, E_t)$

1

Example: Subgraph Isomorphism Problem (SIP)

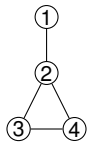
Definition of the SIP:

Given $G_p = (V_p, E_p)$ and $G_t = (V_t, E_t)$:
find an injective mapping $f : V_p \to V_t$ such that
$\forall (i,j) \in E_p, (f(i), f(j)) \in E_t$

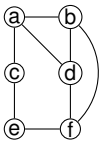
CP model introduced by Ullmann in 1976[1]:

- Variables: $X = \{x_i | i \in V_p\}$
- Domains:
  $\forall i \in V_p, D(x_i) = \{u \in V_t : d^\circ(i) \leq d^\circ(u)\}$
- Constraints:
  - f must be injective: $\forall \{i,j\} \subseteq V_p, x_i \neq x_j$
  - Edge constraints:
    $\forall (i,j) \in E_p, (x_i, x_j) \in E_t$

Example of SIP instance:



$G_p = (V_p, E_p)$



$G_t = (V_t, E_t)$

Domains:

- $D(x_1) = D(x_3) = D(x_4) = \{a, b, c, d, e, f\}$
- $D(x_2) = \{a, b, d, f\}$

1 Ullmann: *An algorithm for subgraph isomorphism*, in J. ACM, 1976

Example: Subgraph Isomorphism Problem (SIP)

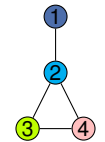
Definition of the SIP:

Given $G_p = (V_p, E_p)$ and $G_t = (V_t, E_t)$:
find an injective mapping $f : V_p \to V_t$ such that
$\forall (i,j) \in E_p, (f(i), f(j)) \in E_t$

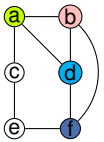
CP model introduced by Ullmann in 1976[1]:

- Variables: $X = \{x_i | i \in V_p\}$
- Domains:
  $\forall i \in V_p, D(x_i) = \{u \in V_t : d^\circ(i) \leq d^\circ(u)\}$
- Constraints:
  - f must be injective: $\forall \{i,j\} \subseteq V_p, x_i \neq x_j$
  - Edge constraints:
    $\forall (i,j) \in E_p, (x_i, x_j) \in E_t$

Example of SIP instance:



$G_p = (V_p, E_p)$



$G_t = (V_t, E_t)$

Domains:

- $D(x_1) = D(x_3) = D(x_4) = \{a, b, c, d, e, f\}$
- $D(x_2) = \{a, b, d, f\}$

Example of solution:

$x_1 = f, x_2 = d, x_3 = a, x_4 = b$

1 Ullmann: *An algorithm for subgraph isomorphism*, in J. ACM, 1976

Example: Subgraph Isomorphism Problem (SIP)

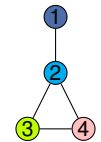
Definition of the SIP:

Given $G_p = (V_p, E_p)$ and $G_t = (V_t, E_t)$:
find an injective mapping $f : V_p \to V_t$ such that
$\forall (i, j) \in E_p, (f(i), f(j)) \in E_t$

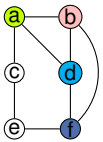
Other CP model with binary variables:

- Variables: $X = \{x_{ij} | i \in V_p, j \in V_t\}$
- Domains: $\forall i \in V_p, \forall j \in V_t, D(x_{ij}) = \{0, 1\}$
- Constraints:
  - f must be injective:
    $\forall i \in V_p, \sum_j x_{ij} = 1$
    $\forall j \in V_t, \sum_i x_{ij} \leq 1$
  - Edge constraints:
    $\forall (i_1, i_2) \in E_p, \forall (j_1, j_2) \in$
    $V_t \times V_t \setminus E_t, x_{i_1 j_1} + x_{i_2 j_2} < 2$

Example of SIP instance:



$G_p = (V_p, E_p)$



$G_t = (V_t, E_t)$

Example of solution:

- $x_{1f} = x_{2d} = x_{3a} = x_{4b} = 1$
- All other variables are set to 0

[1] Ullmann: *An algorithm for subgraph isomorphism*, in J. ACM, 1976

CP Languages and Libraries

First CP language introduced by Jean-Louis Laurière in 1976:

ALICE

Extensions of Prolog:

CHIP, Prolog V, Gnu-Prolog, Sicstus Prolog, Picat, ...

Libraries:

- Open source: Choco (Java), PyCSP3 (Python), OR-Tools/Google (C++, Java, Python, ...), etc
- Commercial: CP-Optimizer (IBM)

Modelling languages (accepted by most solvers):

MiniZinc, XCSP3

Example: SIP in PyCSP3 (using a notebook)

Define the pattern graph and the target graph

```
[3] n_p = 4; n_t = 6
    e_p = [(0,1), (0,3), (1,2), (1,3), (2,3)]
    e_t = [(0,1), (0,4), (1,0), (1,2), (1,3), (2,1), (2,3), (2,5), (3,1), (3,2), (3,4), (4,0), (4,3), (5,1), (5,2)]
    deg_p = [2, 3, 2, 3]
    deg_t = [2, 4, 3, 3, 2, 2]
```

Declare variables and domains

```
[4] x = VarArray(size=n_p, dom=range(n_t))
    for i in range(n_p) :
      x[i] in {j for j in range(n_t) if deg_p[i] <= deg_t[j]}
```

Declare constraints... and solve!

```
[6] satisfy(AllDifferent(x),
            [(x[i], x[j]) in e_t for (i, j) in e_p])
    if solve() is SAT:
        print(values(x))
```

```
[5, 1, 3, 2]
```

Overview of the talk

Branch & Propagate

Branch:

- Choose an unassigned variable x_i
- For each $v \in D(x_i)$, recursively solve a subproblem where v is assigned to x_i

$\rightsquigarrow$ Stop when a solution is found; Backtrack when an inconsistency is detected

Propagate constraints at each subproblem:

- Goal: Remove inconsistent values from variable domains
  $\rightsquigarrow$ Inconsistency detected whenever a domain is wiped out
- Each constraint comes with its own propagation algorithms
  $\rightsquigarrow$ Easy to add new constraints

Branch & Propagate & Bound

Branch:

- Choose an unassigned variable x_i
- For each $v \in D(x_i)$, recursively solve a subproblem where v is assigned to x_i

$\rightsquigarrow$ **Bound** when a solution is found; Backtrack when an inconsistency is detected

Propagate constraints at each subproblem:

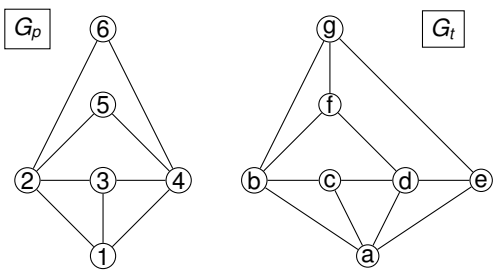
- Goal: Remove inconsistent values from variable domains
  $\rightsquigarrow$ Inconsistency detected whenever a domain is wiped out
- Each constraint comes with its own propagation algorithms
  $\rightsquigarrow$ Easy to add new constraints

A first (very simple) propagation: Forward Checking (FC)

Propagate a constraint whenever all its variables but one are assigned

$\rightsquigarrow$ Remove inconsistent values from the domains of unassigned variables

Example of SIP instance:



Initial domains:

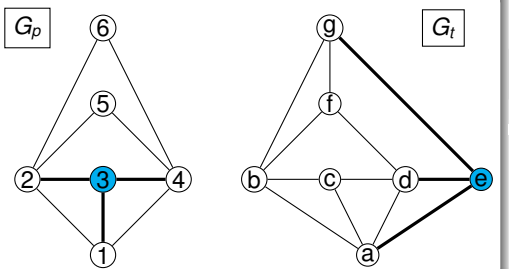
- $D(x_1) = D(x_3) = D(x_5) = D(x_6) = \{a, b, c, d, e, f, g\}$
- $D(x_2) = D(x_4) = \{a, b, d\}$

A first (very simple) propagation: Forward Checking (FC)

Propagate a constraint whenever all its variables but one are assigned

$\rightsquigarrow$ Remove inconsistent values from the domains of unassigned variables

Example of SIP instance:



Initial domains:

- $D(x_1) = D(x_3) = D(x_5) = D(x_6) = \{a, b, c, d, e, f, g\}$
- $D(x_2) = D(x_4) = \{a, b, d\}$

FC propagation when $x_3 = e$:

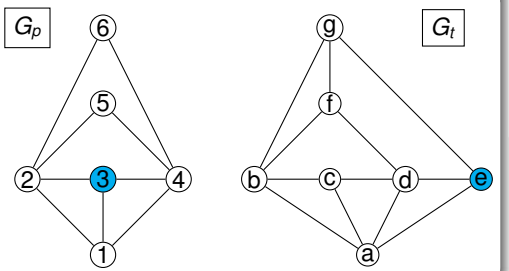
- $\forall i \in \{1, 5, 6\}, x_i \neq e$
  $\rightsquigarrow$ Remove e from $D(x_i)$
- $\forall i \in \{1, 2, 4\}, (x_i, e) \in E_t$
  $\rightsquigarrow$ Remove b, c, and f from $D(x_1)$
  $\rightsquigarrow$ Remove b from $D(x_2)$ and $D(x_4)$

Stronger propagation: Ensuring Domain Consistency

A constraint c defined over a set $S \subseteq X$ of k variables is Domain Consistent (DC) if:

$\forall x_i \in S, \forall v_i \in D(x_i), \forall x_j \in S \setminus \{x_i\}, \exists v_j \in D(x_j)$ such that c is satisfied by the assignment $x_1 = v_1, ..., x_k = v_k$

Example of SIP instance:



Let's assume that $x_3 = e$.

Domains reduced by FC in this case:

- $D(x_1) = \{a, d, g\}$
- $D(x_2) = \{a, d\}$
- $D(x_4) = \{a, d\}$
- $D(x_5) = D(x_6) = \{a, b, c, d, f, g\}$

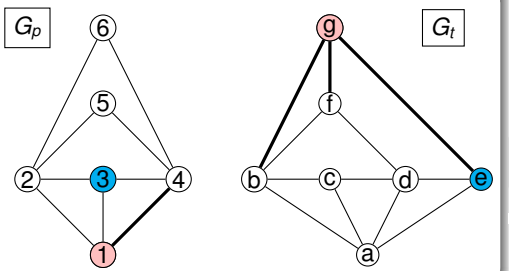
Can we do better?

Stronger propagation: Ensuring Domain Consistency

A constraint c defined over a set $S \subseteq X$ of k variables is Domain Consistent (DC) if:

$\forall x_i \in S, \forall v_i \in D(x_i), \forall x_j \in S \setminus \{x_i\}, \exists v_j \in D(x_j)$ such that c is satisfied by the assignment $x_1 = v_1, ..., x_k = v_k$

Example of SIP instance:



Let's assume that $x_3 = e$.

Domains reduced by FC in this case:

- $D(x_1) = \{a, d, g\}$
- $D(x_2) = \{a, d\}$
- $D(x_4) = \{a, d\}$
- $D(x_5) = D(x_6) = \{a, b, c, d, f, g\}$

Can we do better?

DC propagation of the constraint "$(x_1, x_4) \in E_t$":

If $x_1 = g$ then x_4 cannot be assigned to a neighbour of g
$\rightsquigarrow$ Remove g from $D(x_1)$

Historical Notes on DC (aka as Arc Consistency)

- DC has been introduced by Ullmann in 1976 for solving the SIP[1]
- AC3 is the first "efficient" algorithm to ensure DC for binary constraints
  - Introduced by Mackworth in 1977[2]
  - Time complexity in $\mathcal{O}(ed^3)$ where e = number of constraints and d = maximum domain size
  - Space complexity in $\mathcal{O}(e)$
- AC4 is introduced by Mohr & Henderson in 1986[3]
  - Space and time complexities in $\mathcal{O}(ed^2)$
- Many (really many) algorithms introduced since then, with different time and space tradeoffs
  $\rightsquigarrow$ See the last volume of TAOCP[4]

1 Ullmann. *An algorithm for subgraph isomorphism*, in J. ACM, 1976
2 Mackworth. *Consistency in networks of relations*, in Artificial Intelligence, 1977
3 Mohr & Henderson. *Arc and path consistency revisited*, in Artificial Intelligence, 1986
4 Knuth. *The Art of Computer Programming. Section 7.2.2.3: Constraint Satisfaction*, 2025

Global constraints

What is a global constraint?

Constraint defined over a set of variables (the cardinality of which is not fixed)

Examples of global constraints:

- allDifferent$(x_1, \ldots, x_n) \Leftrightarrow \forall \{x_i, x_j\} \subseteq \{x_1, \ldots, x_n\}, x_i \neq x_j$
- sum$(x_1, \ldots, x_n, s) \Leftrightarrow \sum_{i=1}^{n} x_i = s$
- atLeast$(x_1, \ldots, x_n, k, v) \Leftrightarrow (\sum_{i=1}^{n} [[x_i = v]]) \geq k$ where $[[x_i = v]] = 1$ if $x_i = v$, and 0 otherwise
- ... and many others (see our catalog of 423 global constraints[1])

$\rightsquigarrow$ Ease the modelling step

How to propagate a global constraint?

- First possibility: Decompose it into existing constraints and use existing propagators
- Second possibility: Design a dedicated propagator

1 https://sofdem.github.io/gccat/

Decomposition of global constraints

How to decompose a global constraint?

Replace the constraint with an equivalent set of fixed-arity constraints

Example 1: allDifferent$(x_1, \ldots, x_n)$

- $\forall \{x_i, x_j\} \subseteq \{x_1, \ldots, x_n\}, x_i \neq x_j$

Example 2: atLeast$(x_1, \ldots, x_n, k, v)$

Introduction of n new variables $s_1, \ldots, s_n$

- $s_1 = [[x_1 = v]]$
- $\forall i \in \{2, \ldots, n\}, s_i = s_{i-1} + [[x_i = v]]$
- $s_n \geq k$

Example 3: sum$(x_1, \ldots, x_n, s)$

Introduction of n new variables $s_1, \ldots, s_n$

- $s_1 = x_1$
- $\forall i \in \{2, \ldots, n\}, s_i = s_{i-1} + x_i$
- $s = s_n$

Should we propagate the global constraint or its decomposition?

DC-decomposable constraint

There exists a polynomial size decomposition such that the decomposition is DC iff the global constraint is DC

Example: atLeast$(x_1, \ldots, x_n, k, v)$ is DC-decomposable

Propagating the constraints $s_i = s_{i-1} + [[x_i = v]]$ filters the same values as propagating atLeast$(x_1, \ldots, x_n, k, v)$ (though it may be less efficient...)

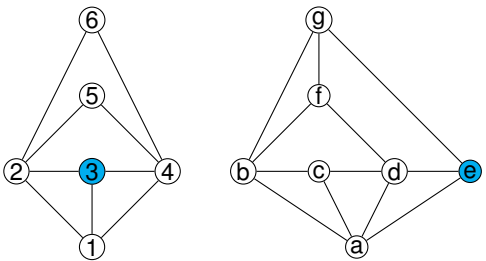
Example: sum$(x_1, \ldots, x_n, s)$ is not DC-decomposable

Proof: Deciding if an instance of sum is DC is an $\mathcal{NP}$-complete problem
$\rightsquigarrow$ Reduction from subset-sum, for example

What about *allDifferent*? Is it DC-decomposable?

Binary decomposition of allDifferent$(x_1, \ldots, x_n)$**:** $\forall \{x_i, x_j\} \subseteq \{x_1, \ldots, x_n\}, x_i \neq x_j$

Let's look again at our SIP instance:



$G_p = (V_p, E_p)$ $G_t = (V_t, E_t)$

Domains after propagation of edge constraints when $D(x_3) = \{e\}$**:**

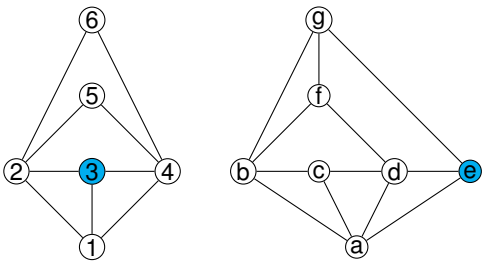
- $D(x_1) = D(x_2) = D(x_4) = \{a, d\}$
- $D(x_3) = \{e\}$
- $D(x_5) = D(x_6) = \{a, b, c, d, f, g\}$

Is the binary decomposition of allDifferent DC?

What about *allDifferent*? Is it DC-decomposable?

Binary decomposition of allDifferent$(x_1, \ldots, x_n)$**:** $\forall \{x_i, x_j\} \subseteq \{x_1, \ldots, x_n\}, x_i \neq x_j$

Let's look again at our SIP instance:



$G_p = (V_p, E_p)$ $G_t = (V_t, E_t)$

Domains after propagation of edge constraints when $D(x_3) = \{e\}$**:**

- $D(x_1) = D(x_2) = D(x_4) = \{a, d\}$
- $D(x_3) = \{e\}$
- $D(x_5) = D(x_6) = \{a, b, c, d, f, g\}$

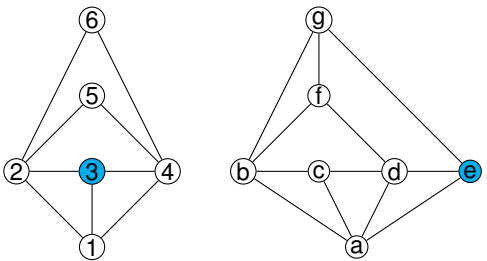
Is the binary decomposition of allDifferent DC?

Yes, because for each constraint $x_i \neq x_j$, we have:
$\forall v_i \in D(x_i), \exists v_j \in D(x_j), v_i \neq v_j$

What about *allDifferent*? Is it DC-decomposable?

Binary decomposition of allDifferent$(x_1, \ldots, x_n)$**:** $\forall \{x_i, x_j\} \subseteq \{x_1, \ldots, x_n\}, x_i \neq x_j$

Let's look again at our SIP instance:



$G_p = (V_p, E_p)$ $G_t = (V_t, E_t)$

Domains after propagation of edge constraints when $D(x_3) = \{e\}$**:**

- $D(x_1) = D(x_2) = D(x_4) = \{a, d\}$
- $D(x_3) = \{e\}$
- $D(x_5) = D(x_6) = \{a, b, c, d, f, g\}$

Is the binary decomposition of allDifferent DC?

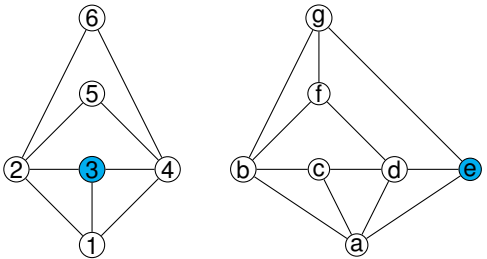
Yes, because for each constraint $x_i \neq x_j$, we have:
$\forall v_i \in D(x_i), \exists v_j \in D(x_j), v_i \neq v_j$

Is allDifferent$(\{x_1, x_2, ..., x_6\})$ **DC?**

What about *allDifferent*? Is it DC-decomposable?

Binary decomposition of allDifferent$(x_1, \ldots, x_n)$**:** $\forall \{x_i, x_j\} \subseteq \{x_1, \ldots, x_n\}, x_i \neq x_j$

Let's look again at our SIP instance:



$G_p = (V_p, E_p)$ $G_t = (V_t, E_t)$

Domains after propagation of edge constraints when $D(x_3) = \{e\}$**:**

- $D(x_1) = D(x_2) = D(x_4) = \{a, d\}$
- $D(x_3) = \{e\}$
- $D(x_5) = D(x_6) = \{a, b, c, d, f, g\}$

Is the binary decomposition of allDifferent DC?

Yes, because for each constraint $x_i \neq x_j$, we have:
$\forall v_i \in D(x_i), \exists v_j \in D(x_j), v_i \neq v_j$

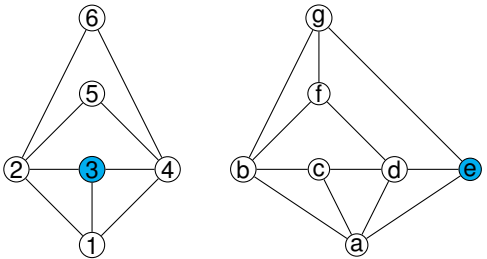
Is allDifferent$(\{x_1, x_2, ..., x_6\})$ **DC?**

- No: DC on the binary decomposition is weaker than DC on allDifferent!
- Can we find a decomposition preserving DC?

What about *allDifferent*? Is it DC-decomposable?

Binary decomposition of allDifferent$(x_1, \ldots, x_n)$**:** $\forall \{x_i, x_j\} \subseteq \{x_1, \ldots, x_n\}, x_i \neq x_j$

Let's look again at our SIP instance:



$G_p = (V_p, E_p)$ $G_t = (V_t, E_t)$

Domains after propagation of edge constraints when $D(x_3) = \{e\}$**:**

- $D(x_1) = D(x_2) = D(x_4) = \{a, d\}$
- $D(x_3) = \{e\}$
- $D(x_5) = D(x_6) = \{a, b, c, d, f, g\}$

Is the binary decomposition of allDifferent DC?

Yes, because for each constraint $x_i \neq x_j$, we have:
$\forall v_i \in D(x_i), \exists v_j \in D(x_j), v_i \neq v_j$

Is allDifferent$(\{x_1, x_2, ..., x_6\})$ **DC?**

- No: DC on the binary decomposition is weaker than DC on allDifferent!
- Can we find a decomposition preserving DC? **No!**[1]

[1] Bessière et al. *Circuit Complexity and Decompositions of Global Constraints*, in IJCAI 2009

Dedicated propagation algorithm for ensuring DC of allDifferent[1]

Suppose that:

- $D(x_1) = \{a, b, d\}$
- $D(x_2) = D(x_3) = \{b, c\}$
- $D(x_4) = \{c, d, e\}$

Propagation of allDifferent(x_1, x_2, x_3, x_4)?

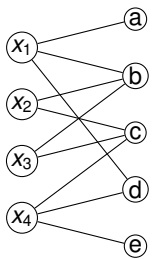
$\rightsquigarrow$ Remove values that cannot belong to a solution

1 Régin. *A filtering algorithm for constraints of difference in CSPs*, in AAAI 1994

Dedicated propagation algorithm for ensuring DC of allDifferent[1]

Suppose that:

- $D(x_1) = \{a, b, d\}$
- $D(x_2) = D(x_3) = \{b, c\}$
- $D(x_4) = \{c, d, e\}$



Propagation of allDifferent(x_1, x_2, x_3, x_4)?

$\rightsquigarrow$ Remove values that cannot belong to a solution

Algorithm for propagating allDifferent:

- Build a variable/value bipartite graph
- Add a source r and a sink s
- Search for a maximum flow from r to s
  $\rightsquigarrow$ Algorithm of Hopcroft & Karp[2] in $\mathcal{O}(p^{5/2})$
  $\rightsquigarrow$ Inconsistency if $|f| < |X|$
- Build the "final flow graph" and search for SCC
  $\rightsquigarrow$ Algorithm of Tarjan[3] in $\mathcal{O}(p)$
- Remove i from $D(x_j)$ if $\mathrm{SCC}(i) \neq \mathrm{SCC}(x_j)$[3]
  $\rightsquigarrow$ Remove b from $D(x_1)$ and c from $D(x_4)$

1 Régin. *A filtering algorithm for constraints of difference in CSPs*, in AAAI 1994

2

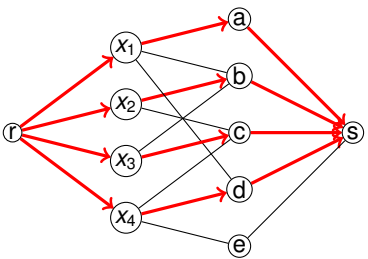
3

4

Dedicated propagation algorithm for ensuring DC of allDifferent[1]

Suppose that:

- $D(x_1) = \{a, b, d\}$
- $D(x_2) = D(x_3) = \{b, c\}$
- $D(x_4) = \{c, d, e\}$



Propagation of allDifferent(x_1, x_2, x_3, x_4)?

$\rightsquigarrow$ Remove values that cannot belong to a solution

Algorithm for propagating allDifferent:

- Build a variable/value bipartite graph
- Add a source r and a sink s
- Search for a maximum flow from r to s
  $\rightsquigarrow$ Algorithm of Hopcroft & Karp[2] in $\mathcal{O}(p^{5/2})$
  $\rightsquigarrow$ Inconsistency if $|f| < |X|$
- Build the "final flow graph" and search for SCC
  $\rightsquigarrow$ Algorithm of Tarjan[3] in $\mathcal{O}(p)$
- Remove i from $D(x_j)$ if $\text{SCC}(i) \neq \text{SCC}(x_j)$[3]
  $\rightsquigarrow$ Remove b from $D(x_1)$ and c from $D(x_4)$

1 Régin. *A filtering algorithm for constraints of difference in CSPs*, in AAAI 1994

2 Hopcroft & Karp. *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, in SIAM J. Comput., 1973

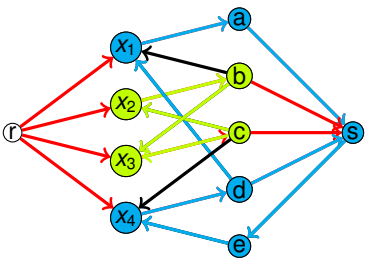
3

4

Dedicated propagation algorithm for ensuring DC of allDifferent[1]

Suppose that:

- $D(x_1) = \{a, b, d\}$
- $D(x_2) = D(x_3) = \{b, c\}$
- $D(x_4) = \{c, d, e\}$



Propagation of allDifferent(x_1, x_2, x_3, x_4)?

$\rightsquigarrow$ Remove values that cannot belong to a solution

Algorithm for propagating allDifferent:

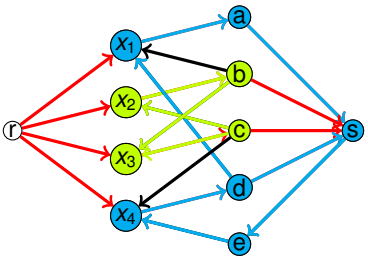
- Build a variable/value bipartite graph
- Add a source r and a sink s
- Search for a maximum flow from r to s
  $\rightsquigarrow$ Algorithm of Hopcroft & Karp[2] in $\mathcal{O}(p^{5/2})$
  $\rightsquigarrow$ Inconsistency if $|f| < |X|$
- Build the "final flow graph" and search for SCC
  $\rightsquigarrow$ Algorithm of Tarjan[3] in $\mathcal{O}(p)$
- Remove i from $D(x_j)$ if $\mathrm{SCC}(i) \neq \mathrm{SCC}(x_j)$[3]
  $\rightsquigarrow$ Remove b from $D(x_1)$ and c from $D(x_4)$

1 Régin. *A filtering algorithm for constraints of difference in CSPs*, in AAAI 1994
2 Hopcroft & Karp. *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, in SIAM J. Comput., 1973
3 Tarjan. *Depth-first search and linear graph algorithms*, in SIAM J. Comput., 1972
4

Dedicated propagation algorithm for ensuring DC of allDifferent[1]

Suppose that:

- $D(x_1) = \{a, b, d\}$
- $D(x_2) = D(x_3) = \{b, c\}$
- $D(x_4) = \{c, d, e\}$



Propagation of allDifferent(x_1, x_2, x_3, x_4)?

$\rightsquigarrow$ Remove values that cannot belong to a solution

Algorithm for propagating allDifferent:

- Build a variable/value bipartite graph
- Add a source r and a sink s
- Search for a maximum flow from r to s
  $\rightsquigarrow$ Algorithm of Hopcroft & Karp[2] in $\mathcal{O}(p^{5/2})$
  $\rightsquigarrow$ Inconsistency if $|f| < |X|$
- Build the "final flow graph" and search for SCC
  $\rightsquigarrow$ Algorithm of Tarjan[3] in $\mathcal{O}(p)$
- Remove i from $D(x_j)$ if SCC$(i) \neq$SCC(x_j)[3]
  $\rightsquigarrow$ Remove b from $D(x_1)$ and c from $D(x_4)$

1 Régin. *A filtering algorithm for constraints of difference in CSPs*, in AAAI 1994
2 Hopcroft & Karp. *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, in SIAM J. Comput., 1973
3 Tarjan. *Depth-first search and linear graph algorithms*, in SIAM J. Comput., 1972
4 Berge. *Graphs and Hypergraphs*, 1973

Ordering heuristics

Branch (recall):

- Choose an unassigned variable x_i
- For each $v \in D(x_i)$, recursively solve a subproblem where v is assigned to x_i

$\rightsquigarrow$ Stop when a solution is found; Backtrack when an inconsistency is detected

Classical variable ordering heuristics:

- *deg*: Variable involved in the largest number of constraints $\rightsquigarrow$ Reduce tree depth
- *dom*: Variable with the smallest domain $\rightsquigarrow$ Reduce tree width
- $\frac{dom}{deg}$: Compromise between *dom* and *deg*
- $\frac{dom}{wdeg}$: Each constraint has a weight (incremented on failures)
  $\rightsquigarrow$ divide $|D(x_i)|$ by sum of weights of constraints associated with x_i[1]

1 Boussemart et al. *Boosting systematic search by weighting constraints*, in ECAI 2004

2

Ordering heuristics

Branch (recall):

- Choose an unassigned variable x_i
- For each $v \in D(x_i)$, recursively solve a subproblem where v is assigned to x_i

$\rightsquigarrow$ Stop when a solution is found; Backtrack when an inconsistency is detected

Value ordering heuristics:

Choose values that are more likely to belong to solutions
$\rightsquigarrow$ Useless for proving inconsistency of infeasible instances

- Can be learned... but this may be expensive
- Select values in the best found solution (for optimization problems)[2]

1 Boussemart et al. *Boosting systematic search by weighting constraints*, in ECAI 2004
2 Demirovic et al. *Solution-based phase saving for CP: A value-selection heuristic to simulate local search*, in CP 2018

Overview of the talk

1. Modelling with CP
2. Generic CP Solving Algorithms
3. **Can CP help us meet everyone's needs within the planet's boundaries?**
4. Conclusion

Some "classical" applications of CP (and OR)

- Car sequencing
- Scheduling
- Pricing
- ... and many others ...
- Packing
- Vehicle Routing
- Picking

Questions:

- What are their positive and negative impacts on planet and social boundaries?
- Can we add constraints to forbid negative rebound effects?
- Should we collectively choose the constraints to be imposed to get back within planet and social boundaries, or go on our business as usual and suffer the consequences?
- What values do we want to defend? Do our tools allow us to defend them?
  What values are carried by our tools?

Atelier sur la pensée industrielle de Hadrien Cambazard

https://tools.caseine.org/pensee-industrielle/index.html

Examples of applications for minimising carbon footprint

- Optimisation of the use of energy resources to meet the electricity demand [1]
- Minimisation of energy requirements of data-centers[2]
- Hardware optimisation: Minimise the number of additions performed by filters
- ...

Any reaction?

1 De Lara et al. *Optimization Methods for the Smart Grid*, in Report commissioned by the Conseil Français de l'Énergie, 2016
2 De Cauwer et al. *Proactive Workload Consolidation for Reducing Energy Cost over a Given Time Horizon*, in IEEE ISCCGC 2014
3 Garcia et al. *Hardware-aware Design of Multiplierless Second-Order IIR Filters with Minimum Adders*, in IEEE TSP 2022

Efficience vs Frugalité

AFNOR SPEC 2314 : Référentiel général pour l'IA frugale - Mesurer et réduire l'impact environnemental de l'IA

	Définition	Notions connexes	Raisonnement	Approche	Précisions
Efficience	Aptitude à optimiser les moyens alloués pour atteindre un résultat défini	Efficacité, optimisation	En relatif/par unité d'usage Le besoin prime : optimisation d'une solution jugée celle répondant le mieux au besoin	Recherche d'un optimum local ou d'un compromis sur un niveau de résultat fortement contraint	Prise en compte des effets de premier ordre pour les minimiser Prise en compte des parties prenantes de l'IA
Frugalité	Aptitude à se contenter d'un niveau de résultat jugé suffisant en redéfinissant les usages et les besoins	Sobriété (ou *Sufficiency* [10] en anglais)	En global La contrainte sur les ressources prime : recherche de la solution utilisant le moins de ressources possible et apportant une réponse satisfaisante au besoin	Recherche d'un optimum global ou d'un compromis large sur un niveau de résultat, ce qui nécessite d'élargir ou d'assouplir le besoin	Prise en compte des effets de premier ordre et de second ordre pour minimiser les impacts environnementaux négatifs Prise en compte de tous les acteurs au-delà des seules parties prenantes de l'IA

Examples of applications that aim at addressing some boundaries

- Kunming-Montreal Global Biodiversity Framework: Protect 30% of ecosystems worldwide
  - ⇝ Maintain and restore ecological connectivity between natural habitats and protected area[1]
  - ⇝ Protection of freshwater fish species [2]
- Kidney exchange programs: Maximise the number of transplantations through cycles of kidney swaps[3]
- Crop allocation problem[4]
- Fair and collaborative transportation in Short Food Supply Chains[5]
- Equitable management of microtransit demand[6]
- ...

1 Le Bozec-Chiffoleau et al. *Towards the 30 by 30 Kunming-Montreal Global Biodiversity Framework Target: Optimising Graph Connectivity in Constraint-Based Spatial Planning*, in IJCAI 2025

2 Mao et al. *Expanding Connected Components from Alternative Terminals: Global Optimization for Freshwater Fishes Under the UN's 30×30 Conservation Goal* , in IJCAI 2025

3 Barkel et al. *Operational research approaches and mathematical models for kidney exchange: A literature survey and empirical evaluation*, in EJOR 2025

4 Challand et al. *Supporting Sustainable Agroecological Initiatives for Small Farmers through Constraint Programming*, in IJCAI 2023

5 Besson et al. *Traveling salesman games: semicore allocations for collaborative transportation in short food supply chains*, 2024

6 Bardaka et al. *Empathy and AI: Achieving Equitable Microtransit for Underserved Communities*, in IJCAI 2024

Overview of the talk

Conclusion

CP allows us to easily solve combinatorial optimisation problems

- High-level modelling languages for defining constraints and objective functions
- Generic solving engines

$\rightsquigarrow$ Nice tool for exploring alternative scenarios

But designing the "right" model is a hard task!

- Identify **all** stakeholders together with their interests
  $\rightsquigarrow$ Forgotten stakeholders are usually strongly negatively impacted
- There are usually several conflicting objectives (e.g., social vs planet boundaries)
  - Compute a set of solutions corresponding to different compromises?
  - Transform some objectives into constraints?
  - ...
- How to evaluate systemic impacts of decisions on society and how to integrate them in the model?
- ...