Docker setup

You can start

```
docker pull \
registry.gitlab.inria.fr/soliman/inf555/td6
```

now

Global Constraints

Sylvain Soliman

October 24th, 2018

Thanks to J.-C. Régin, P. van Hentenryck and C. Bessiere for inspiration

and to C. Berge for genius

What's a global constraint?

Constraints that can involve any number n of variables (i.e., not only binary)

Complex relations between variables, **useful** in applications (e.g.)

With better/**more powerful** propagation than binary constraints (solved open problems about sport scheduling)

Which require *ad-hoc* AC algorithms (otherwise $|D|^n$)

What's a global constraint?

Constraints that can involve any number n of variables (i.e., not only binary)

Complex relations between variables, **useful** in applications (e.g. `alldifferent`)

With better/**more powerful** propagation than binary constraints (solved open problems about sport scheduling)

Which require *ad-hoc* AC algorithms (otherwise $|D|^n$)

Arc-consistency (Domain-consistency)

Obviously

$x_1 \neq x_2 \wedge x_2 \neq x_3 \wedge x_3 \neq x_1$
is **arc-consistent** on the domains $\{0, 1\}$

whereas

`alldifferent`(x_1, x_2, x_3)
is **not**

Demo


Knapsack?

The knapsack problem

Wikipedia:

> *The knapsack problem has been studied for more than a century, with early works dating as far back as 1897. The name* knapsack problem *dates back to the early works of mathematician Tobias Dantzig (1884–1956), and refers to the commonplace problem of packing the most valuable or useful items without overloading the luggage.*

A knapsack global constraint

- **Original optimization problem:** Given n items of weights w_i and value v_i fit as much value as possible in a knapsack of capacity S.

- **Derived decision problem:** Given n items of weights w_i can we chose some of them (x_i) such that they fit in a knapsack with bounded capacity.

  $$l \leq \sum_{i=1}^{n} w_i x_i \leq U$$

  Global constraint on the variables

A knapsack global constraint

- **Original optimization problem:** Given n items of weights w_i and value v_i fit as much value as possible in a knapsack of capacity S.

- **Derived decision problem:** Given n items of weights w_i can we chose some of them (x_i) such that they fit in a knapsack with bounded capacity.

  $$l \leq \sum_{i=1}^{n} w_i x_i \leq U$$

  Global constraint on the variables $x_i \in \{0, 1\}$

How can we filter?

Brute force enumeration is not great: decision is **NP-complete**

Take inspiration from the optimization problem

Pseudo-polynomial dynamic algorithm in $O(nU)$: build a graph (*forward phase*) and find a shortest path in it

We keep a (simplified) forward phase, but add a backward phase to remove paths incompatible with the $[l, U]$ constraint.
Then prune impossible values from the domain.

Example: $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$

0												

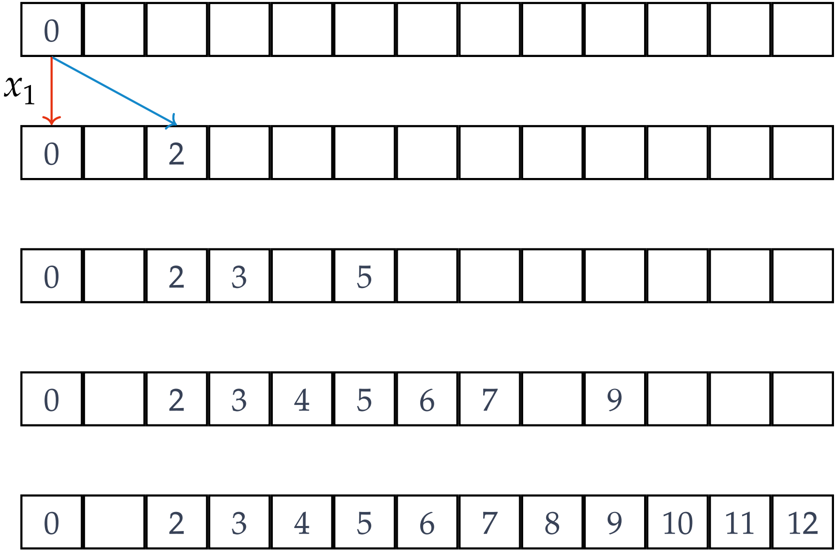
0		2										

0		2	3		5							

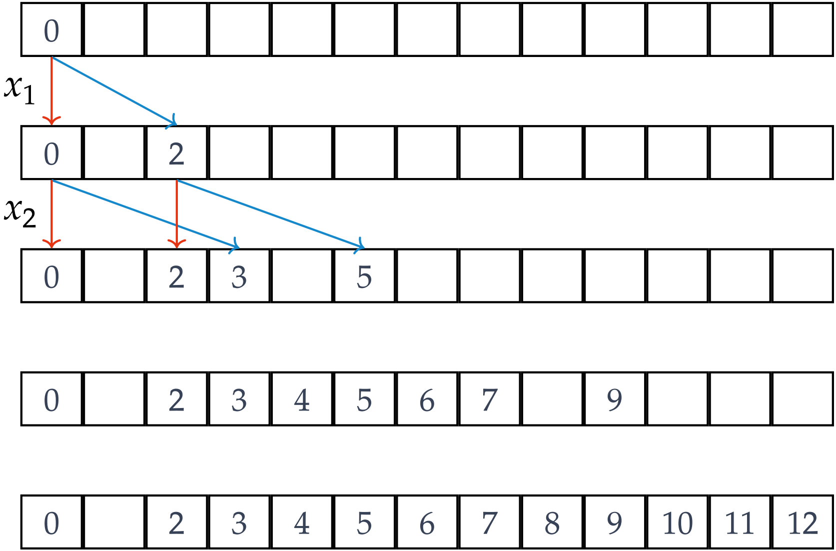
0		2	3	4	5	6	7		9			

0		2	3	4	5	6	7	8	9	10	11	12

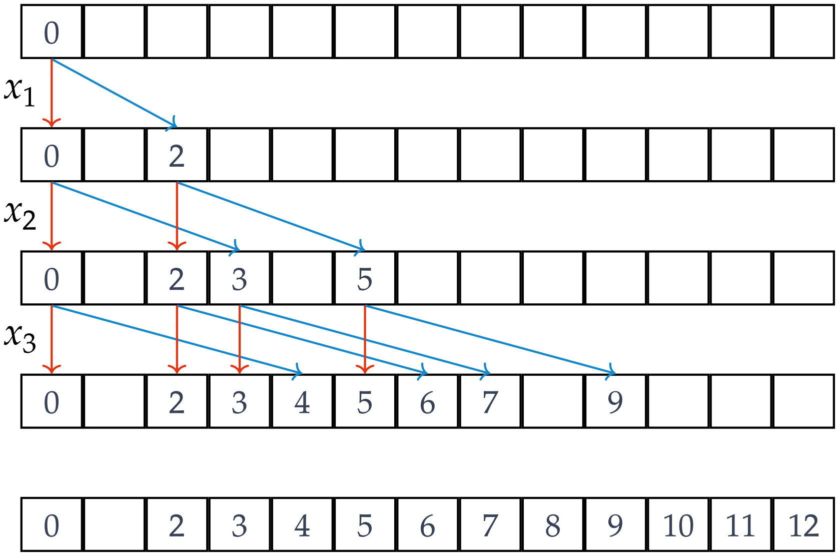
Example: $10 \le 2x_1 + 3x_2 + 4x_3 + 5x_4 \le 12$



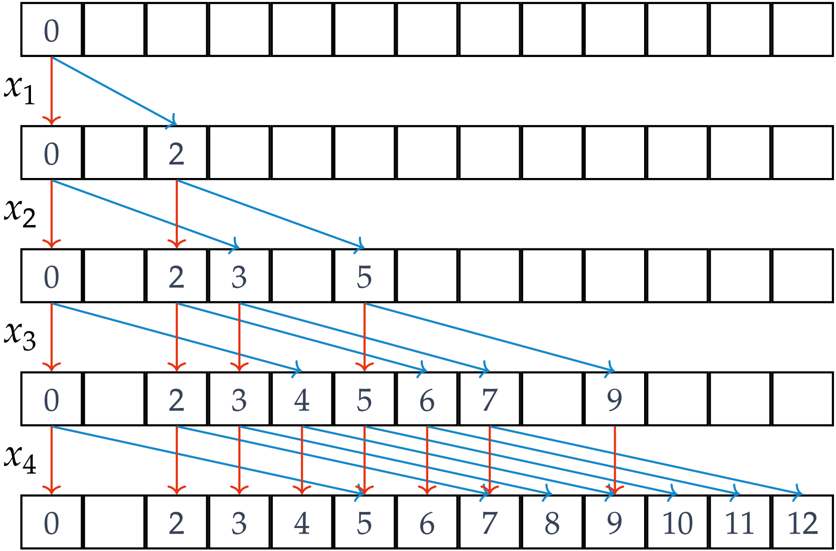
Example: $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



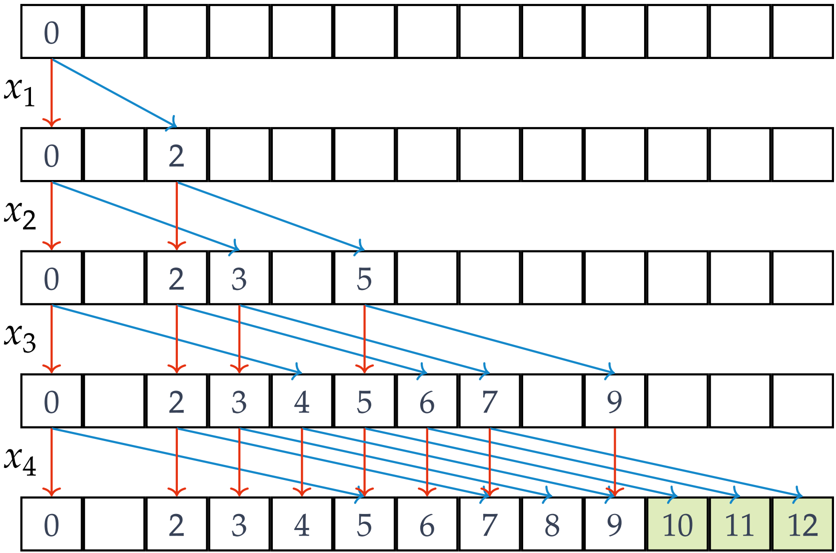
Example: $10 \le 2x_1 + 3x_2 + 4x_3 + 5x_4 \le 12$



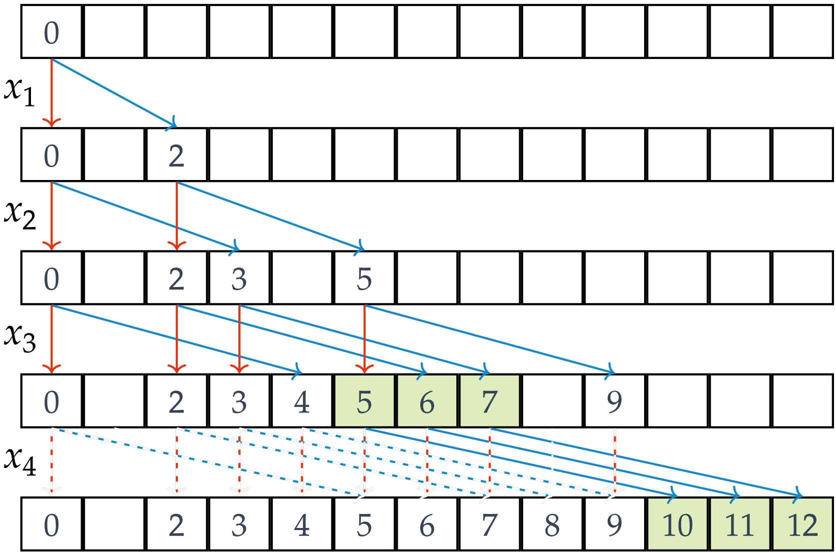
Example: $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



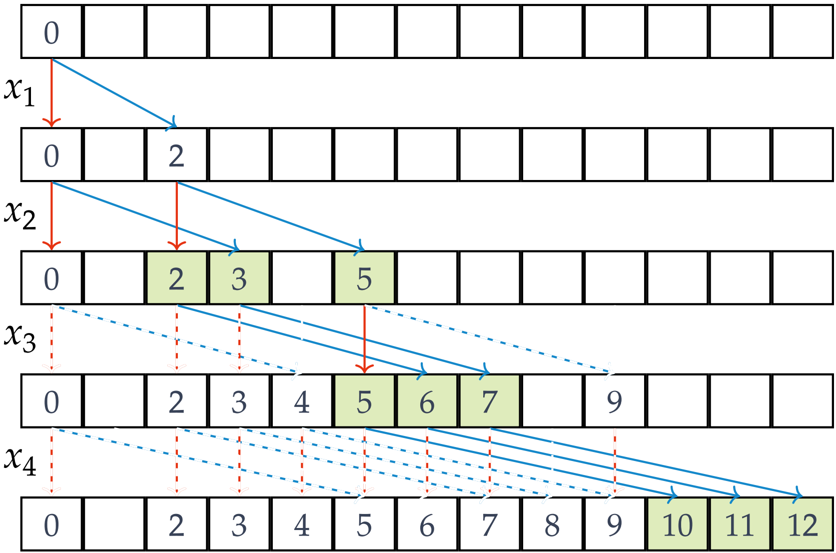
Example: $10 \le 2x_1 + 3x_2 + 4x_3 + 5x_4 \le 12$



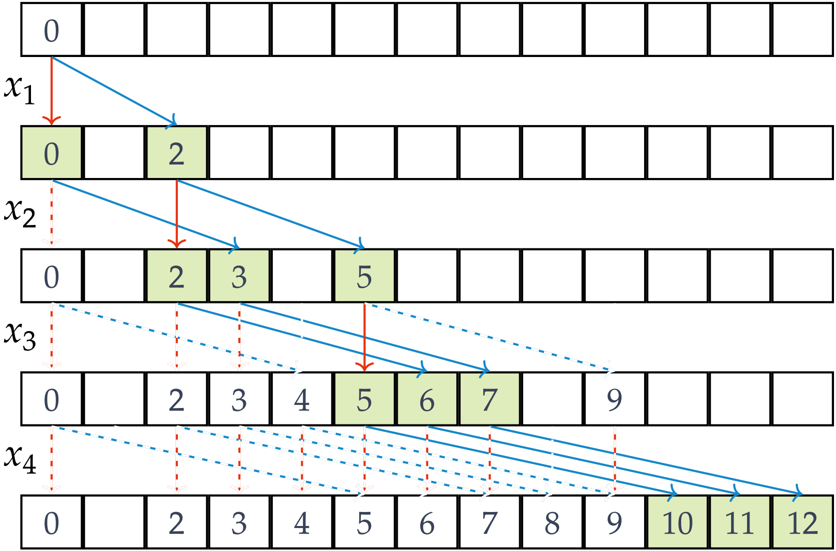
Example: $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



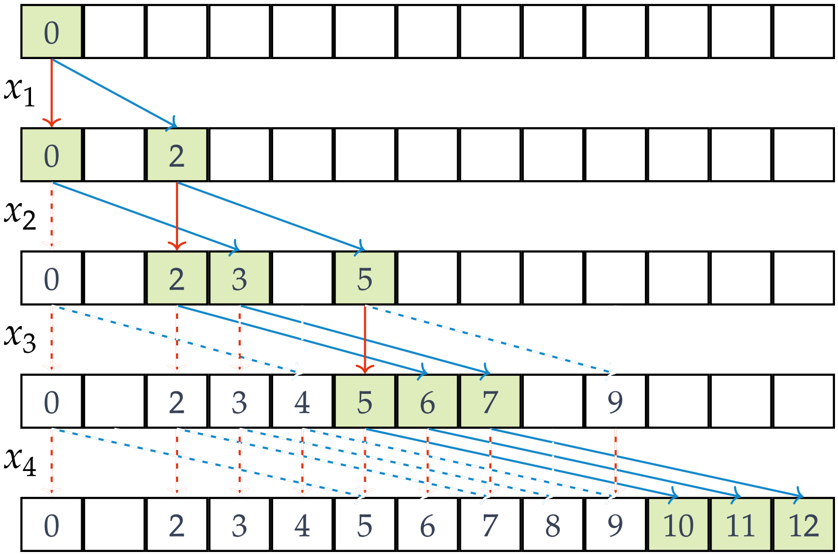
Example: $10 \le 2x_1 + 3x_2 + 4x_3 + 5x_4 \le 12$



Example: $10 \le 2x_1 + 3x_2 + 4x_3 + 5x_4 \le 12$



Example: $10 \leq 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 12$



Satisfiability $\Rightarrow$ filtering

In the previous example we obtain $x_4 = 1$ in all feasible solutions

this can be propagated and maintained incrementally:

squash one level of the graph, and if necessary shift right the levels above

We deduce an AC algorithm from one that computes efficiently (and if possible incrementally), not one, but **all feasible solutions**

Alldifferent

Global Constraint Catalog:

> *Denotes the fact that we have one or several cliques of disequalities.*

Example:

$$D(x_1) = \{1,2\} \quad D(x_2) = \{2,3\} \quad D(x_3) = \{1,3\}$$
$$D(x_4) = \{2,4\} \quad D(x_5) = \{3,4,5,6\} \quad D(x_6) = \{6,7\}$$

Maximum Matching

Wikipedia:

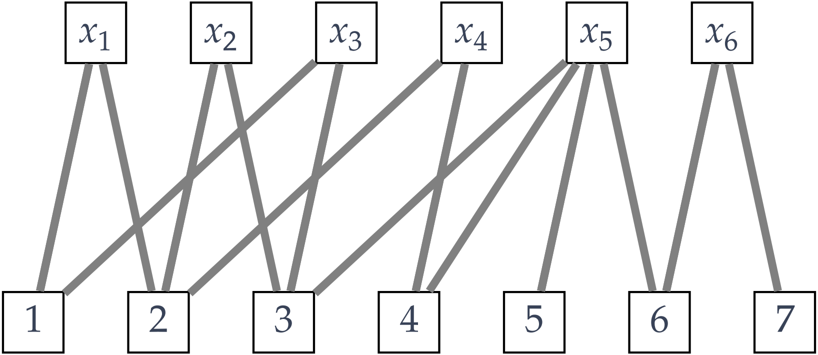
> *Given a graph $G = (V, E)$, a* matching M *in* G *is a set of pairwise non-adjacent edges*
> *[...]*
> *A* maximum matching *is a matching that contains the largest possible number of edges*

Now associate a vertex to each variable, one to each value and an edge between x and v if $v \in D(x)$

If the size of the matching is equal to the number of variables, it represents a solution to the `alldifferent` constraint

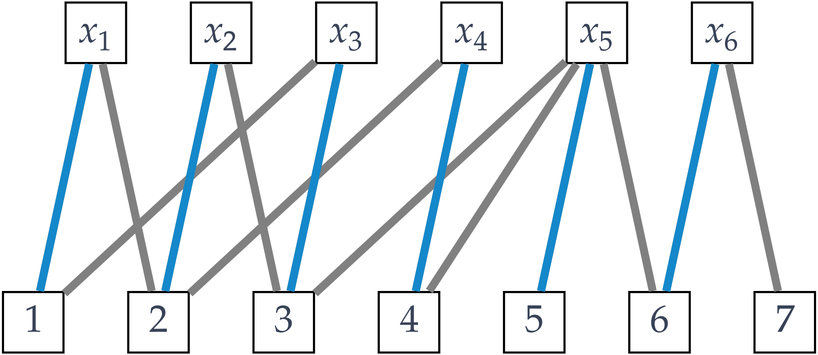
Graph constraint for Alldifferent

$D(x_1) = \{1,2\}$ $D(x_2) = \{2,3\}$ $D(x_3) = \{1,3\}$
$D(x_4) = \{2,4\}$ $D(x_5) = \{3,4,5,6\}$ $D(x_6) = \{6,7\}$



Graph constraint for Alldifferent

$D(x_1) = \{1,2\}$ $D(x_2) = \{2,3\}$ $D(x_3) = \{1,3\}$
$D(x_4) = \{2,4\}$ $D(x_5) = \{3,4,5,6\}$ $D(x_6) = \{6,7\}$



General GC filtering strategy

General GC filtering strategy

Find an efficient algorithm for checking satisfiability

General GC filtering strategy

Find an efficient algorithm for checking satisfiability

make it efficient for *all* solutions

General GC filtering strategy

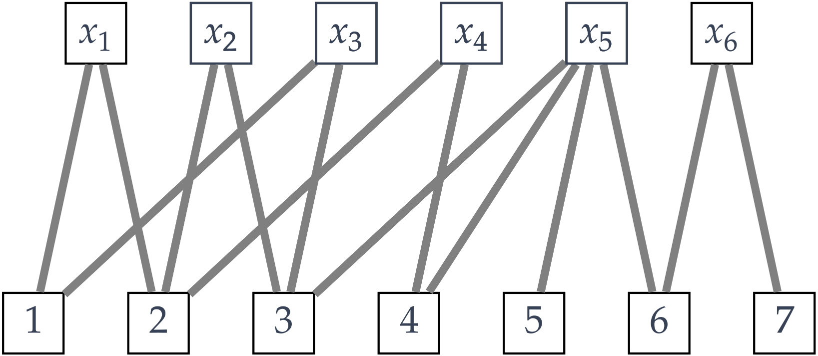
Find an efficient algorithm for checking satisfiability

make it efficient for *all* solutions

obtain arc-consistency/domain-consistency

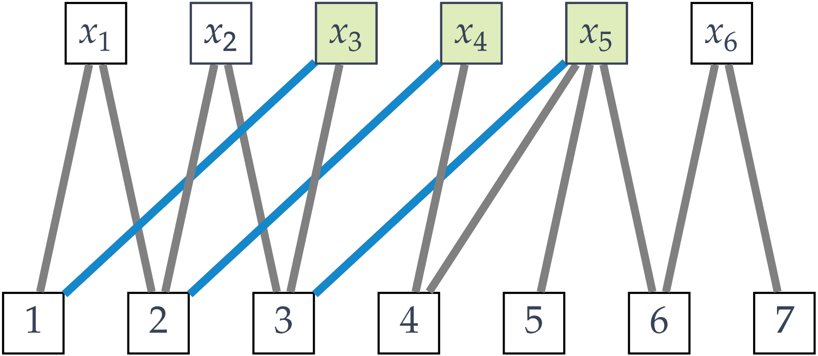
How to find a maximum matching?

Iteratively **improve** a matching



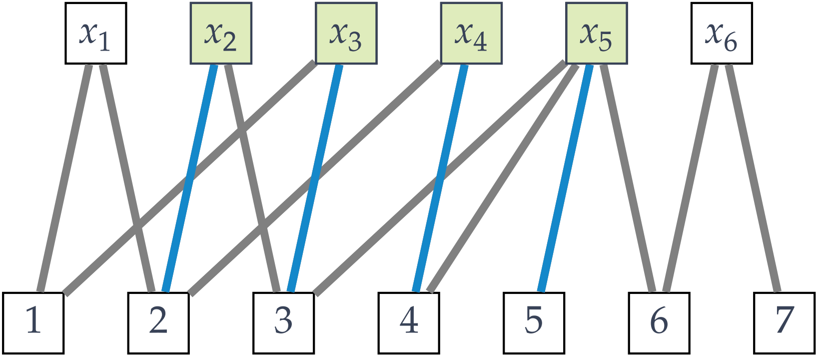
How to find a maximum matching?

Iteratively **improve** a matching



How to find a maximum matching?

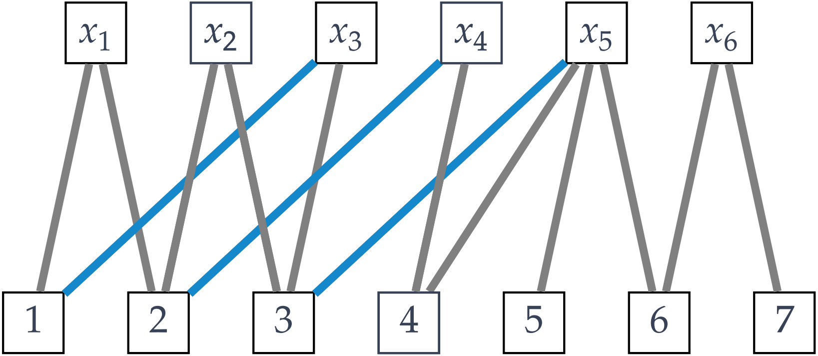
Iteratively **improve** a matching



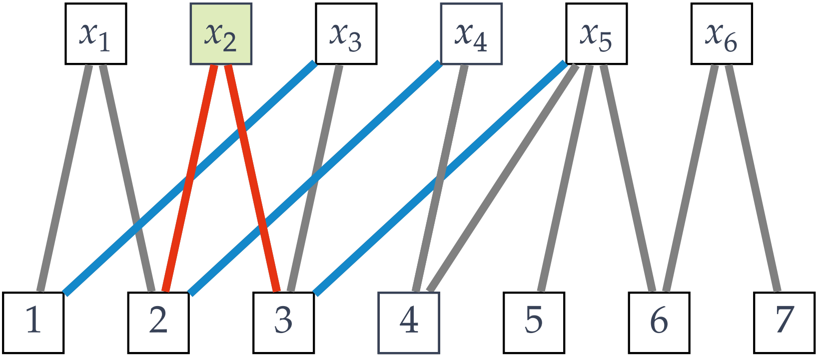
Improving a matching

1. find a free vertex x
2. if $\exists(x, v) \in G$ s. t. v is not matched, add it to M
3. otherwise
   1. take (x, v) such that $(y, v) \in M$
   2. restart at 1 but using y instead of x

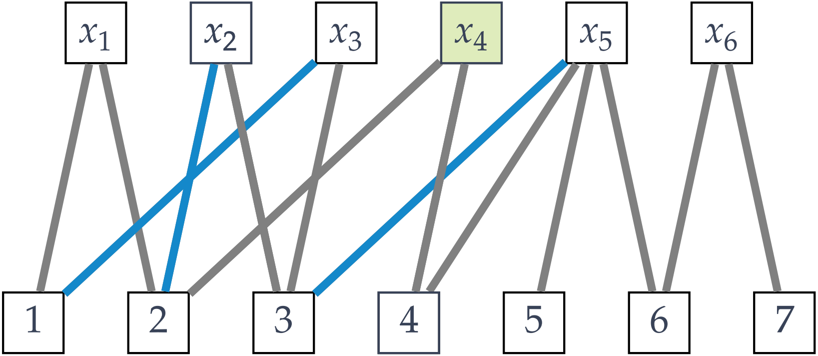
Improving a matching



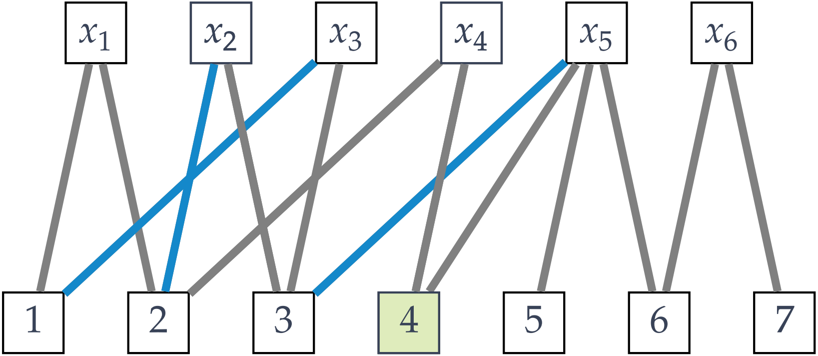
Improving a matching



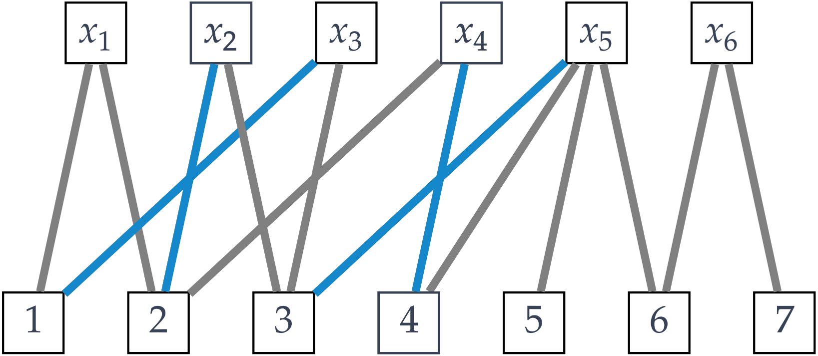
Improving a matching



Improving a matching



Improving a matching



Does it always work?

Does it always work?

No! It can

Does it always work?

No! It can loop...

Does it always work?

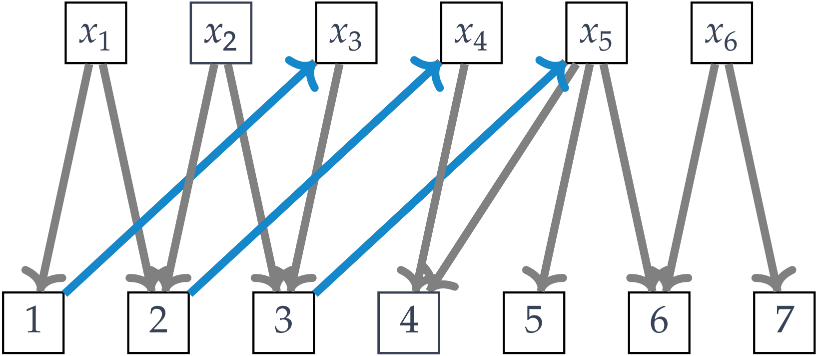
No! It can loop...

It works if we can find an *odd-length* **alternating path** (one edge in M one not in M), starting and ending in a *free vertex*

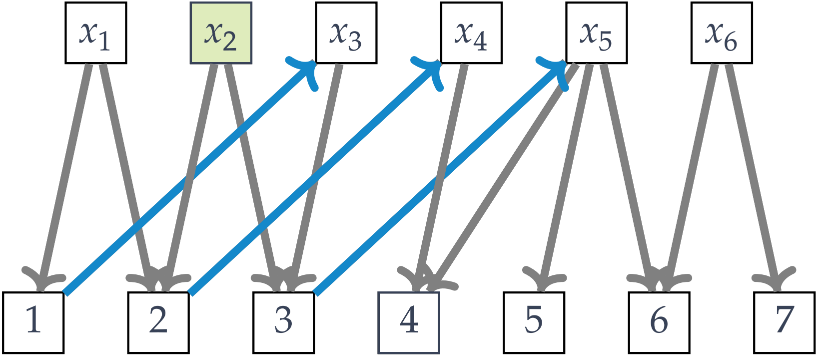
Enforce alternation by orienting G as follows:

- $(x, v) \in M$, orient it as $v \to x$
- $(x, v) \notin M$, orient it as $x \to v$

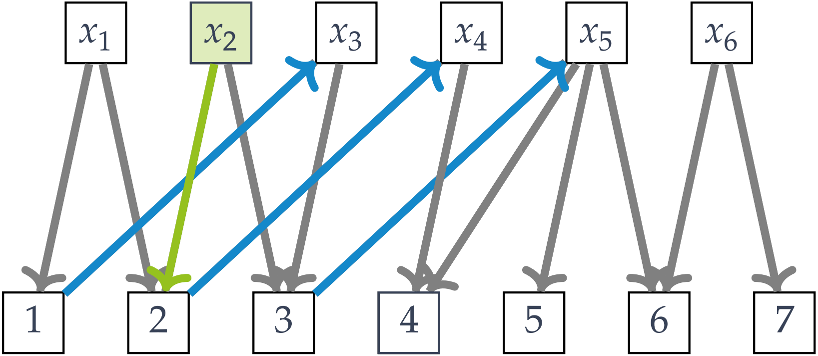
Alternating path



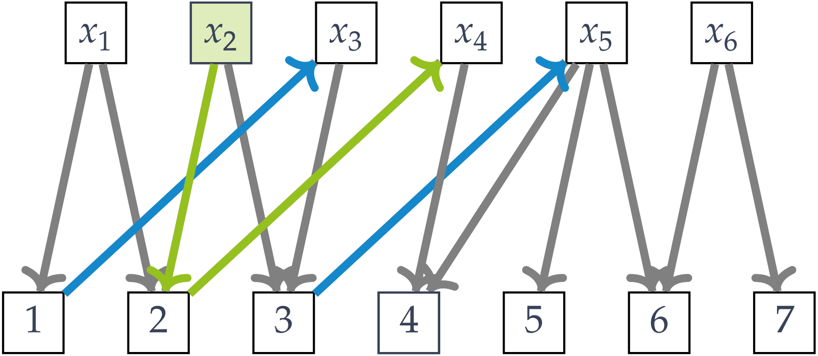
Alternating path



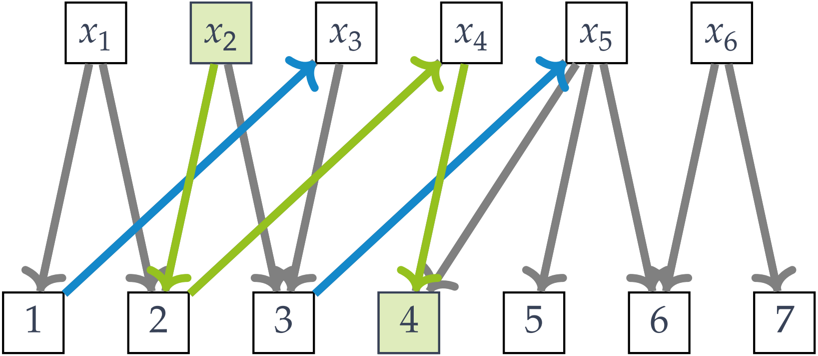
Alternating path



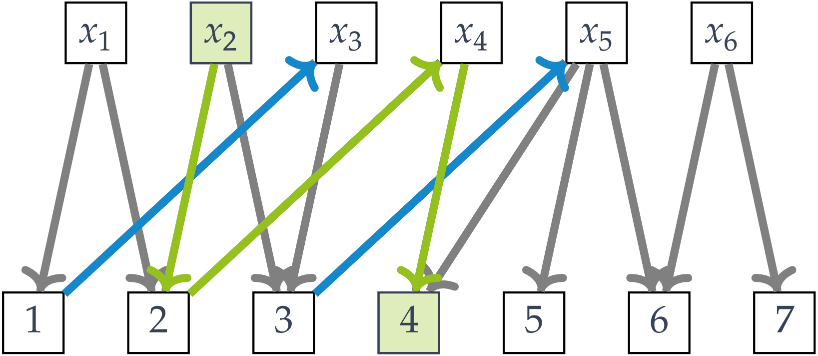
Alternating path



Alternating path

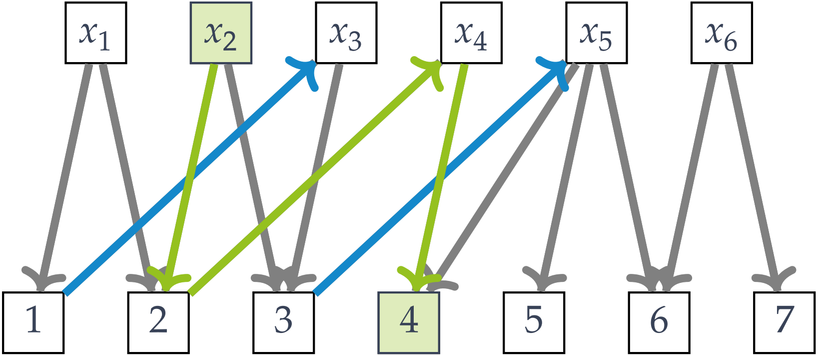


Alternating path



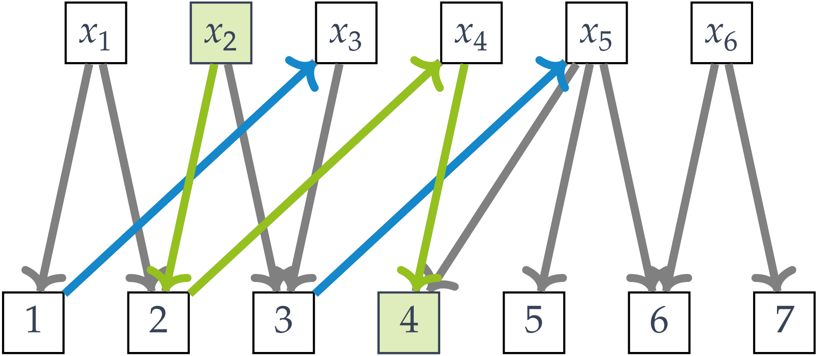
How do we find such a path?

Alternating path



How do we find such a path?
DFS (or any other similar algorithm), $O(|V| + |E|)$

Alternating path

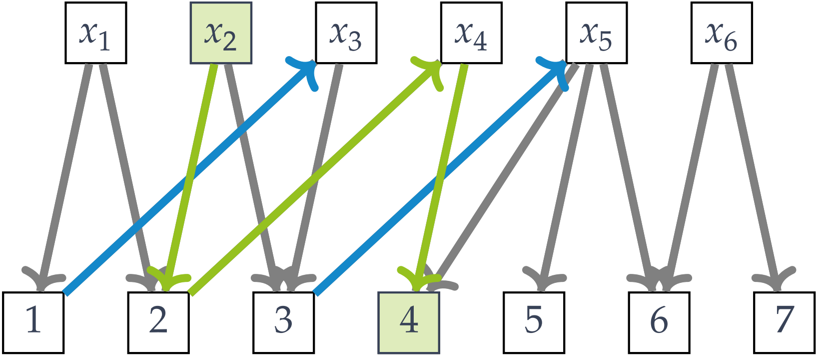


How do we find such a path?

DFS (or any other similar algorithm), $O(|V| + |E|)$

Update arrows and iterate

Alternating path



How do we find such a path?
DFS (or any other similar algorithm), $O(|V| + |E|)$
Update arrows and iterate
No path starting from free *variable* x_i means that x_i not in the maximum matching

Summary

To check for satisfiability of the `alldifferent` constraint

Summary

To check for satisfiability of the `alldifferent` constraint

build a graph G with $V = \{x_i\} \cup \{v_i\}$ and $E = \{(x, v) \mid v \in D(x)\}$

Summary

To check for satisfiability of the `alldifferent` constraint

build a graph G with $V = \{x_i\} \cup \{v_i\}$ and $E = \{(x, v) \mid v \in D(x)\}$

look for a **maximum matching** by iterative improvement using DFS for alternating paths in a directed version of G

Summary

To check for satisfiability of the `alldifferent` constraint

build a graph G with $V = \{x_i\} \cup \{v_i\}$ and $E = \{(x, v) \mid v \in D(x)\}$

look for a **maximum matching** by iterative improvement using DFS for alternating paths in a directed version of G

if $|M| = |X|$ we have satisfiability

and now?

Claude
Berge
1926–2002

Claude Berge

One of the co-founders of French literary group **Oulipo**

Great-grandson of French President Félix Faure

Two conjectures in the 60s on *perfect graphs* proven much later

Notions of acyclicity for **hypergraphs** (e.g. constraint hypergraphs!)

Berge's lemma about maximum matchings

Berge's lemma

An edge is considered *free* if it belongs to a maximum matching but does not belong to all maximum matchings.

An edge e is *free* if and only if, in an arbitrary maximum matching M, the edge e belongs to an **even alternating path starting at an unmatched vertex** or to an **alternating cycle**.

How do we use that?

Find a maximum matching M

How do we use that?

Find a maximum matching M

Start from a free **value**

How do we use that?

Find a maximum matching M

Start from a free **value**

look for a path π in G with the **opposite orientation** as before

How do we use that?

Find a maximum matching M

Start from a free **value**

look for a path π in G with the **opposite orientation** as before

any arc in π belongs to some maximum matching

How do we use that?

Find a maximum matching M

Start from a free **value**

look for a path π in G with the **opposite orientation** as before

any arc in π belongs to some maximum matching (i.e., solution)

How do we use that?

Find a maximum matching M

Start from a free **value**

look for a path π in G with the **opposite orientation** as before

any arc in π belongs to some maximum matching (i.e., solution)

SCCs are alternating cycles, arcs also belong to some solution

How do we use that?

Find a maximum matching M

Start from a free **value**

look for a path π in G with the **opposite orientation** as before

any arc in π belongs to some maximum matching (i.e., solution)

SCCs are alternating cycles, arcs also belong to some solution

filter edges

How do we use that?

Find a maximum matching M

Start from a free **value**

look for a path π in G with the **opposite orientation** as before

any arc in π belongs to some maximum matching (i.e., solution)

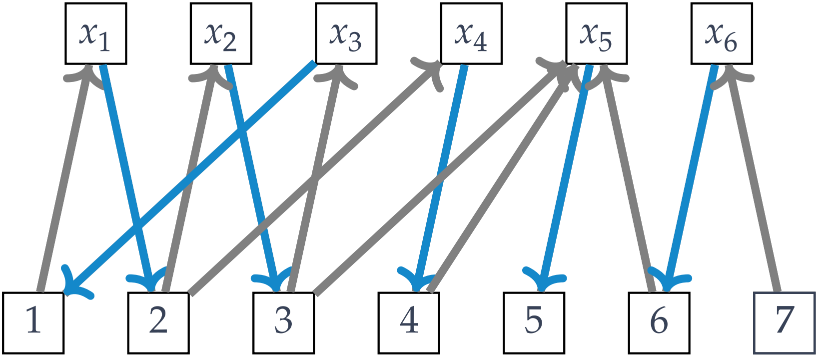
SCCs are alternating cycles, arcs also belong to some solution

filter edges **not in M, nor in π nor in an SCC**

Filtering

$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

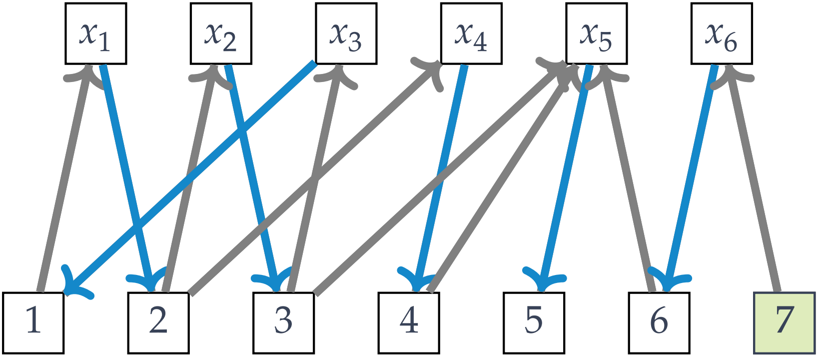
$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



Filtering

$D(x_1) = \{1,2\}$ $D(x_2) = \{2,3\}$ $D(x_3) = \{1,3\}$

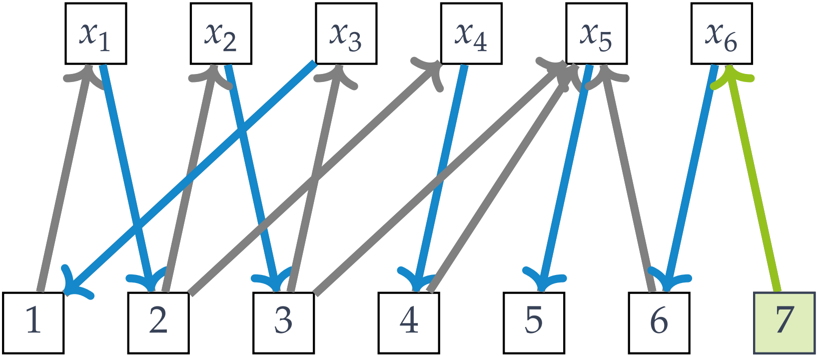
$D(x_4) = \{2,4\}$ $D(x_5) = \{3,4,5,6\}$ $D(x_6) = \{6,7\}$



Filtering

$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

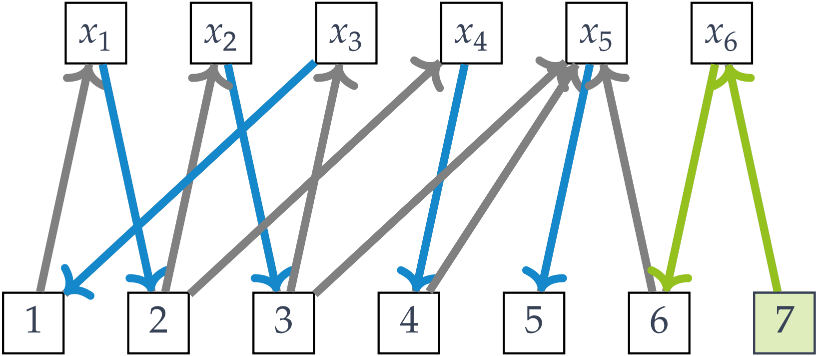
$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



Filtering

$D(x_1) = \{1,2\}$ $D(x_2) = \{2,3\}$ $D(x_3) = \{1,3\}$

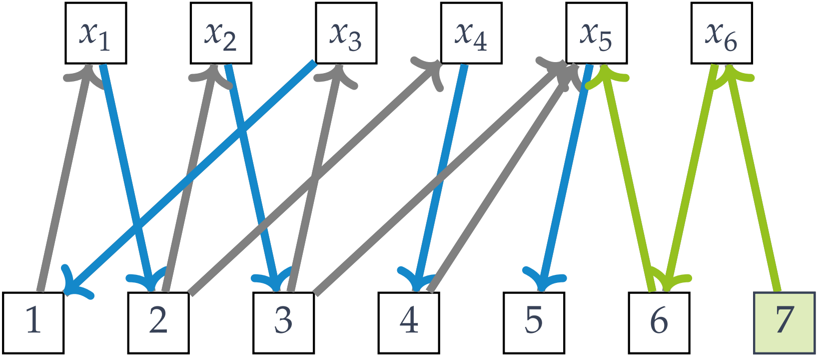
$D(x_4) = \{2,4\}$ $D(x_5) = \{3,4,5,6\}$ $D(x_6) = \{6,7\}$



Filtering

$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

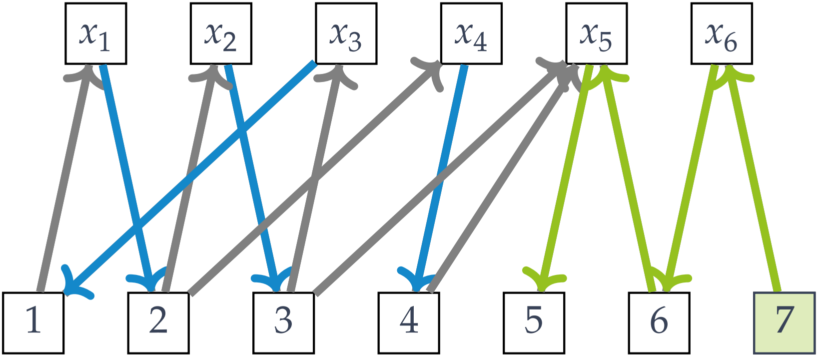
$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



Filtering

$D(x_1) = \{1,2\}$ $D(x_2) = \{2,3\}$ $D(x_3) = \{1,3\}$

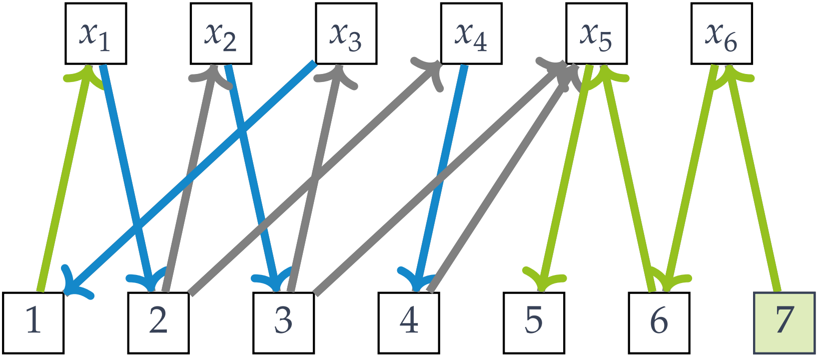
$D(x_4) = \{2,4\}$ $D(x_5) = \{3,4,5,6\}$ $D(x_6) = \{6,7\}$



Filtering

$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

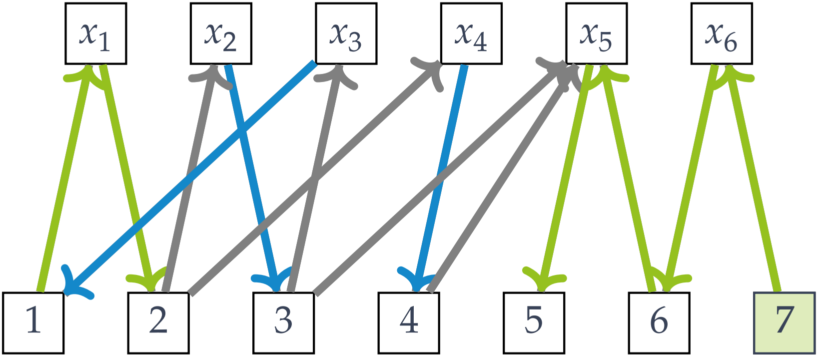
$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



Filtering

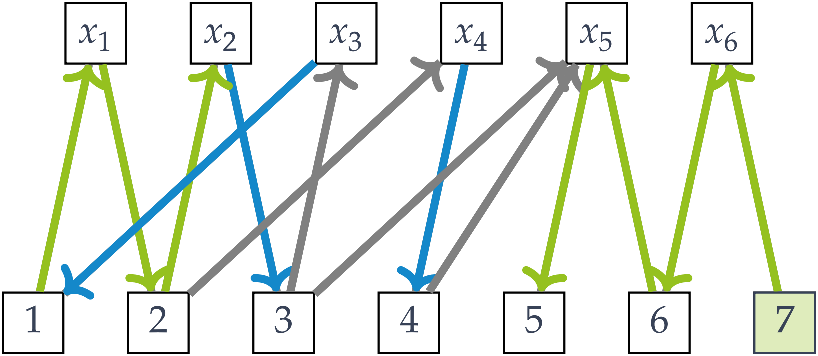
$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



Filtering

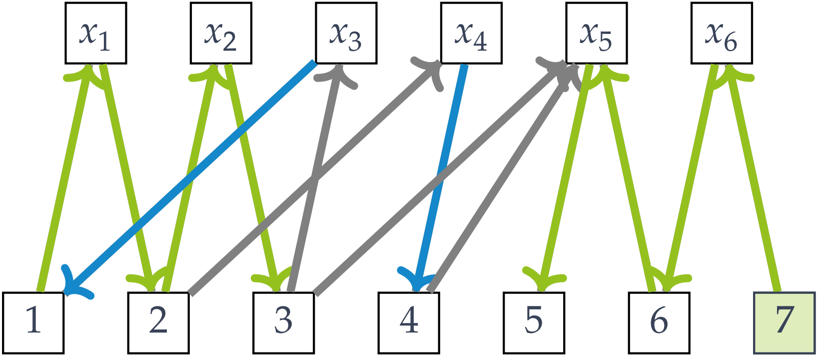
$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$
$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



Filtering

$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

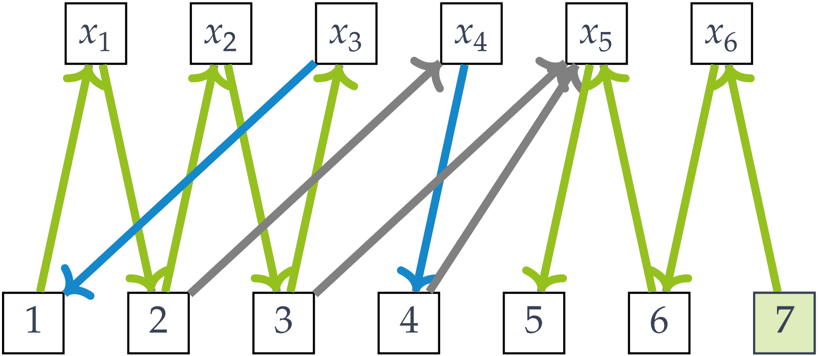
$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



Filtering

$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

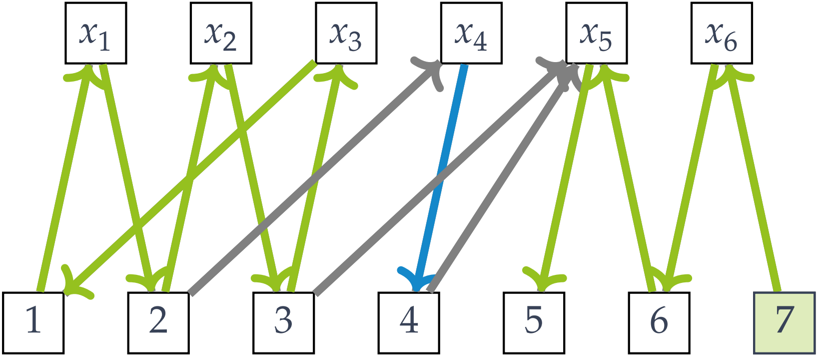
$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



Filtering

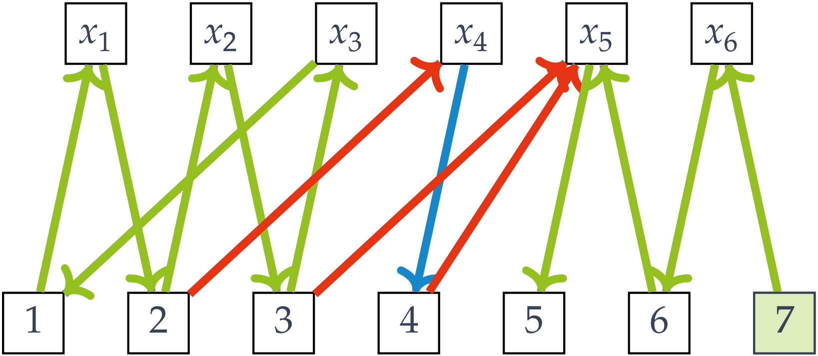
$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



Filtering

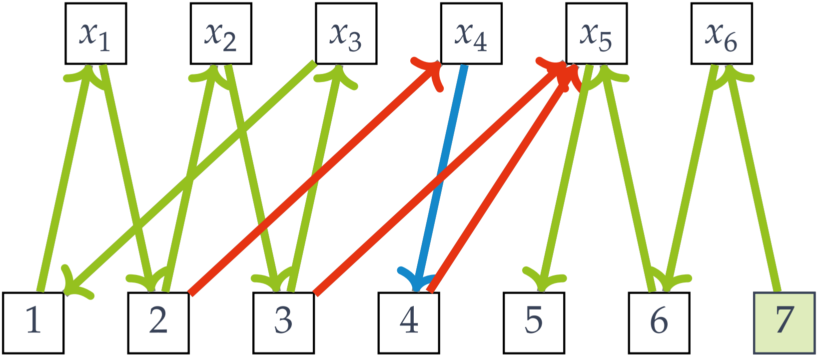
$$D(x_1) = \{1, 2\} \quad D(x_2) = \{2, 3\} \quad D(x_3) = \{1, 3\}$$
$$D(x_4) = \{2, 4\} \quad D(x_5) = \{3, 4, 5, 6\} \quad D(x_6) = \{6, 7\}$$



Filtering

$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

$D(x_4) = \{2, 4\}$ $D(x_5) = \{3, 4, 5, 6\}$ $D(x_6) = \{6, 7\}$



$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

$D(x_4) = \{4\}$ $D(x_5) = \{5, 6\}$ $D(x_6) = \{6, 7\}$

423 global constraints

Global Constraint Catalog
(`http://sofdem.github.io/gccat/`)

Created by Nicolas Beldiceanu (EMN) in 2006

aims at an exhaustive characterization of all global constraints, with their filtering algorithm(s)

as comparison, only about 100 global constraints in `globals.mzn`

Common global constraints

`alldifferent`, N-queens, Sudoku, any mutual exclusion, ...

`minimum`/`maximum`, imposes that the value of one variable is the min/max of those of other variables. Bidirectional!

`global_cardinality`, specifies the number of occurrences of each value in a list of variables
Can be used for magic-series, and derived in `atleast`, `atmost`, etc.
Flow algorithm for consistency

Common global constraints

`lex_chain`, imposes that vectors are lexicographically ordered

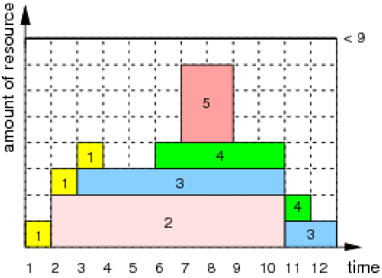
Most common usage will be given next week

Can be encoded with reified constraints

but has a dedicated filtering algorithm based on computing tight lower/upper bounds

Common global constraints

`cumulative`, limits the *capacity* of a machine handling several tasks (s_i, d_i) at any point in time



Filtering based on computing *compulsory parts*, but if all durations (and heights) are fixed ⇒ **balancing knapsack** constraint (i.e., dynamic programming)

Semantic decomposition

Same solutions, but simpler (binary, sometimes ternary) constraints
Typical example `alldifferent`

One can allow extra variables and project solutions
e.g., `exactly`$([x_1, \dots, x_n], k, v)$ can be decomposed using $n + 1$ extra variables $b_0, \dots, b_n \in \{0, \dots, n\}$, such that:

Semantic decomposition

Same solutions, but simpler (binary, sometimes ternary) constraints
Typical example `alldifferent`

One can allow extra variables and project solutions e.g., `exactly`$([x_1, \dots, x_n], k, v)$ can be decomposed using $n + 1$ extra variables $b_0, \dots, b_n \in \{0, \dots, n\}$, such that:

- $b_0 = 0$
- $(x_i = v \wedge b_i = b_{i-1} + 1) \vee (x_i \neq v \wedge b_i = b_{i-1})$
- $b_n = k$

AC-decomposition

Semantic decomposition is actually always feasible, and doesn't help with filtering...

We want the same solutions but also the same **level of propagation**

Not the case for

AC-decomposition

Semantic decomposition is actually always feasible, and doesn't help with filtering...

We want the same solutions but also the same **level of propagation**

Not the case for `alldifferent`
but true for

AC-decomposition

Semantic decomposition is actually always feasible, and doesn't help with filtering...

We want the same solutions but also the same **level of propagation**

Not the case for `alldifferent`
but true for `exactly`

Why?

AC-decomposition

Semantic decomposition is actually always feasible, and doesn't help with filtering...

We want the same solutions but also the same **level of propagation**

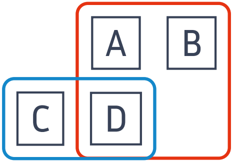
Not the case for `alldifferent`
but true for `exactly`

Why? ⇒ we need to call Claude Berge to the rescue again!

Hypergraphs and constraints

An *hypergraph* $H = (V, E)$ is a generalization of a graph where the edges E are arbitrary subsets of the vertices V

The *incidence graph* of H is the bipartite graph with vertices $V \uplus E$ and edges: $\{(v, e) \mid v \in e \text{ in } H\}$

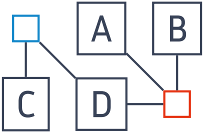


an n-ary constraint can be seen as an hyperedge in the hypergraph of constraints

Hypergraphs and constraints

An *hypergraph* $H = (V, E)$ is a generalization of a graph where the edges E are arbitrary subsets of the vertices V

The *incidence graph* of H is the bipartite graph with vertices $V \uplus E$ and edges: $\{(v, e) \mid v \in e \text{ in } H\}$



an n-ary constraint can be seen as an hyperedge in the hypergraph of constraints

Berge acyclicity

An hypergraph is **Berge-acyclic** iff its *incidence graph* is acyclic

Very strict: no hyperedge should intersect any other hyperedge with cardinal > 1

Theorem: If the decomposition of a constraint is Berge-acyclic, AC on the decomposition is equivalent to AC on the original constraint

Berge acyclicity

An hypergraph is **Berge-acyclic** iff its *incidence graph* is acyclic

Very strict: no hyperedge should intersect any other hyperedge with cardinal > 1

Theorem: If the decomposition of a constraint is Berge-acyclic, AC on the decomposition is equivalent to AC on the original constraint

Sketch of proof: the choice of values for the satisfiability of the decomposed constraints is *compatible*

Even better...

Actually, if the decomposition is **Berge-acyclic** it is enough to propagate twice each of the decomposed constraints

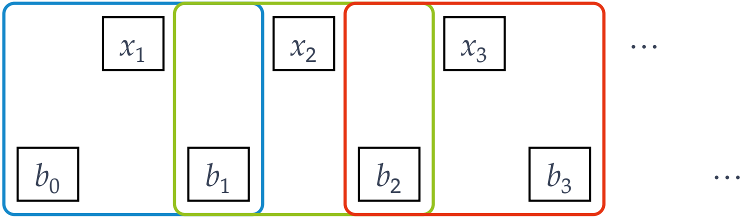
Even better...

Actually, if the decomposition is **Berge-acyclic** it is enough to propagate twice each of the decomposed constraints

The constraint hypergraph is a tree/forest (acyclic): propagate from leaves and contract, until you reach a single constraint, then propagate in the opposite order

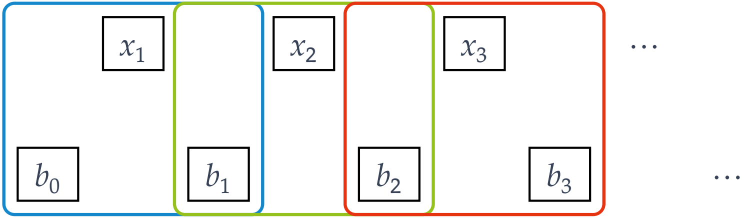
Example: `exactly`(x, v, k)

$$(x_i = v \wedge b_i = b_{i-1} + 1) \vee (x_i \neq v \wedge b_i = b_{i-1})$$



Example: `exactly`(x, v, k)

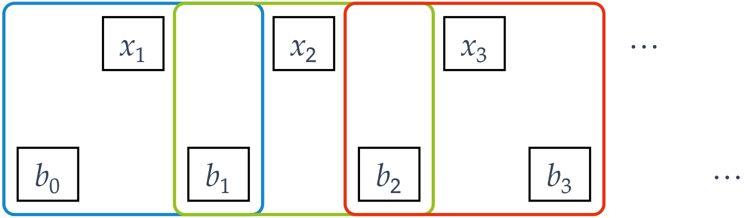
$$(x_i = v \land b_i = b_{i-1} + 1) \lor (x_i \neq v \land b_i = b_{i-1})$$



Propagate from one end or the other

Example: `exactly`(x, v, k)

$$(x_i = v \land b_i = b_{i-1} + 1) \lor (x_i \neq v \land b_i = b_{i-1})$$

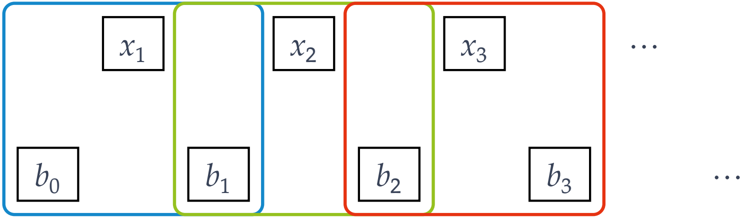


Propagate from one end or the other

What if we had decomposed `exactly` using **reified constraints**?

Example: `exactly`(x, v, k)

$$(x_i = v \wedge b_i = b_{i-1} + 1) \vee (x_i \neq v \wedge b_i = b_{i-1})$$

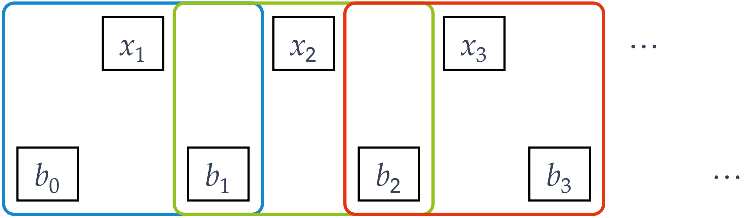


Propagate from one end or the other

What if we had decomposed `exactly` using **reified constraints**? $\bigwedge (b_i \Leftrightarrow x_i = v)$

Example: `exactly`(x, v, k)

$$(x_i = v \wedge b_i = b_{i-1} + 1) \vee (x_i \neq v \wedge b_i = b_{i-1})$$



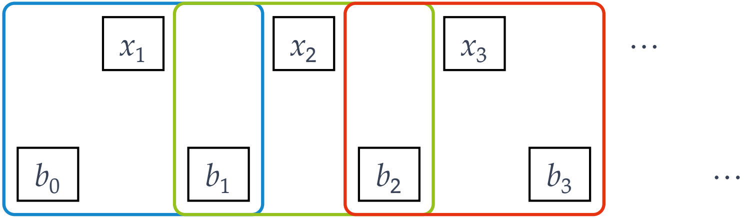
Propagate from one end or the other

What if we had decomposed `exactly` using **reified constraints**?

$$\bigwedge (b_i \Leftrightarrow x_i = v) \wedge \sum_{b_i} = k$$

Example: `exactly`(x, v, k)

$$(x_i = v \land b_i = b_{i-1} + 1) \lor (x_i \neq v \land b_i = b_{i-1})$$



Propagate from one end or the other

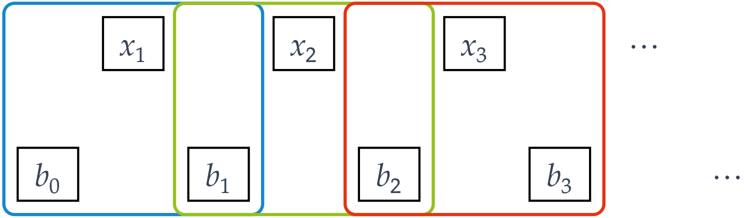
What if we had decomposed `exactly` using **reified constraints**?

$$\bigwedge (b_i \Leftrightarrow x_i = v) \land \sum_{b_i} = k$$

Still *Berge-acyclic*

Example: `exactly`(x, v, k)

$$(x_i = v \wedge b_i = b_{i-1} + 1) \vee (x_i \neq v \wedge b_i = b_{i-1})$$



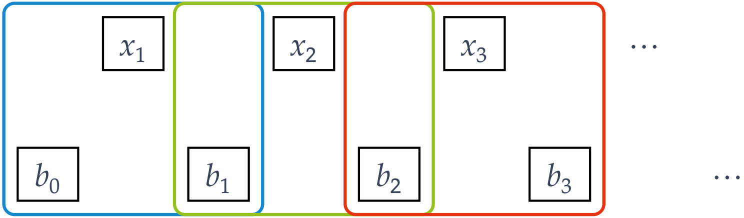
Propagate from one end or the other

What if we had decomposed `exactly` using **reified constraints**? $\bigwedge(b_i \Leftrightarrow x_i = v) \wedge \sum_{b_i} = k$

Still *Berge-acyclic* (star shape)

Example: $\texttt{exactly}(x, v, k)$

$$(x_i = v \wedge b_i = b_{i-1} + 1) \vee (x_i \neq v \wedge b_i = b_{i-1})$$



Propagate from one end or the other

What if we had decomposed `exactly` using **reified constraints**?

$$\bigwedge (b_i \Leftrightarrow x_i = v) \wedge \sum_{b_i} = k$$

Still *Berge-acyclic* (star shape) but `sum`!

“With great power comes great responsibility”

“With great power comes great responsibility”

(French National Convention, May 8th 1793 « une grande responsabilité est la suite inséparable d’un grand pouvoir »)

Global constraints are n-ary constraints, with **dedicated algorithms for efficient propagation**

they can sometimes be decomposed, but the cost might not be negligible (loss of filtering, additional variables, ...)

When they are available, they are very powerful, so, use them!

Docker setup

You can start

```
docker pull \
registry.gitlab.inria.fr/soliman/inf555/td6
```

now