THE CONSTRAINT PROGRAMMER'S TOOLBOX

Christian Schulte, SCALE, KTH & SICS

Constraint Programming

- What is constraint programming?

Sudoku is constraint programming

Sudoku

...is constraint programming!

Sudoku

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

- Assign blank fields digits such that:
  digits distinct per **rows**, columns, blocks

Sudoku

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

- Assign blank fields digits such that:
  digits distinct per rows, **columns**, blocks

Sudoku

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

- Assign blank fields digits such that:
  digits distinct per rows, columns, **blocks**

Block Propagation

	8	
	6	3

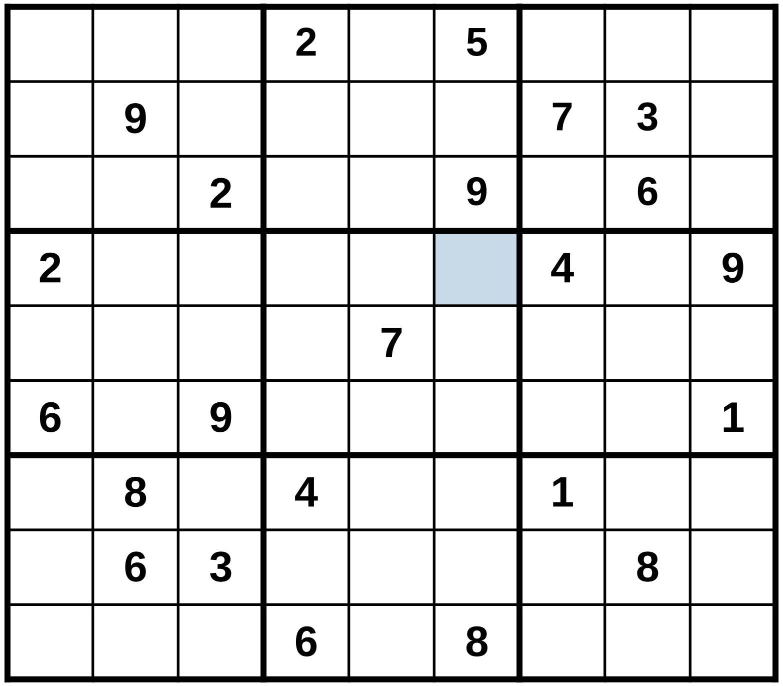
- No field in block can take digits 3,6,8

Block Propagation

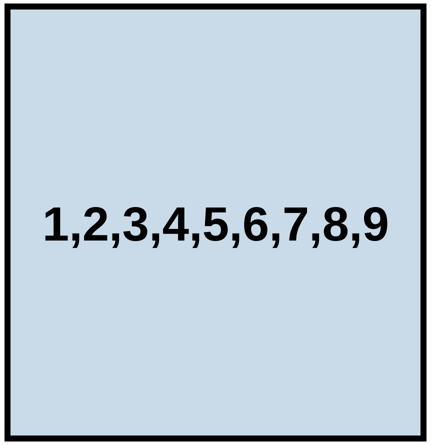
1,2,4,5,7,9	8	1,2,4,5,7,9
1,2,4,5,7,9	6	3
1,2,4,5,7,9	1,2,4,5,7,9	1,2,4,5,7,9

- No field in block can take digits 3,6,8
  - propagate to other fields in block
- Rows and columns: likewise

Propagation

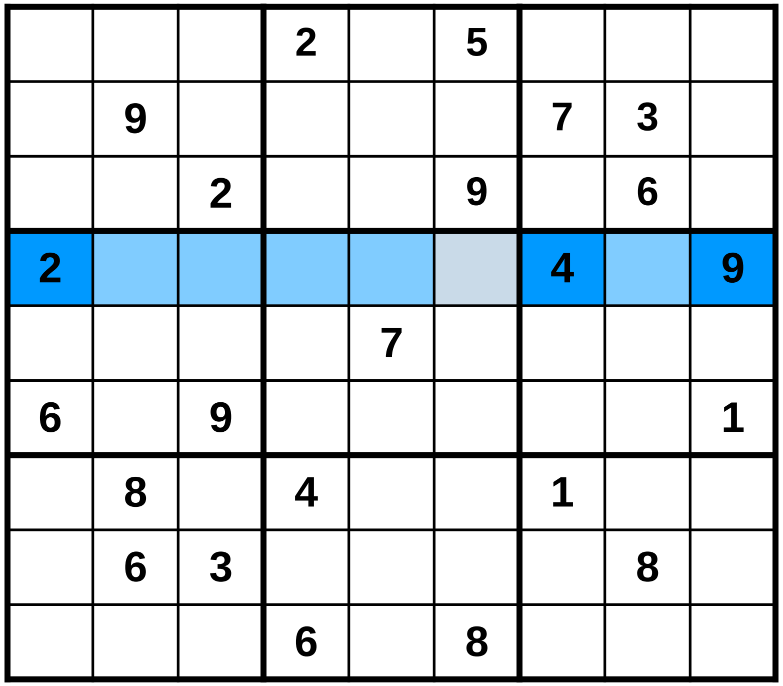


			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

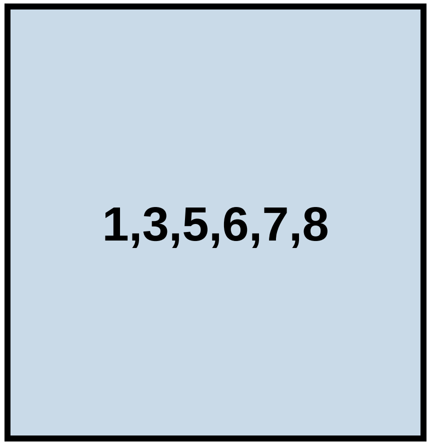


- Prune digits from fields such that:
  digits distinct per rows, columns, blocks

Propagation

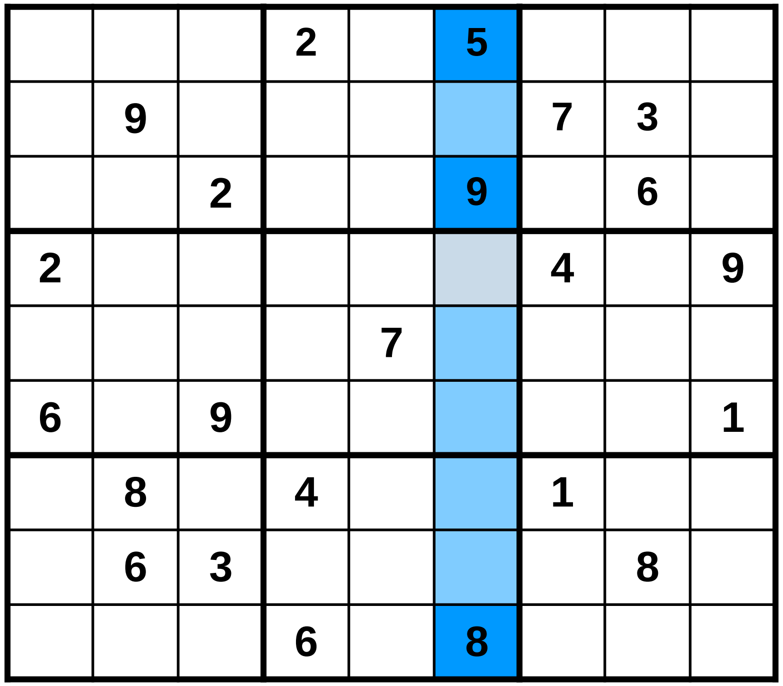


			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

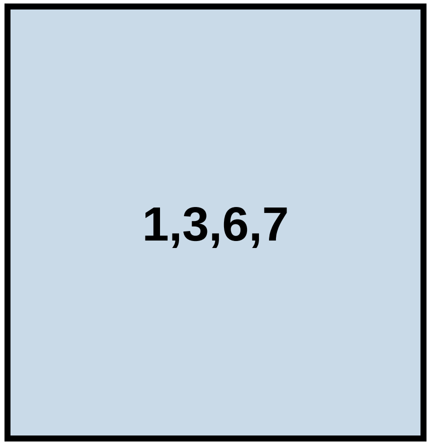


- Prune digits from fields such that:
  digits distinct per **rows**, columns, blocks

Propagation

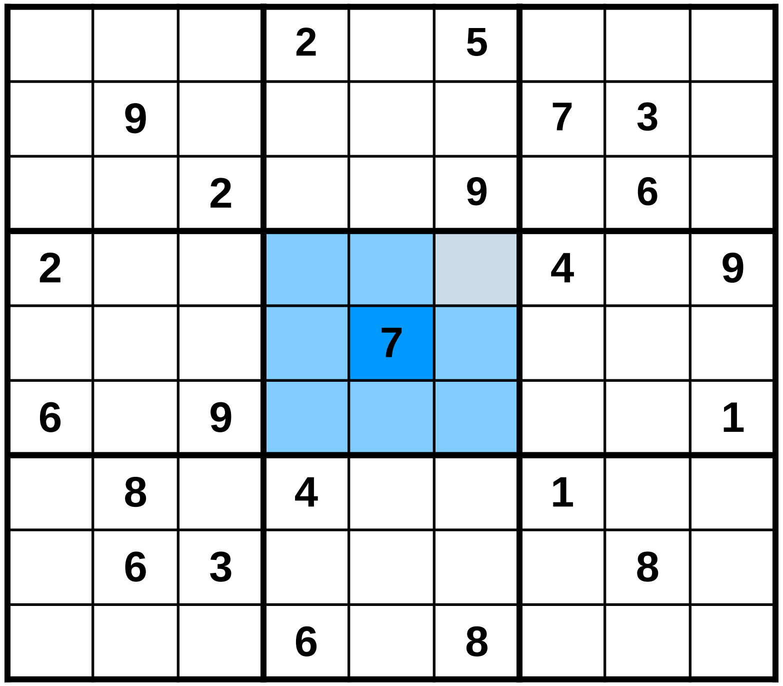


			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

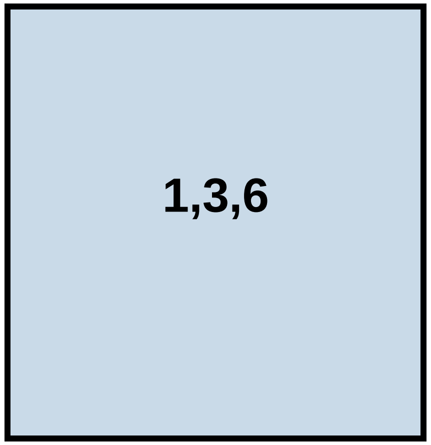


- Prune digits from fields such that:
  digits distinct per rows, **columns**, blocks

Propagation



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



- Prune digits from fields such that:
  digits distinct per rows, columns, **blocks**

Iterated Propagation

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

- Iterate propagation for rows, columns, blocks
- What if no assignment: search... later

Sudoku is Constraint Programming

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

- Variables: fields
  - take values: digits
  - maintain set of possible values
- Constraints: distinct
  - relation among values for variables

- Modeling: variables, values, constraints
- Solving: propagation, search

Constraint Programming

- Variable domains
    - finite domain integer, finite sets, multisets, intervals, ...
- Constraints
    - distinct, arithmetic, scheduling, graphs, ...
- Solving
    - propagation, search, ...
- Modeling
    - variables, values, constraints, heuristics, symmetries, ...

This Talk...

- Key concepts
  - **constraint propagation**
  - **search**
- The Constraint Programmer’s Toolbox..
- Some few tools

global constraints:	distinct reconsidered
branching heuristics:	bin packing
user-defined constraints:	personnel rostering

- Summary
  - essence of constraint programming and (very few) resources

Key Concepts

Running Example: SMM

- Find distinct digits for letters such that

```
     SEND
+    MORE
---------
=   MONEY
```

Constraint Model for SMM

- Variables:

  $S,E,N,D,M,O,R,Y \in \{0,\dots,9\}$

- Constraints:

  ```
  distinct(S,E,N,D,M,O,R,Y)
  ```

$$\begin{array}{rr} & 1000\times S+100\times E+10\times N+D \\ + & 1000\times M+100\times O+10\times R+E \\ = & 10000\times M+1000\times O+100\times N+10\times E+Y \end{array}$$

$S\neq 0 \qquad M\neq 0$

Solving SMM

- Find values for variables

  such that

  all constraints satisfied

Finding a Solution

- Compute with possible values
  - rather than enumerating assignments
- Prune inconsistent values
  - constraint propagation
- Search
  - branch: define shape of search tree
  - explore: explore search tree for solution

Constraint Propagation

- constraint store
- propagators
- constraint propagation

Constraint Store

$x \in \{1,2,3,4\}$ $y \in \{1,2,3,4\}$ $z \in \{1,2,3,4\}$

- Maps variables to possible values

Constraint Store

finite domain constraints

$x \in \{1,2,3,4\}$ $y \in \{1,2,3,4\}$ $z \in \{1,2,3,4\}$

- Maps variables to possible values
  - other domains: finite sets, float intervals, graphs, ...

Propagators

- Implement constraints

  distinct($x_1, \ldots, x_n$)

  $x + 2 \times y = z$

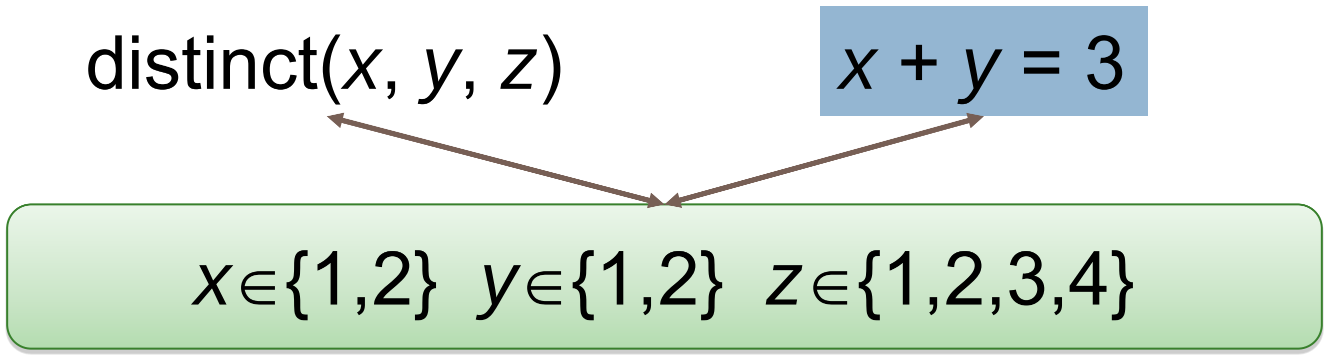
  schedule($t_1, \ldots, t_n$)

Propagators

$$\begin{tikzcd} \text{distinct}(x, y, z) \arrow[dr, leftrightarrow] & & x + y = 3 \arrow[dl, leftrightarrow] \\ & \boxed{x \in \{1,2,3,4\} \quad y \in \{1,2,3,4\} \quad z \in \{1,2,3,4\}} & \end{tikzcd}$$

- Strengthen store by constraint propagation
  - prune values in conflict with implemented constraint

Propagators



- Strengthen store by constraint propagation
  - prune values in conflict with implemented constraint

Propagators

$$\begin{tikzcd}
\text{distinct}(x, y, z) \arrow[dr, leftrightarrow] & & x + y = 3 \arrow[dl, leftrightarrow] \\
& x \in \{1,2\} \;\; y \in \{1,2\} \;\; z \in \{3,4\} &
\end{tikzcd}$$

- Iterate propagator execution until fixpoint
  - no more pruning possible

Propagation for SMM

- Results in store

```
S∈{9}  E∈{4,…,7}  N∈{5,…,8} D∈{2,…,8}
M∈{1}  O∈{0}      R∈{2,…,8} Y∈{2,…,8}
```

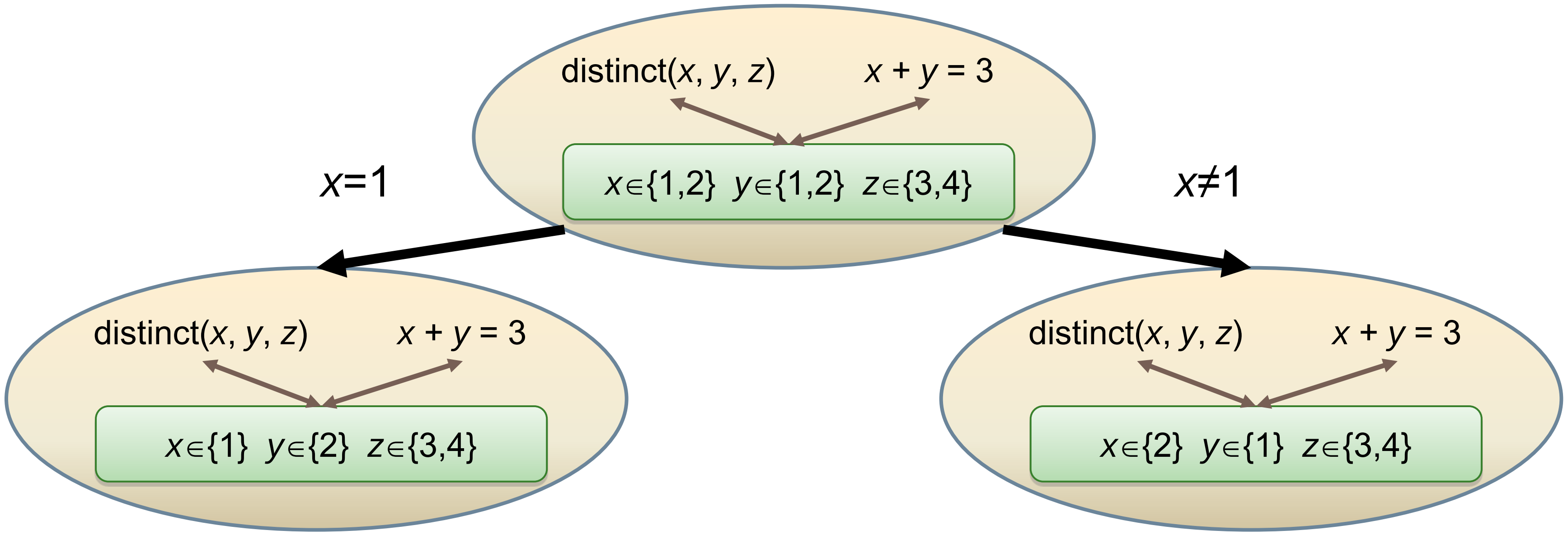
- Propagation **alone** not sufficient!
  - decompose into simpler sub-problems
  - branching and exploration for search

Search

branching

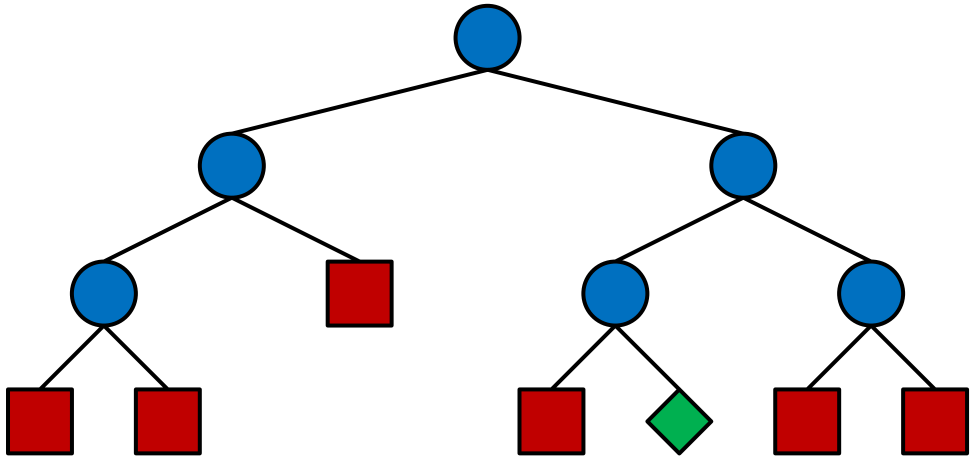
exploration

Branching


distinct(x, y, z)
$x + y = 3$
$x \in \{1,2\}$ $y \in \{1,2\}$ $z \in \{3,4\}$
$x=1$
$x \neq 1$
distinct(x, y, z)
$x + y = 3$
$x \in \{1\}$ $y \in \{2\}$ $z \in \{3,4\}$
distinct(x, y, z)
$x + y = 3$
$x \in \{2\}$ $y \in \{1\}$ $z \in \{3,4\}$

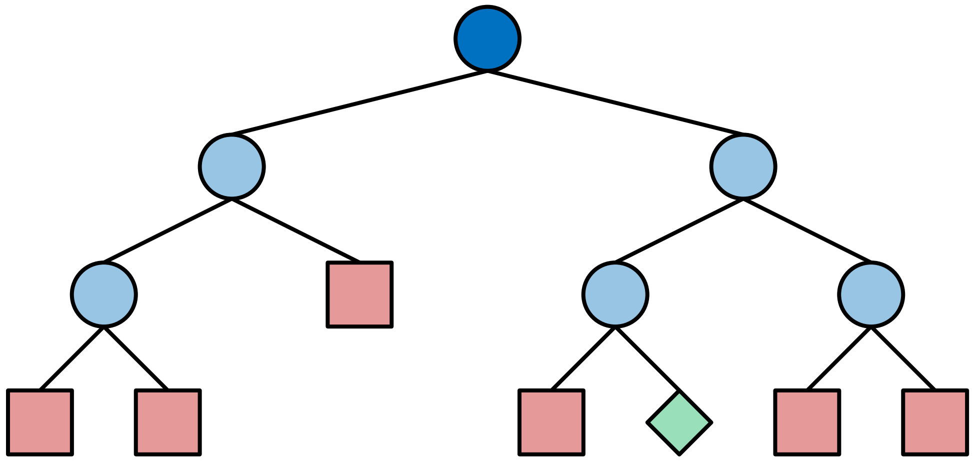
- Create subproblems with additional constraints
  - enables further propagation
  - defines search tree

Heuristic Branching



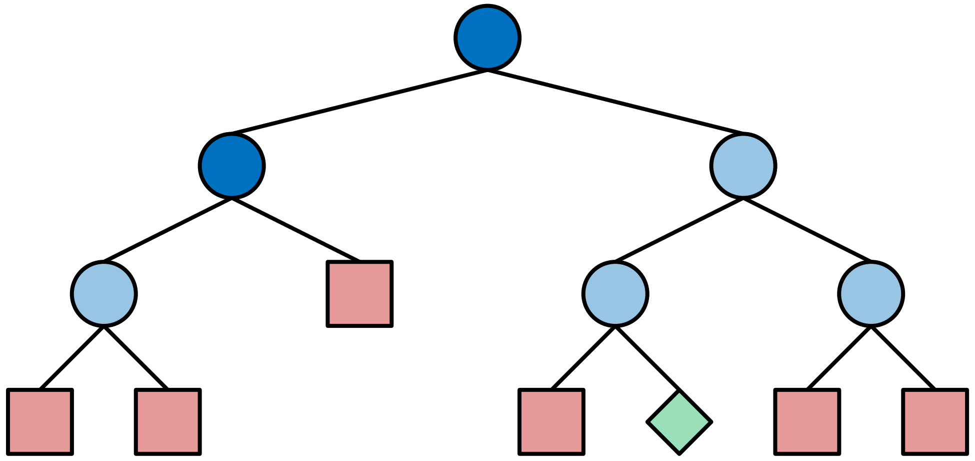
- Example branching
  - pick variable *x* (at least two values)
  - pick value *n* (from domain of x)
  - branch with $x = n$ and $x \neq n$
- Heuristic needed
  - which variable to select?
  - which value to select?

Search: Heuristic ⇆ Exploration



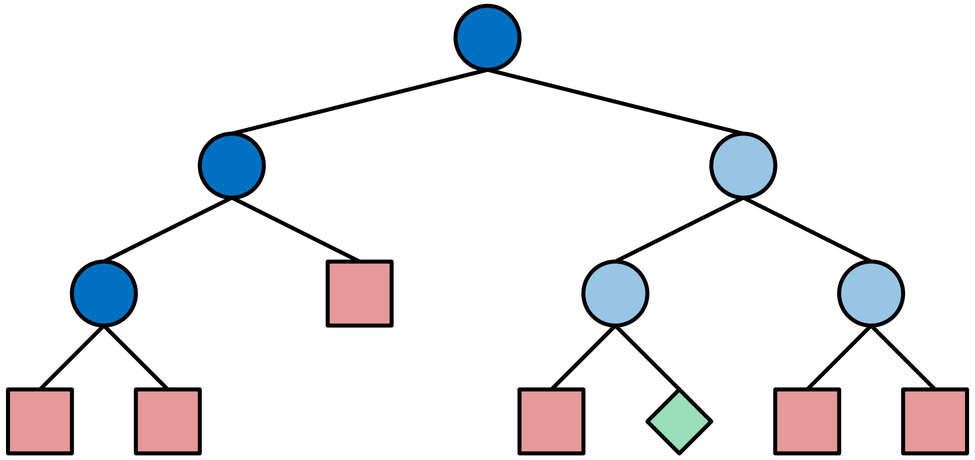
- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



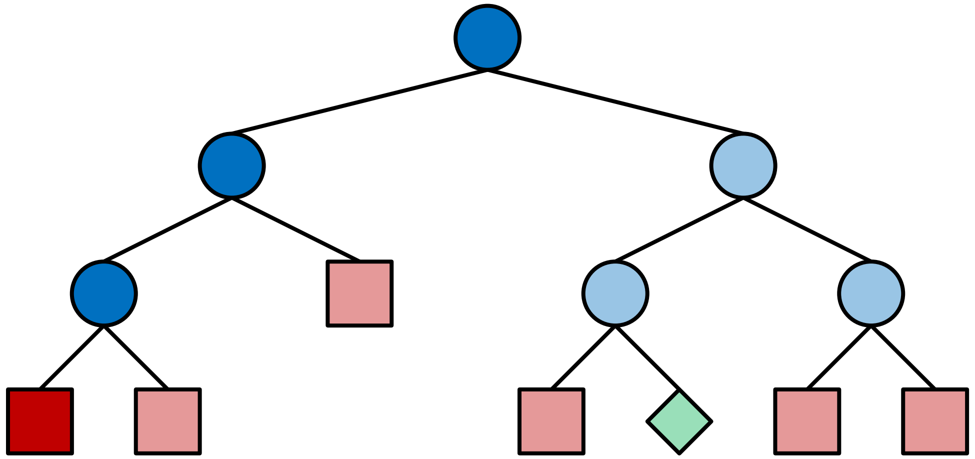
- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



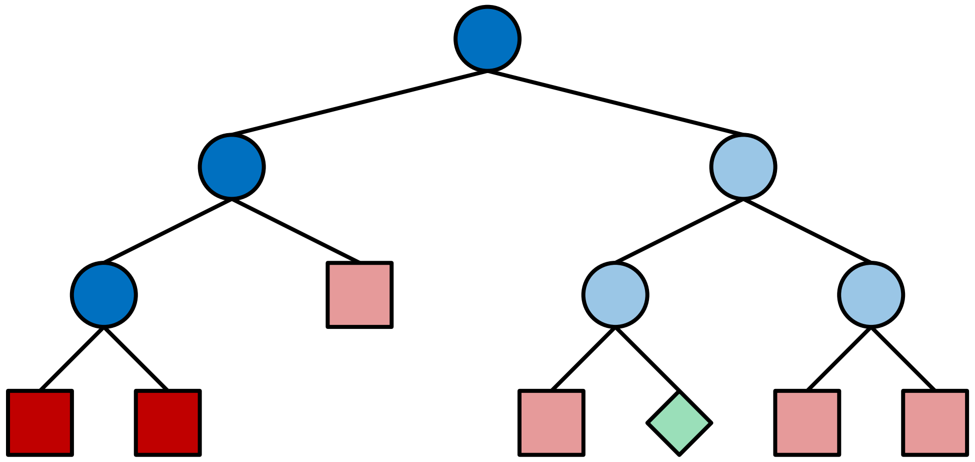
- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



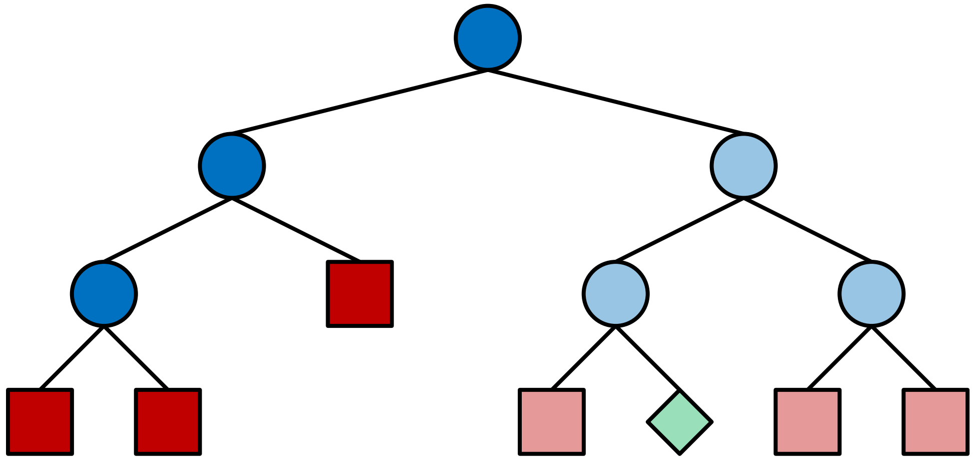
- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



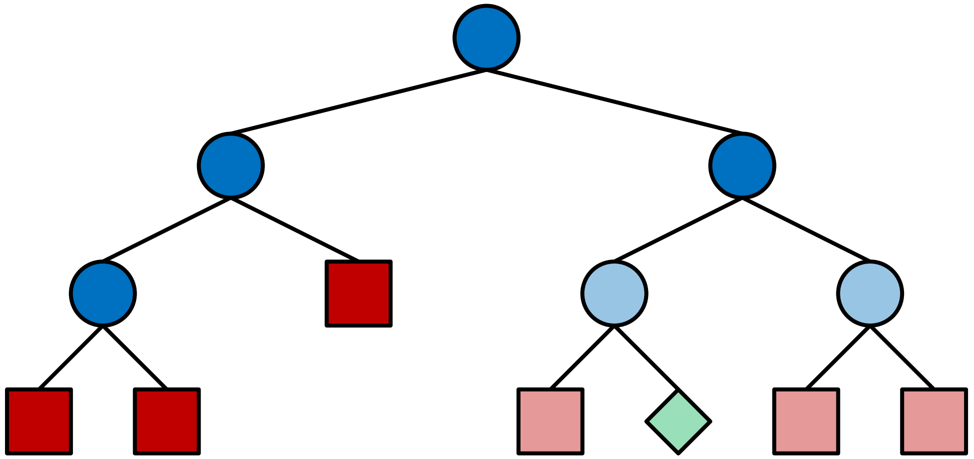
- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



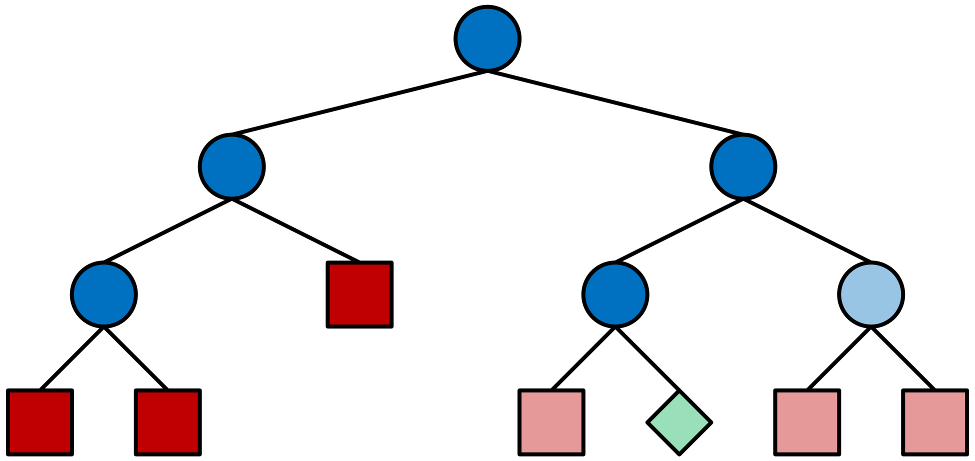
- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



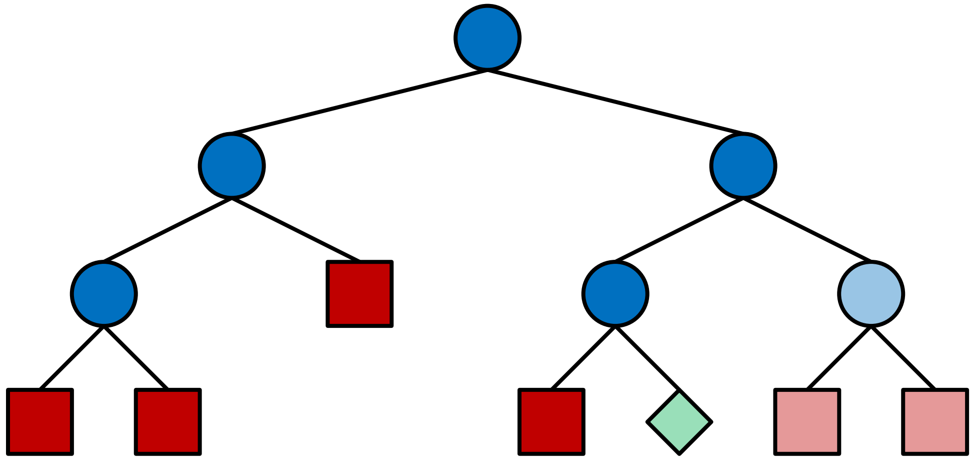
- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



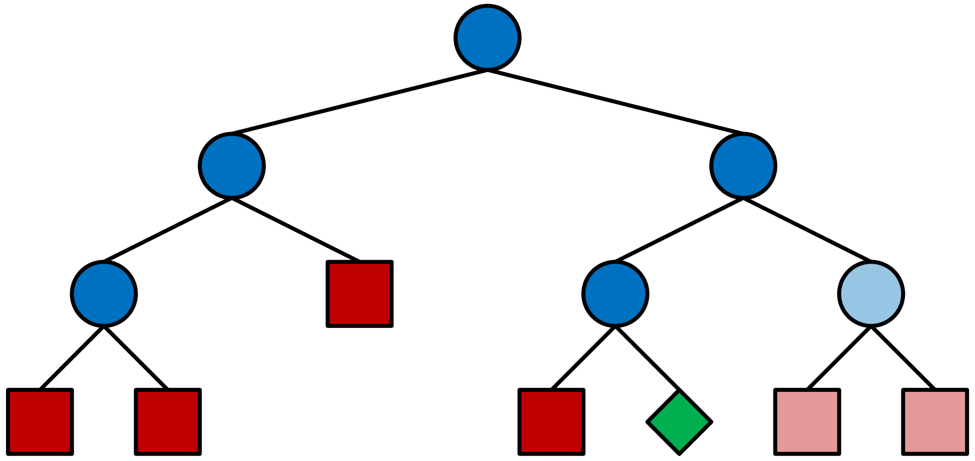
- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



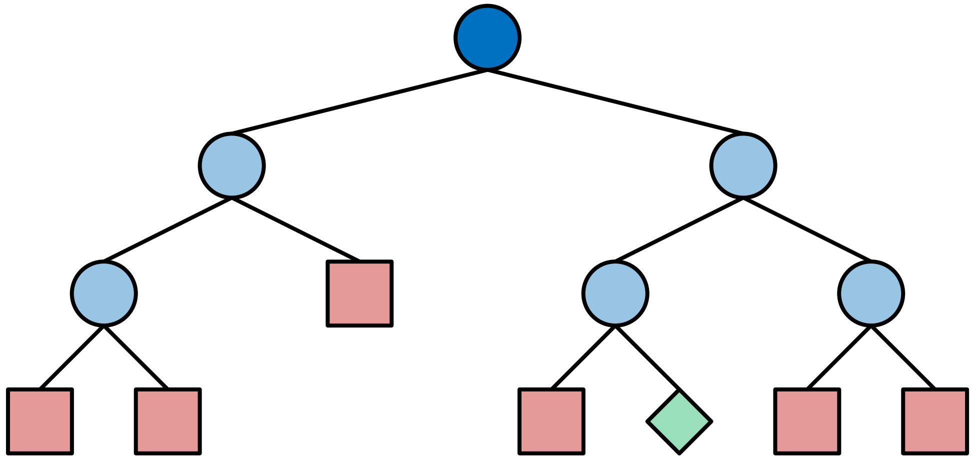
- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS,

Search: Heuristic ⇆ Exploration



- Heuristic branching
  - defines tree shape
- Exploration of search tree
  - orthogonal aspect: DFS, BFS, IDFS, LDS, parallel, ...

Summary

- Modeling
  - variables with domain
  - constraints to state relations
  - branching strategy
  - in real: an array of modeling techniques…

- Solving
  - constraint propagation
  - branching
  - search tree exploration
  - in real: an array of solving techniques…

The Constraint Programmer’s Toolbox

Widely Applicable

- Timetabling
- Scheduling
- Personnel and crew rostering
- Resource allocation
- Workflow planning and optimization
- Gate allocation at airports
- Sports-event scheduling
- Railroad: track allocation, train allocation, schedules
- Automatic composition of music
- Genome sequencing
- Frequency allocation
- …

Current Interest: Constraint-based Code Generation

- From
  - input program (in intermediate representation)
  - hardware architecture description

  generate constraint problem

- Solution = executable program code
  - simplicity: avoid bugs, …
  - flexibility: architecture change, …
  - quality: possibly optimal, …

- Ongoing project 2010 – 2015
  - funded by Ericsson and Vetenskapsrådet

Problems Are Hard

- The problems are NP hard
  - no efficient algorithm is likely to exist

- Tremendously difficult to
  - always solve any problem instance
  - scale to large instances
  - have single silver bullet method

- Property of problems…

  …not of method

  …hence no silver bullet

Why Is a Toolbox Needed?

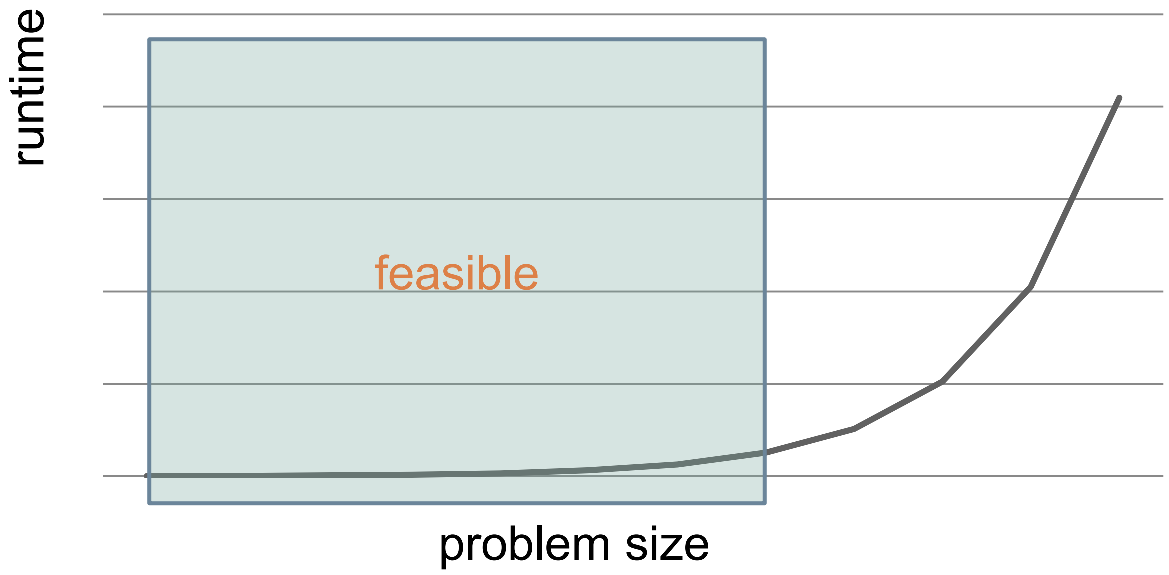
- Initial model: model to **capture** problem
  - correctness
- Improved model: model to **solve** problem
  - robustness and scalability
  - often difficult
- Tools in the toolbox are needed for…

  …**modeling to solve problems**

Parts of the Toolbox

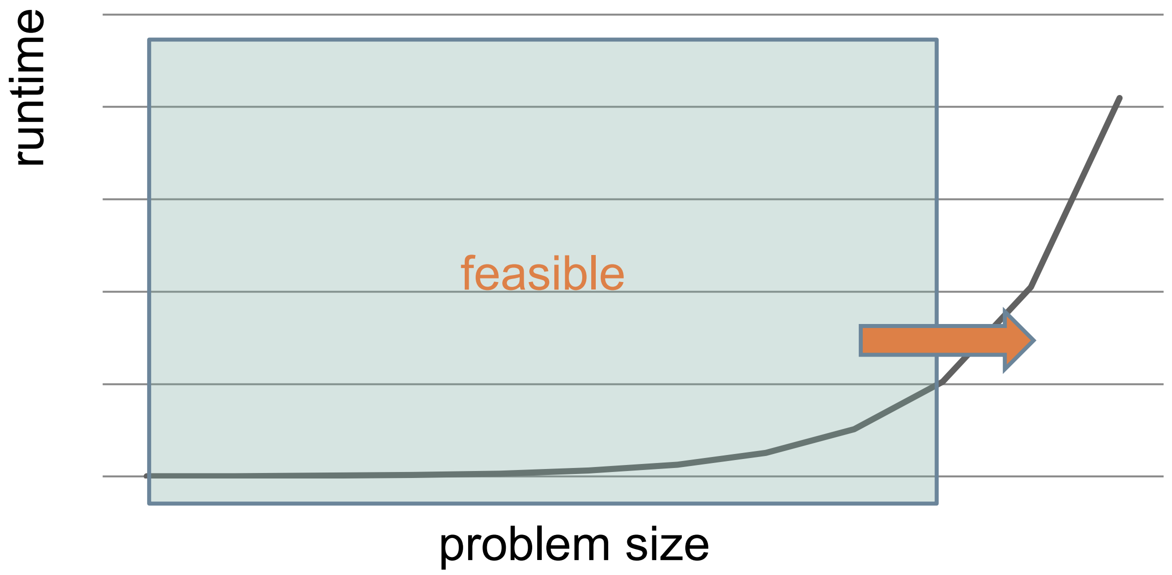
- "Global" constraints
  - capture structure during modeling
  - provide strong constraint propagation
- Search heuristics
  - application specific
- Symmetries and dominance relations
  - reduce size of search space
- Propagation-boosting constraints
- Randomized restarts during search
  - including no-goods from restarts
- …

The Best We Can Hope for…



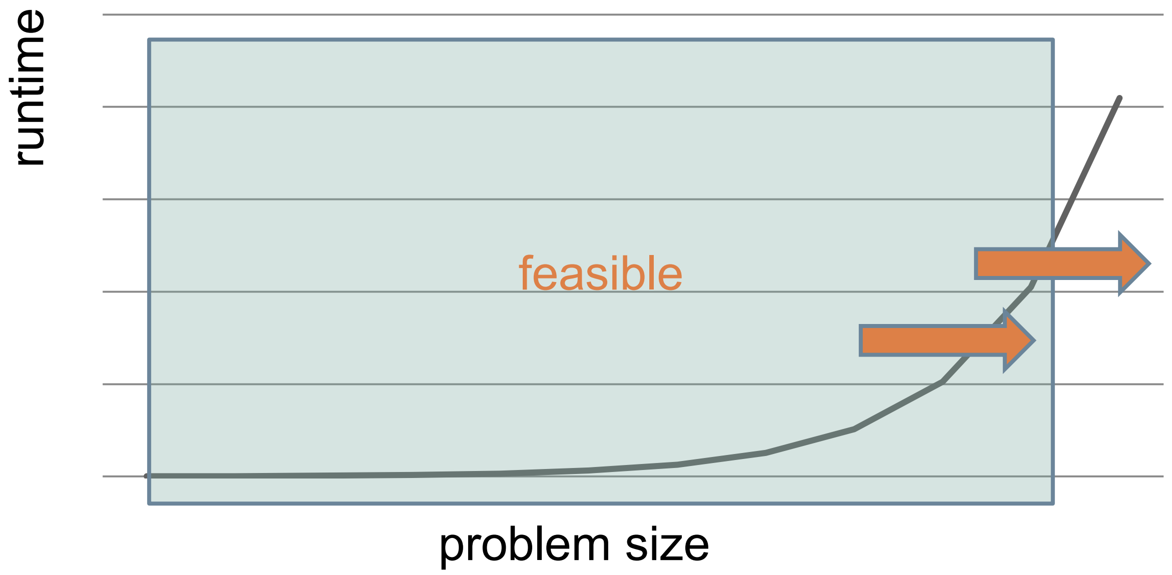
- Exponential growth in runtime
- **Without** using tools

The Best We Can Hope for…



- Exponential growth in runtime
- **With** propagation and heuristic search

The Best We Can Hope for…



- Exponential growth in runtime
- **With** propagation, heuristic search, symmetry breaking, restarts, …

Capturing Structure

distinct reconsidered

Naïve Is Not Good Enough

- `distinct`(x, y, z)
  - naïve decomposition: $x \neq y$ and $x \neq z$ and $y \neq z$
  - propagates only as soon as x, y, or z assigned

- $x \in \{1,2,3\}$, $y \in \{1,2\}$, $z \in \{1,2\}$
  - should propagate $x \in \{3\}$

- $x \in \{1,2\}$, $y \in \{1,2\}$, $z \in \{1,2\}$
  - should exhibit failure without search

Strong Propagation Idea

- distinct(x_0, …, x_4)
  - $x_0 \in \{0,1,2\}$ $x_1 \in \{1,2\}$ $x_2 \in \{1,2\}$ $x_3 \in \{2,4,5\}$ $x_4 \in \{5,6\}$
- Collect all solutions (permutations)
  - $x_0=0$ $x_1=1$ $x_2=2$ $x_3=4$ $x_4=5$
  - $x_0=0$ $x_1=1$ $x_2=2$ $x_3=4$ $x_4=6$
  - $x_0=0$ $x_1=1$ $x_2=2$ $x_3=5$ $x_4=6$
  - $x_0=0$ $x_1=2$ $x_2=1$ $x_3=4$ $x_4=5$
  - $x_0=0$ $x_1=2$ $x_2=1$ $x_3=4$ $x_4=6$
  - $x_0=0$ $x_1=2$ $x_2=1$ $x_3=5$ $x_4=6$
- Collect values from solutions
  - $x_0 \in \{0\}$ $x_1 \in \{1,2\}$ $x_2 \in \{1,2\}$ $x_3 \in \{4,5\}$ $x_4 \in \{5,6\}$

Strong Propagation Idea

- distinct(x_0, …, x_4)
  - $x_0 \in \{0,1,2\}$ $x_1 \in \{1,2\}$ $x_2 \in \{1$
- Collect all solutions (permutations)
  - $x_0=0$ $x_1=1$ $x_2=2$ $x_3=4$ $x_4=5$
  - $x_0=0$ $x_1=1$ $x_2=2$ $x_3=4$ $x_4=6$
  - $x_0=0$ $x_1=1$ $x_2=2$ $x_3=5$ $x_4=6$
  - $x_0=0$ $x_1=2$ $x_2=1$ $x_3=4$ $x_4=5$
  - $x_0=0$ $x_1=2$ $x_2=1$ $x_3=4$ $x_4=6$
  - $x_0=0$ $x_1=2$ $x_2=1$ $x_3=5$ $x_4=6$
- Collect values from solutions
  - $x_0 \in \{0\}$ $x_1 \in \{1,2\}$ $x_2 \in \{1,2\}$ $x_3 \in \{4,5\}$ $x_4 \in \{5,6\}$

infeasible: all permutations!

Strong Propagation Idea

translate into simple graph problem!

- distinct(x_0, …, x_4)
  - $x_0 \in \{0,1,2\}$ $x_1 \in \{1,2\}$ $x_2 \in \{1$
- **Characterize** all solutio
  - $x_0=0$ $x_1=1$ $x_2=2$ $x_3=4$ $x_4=5$
  - $x_0=0$ $x_1=1$ $x_2=2$ $x_3=4$ $x_4=6$
  - $x_0=0$ $x_1=1$ $x_2=2$ $x_3=5$ $x_4=6$
  - $x_0=0$ $x_1=2$ $x_2=1$ $x_3=4$ $x_4=5$
  - $x_0=0$ $x_1=2$ $x_2=1$ $x_3=4$ $x_4=6$
  - $x_0=0$ $x_1=2$ $x_2=1$ $x_3=5$ $x_4=6$
- Collect values from solutions
  - $x_0 \in \{0\}$ $x_1 \in \{1,2\}$ $x_2 \in \{1,2\}$ $x_3 \in \{4,5\}$ $x_4 \in \{5,6\}$

[Régin. A Filtering Algorithm for Constraints of Difference in CSPs. AAAI 1994]

Variable Value Graph

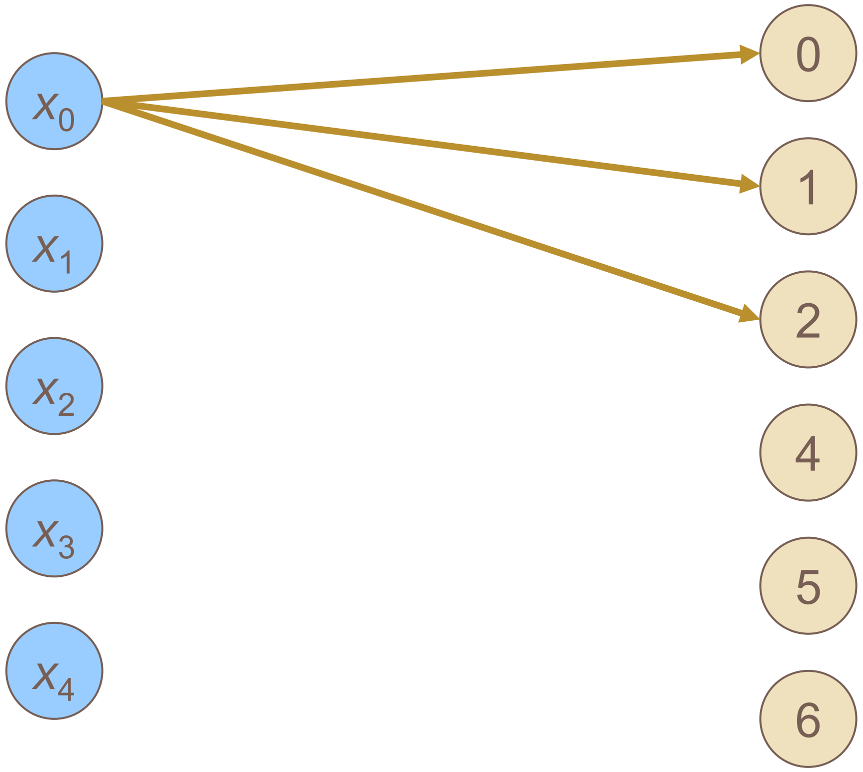
$x_0 \in \{0,1,2\}$

x_1

x_2

x_3

x_4



- Translates propagation into graph problem
  - variable nodes → value nodes

Variable Value Graph

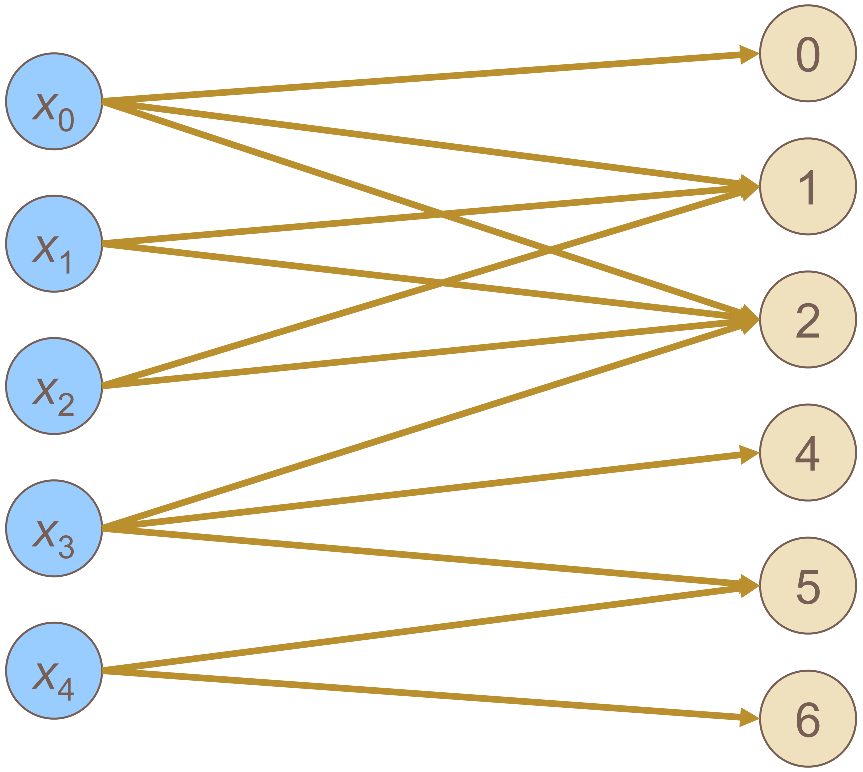
$x_0 \in \{0,1,2\}$

$x_1 \in \{1,2\}$

$x_2 \in \{1,2\}$

$x_3 \in \{2,4,5\}$

$x_4 \in \{5,6\}$



- Translates propagation into graph problem
  - variable nodes → value nodes

Graph Solution (1)

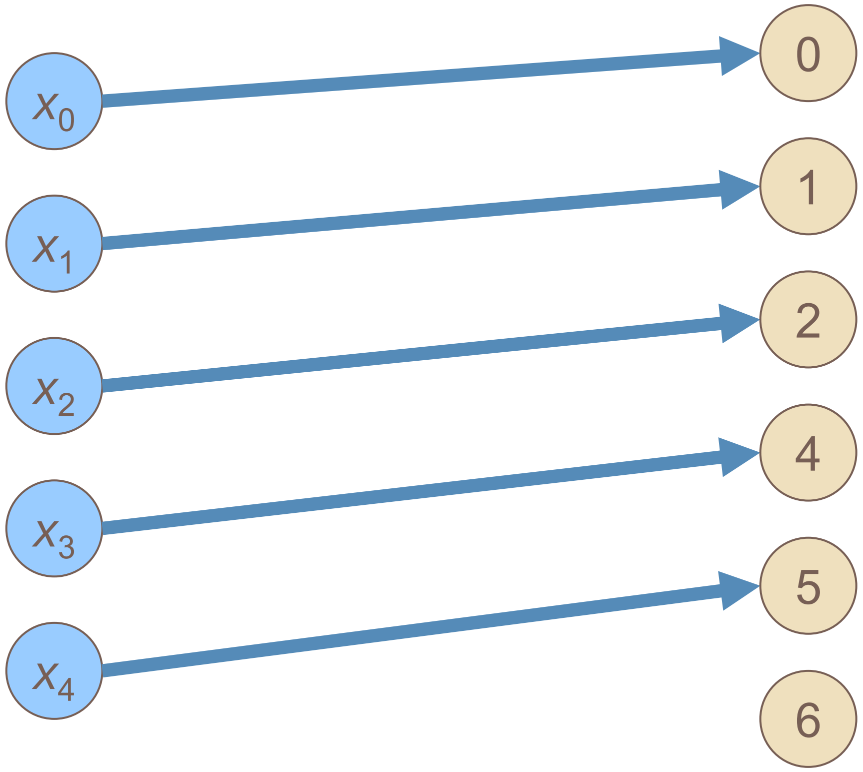
$x_0 \in \{0\}$

$x_1 \in \{1\}$

$x_2 \in \{2\}$

$x_3 \in \{4\}$

$x_4 \in \{5\}$



- Solutions maximal matchings in variable value graph
  - variable nodes → value nodes

Graph Solution (1)

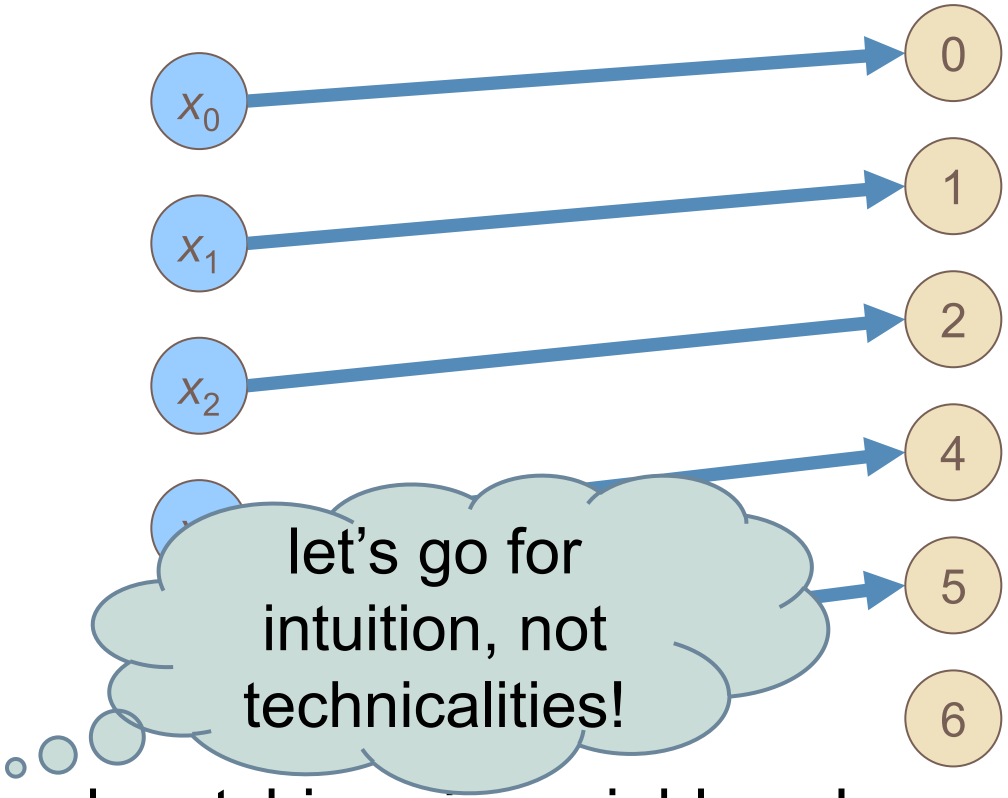
$x_0 \in \{0\}$

$x_1 \in \{1\}$

$x_2 \in \{2\}$

$x_3 \in \{4\}$

$x_4 \in \{5\}$



- Solutions maximal matchings in variable value graph
  - variable nodes → value nodes

Graph Solution (2)

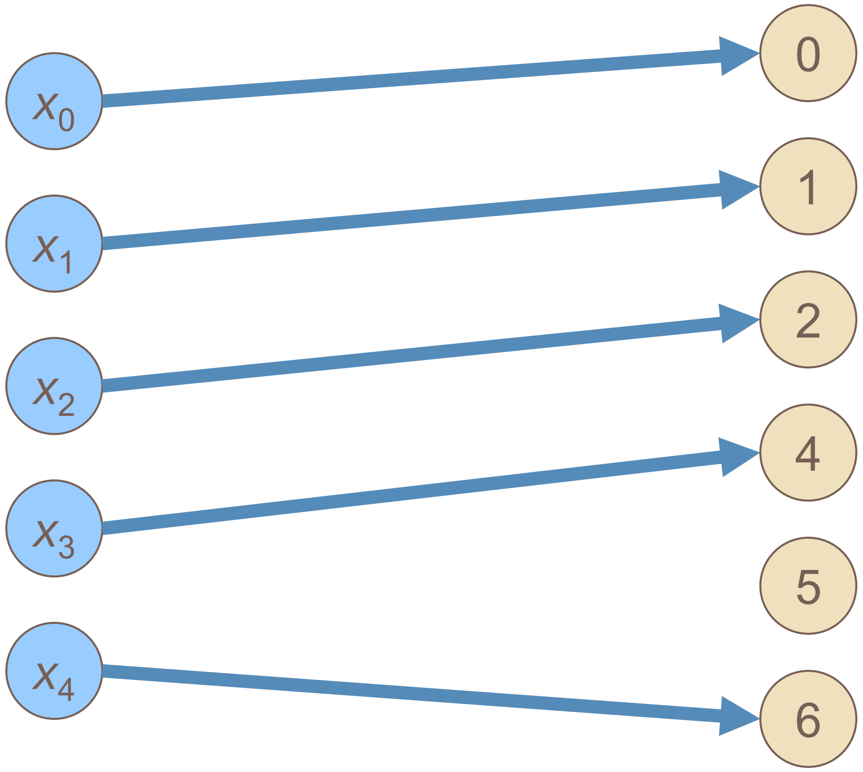
$x_0 \in \{0\}$

$x_1 \in \{1\}$

$x_2 \in \{2\}$

$x_3 \in \{4\}$

$x_4 \in \{6\}$



- Solutions maximal matchings in variable value graph
  - variable nodes → value nodes

Graph Solution (3)

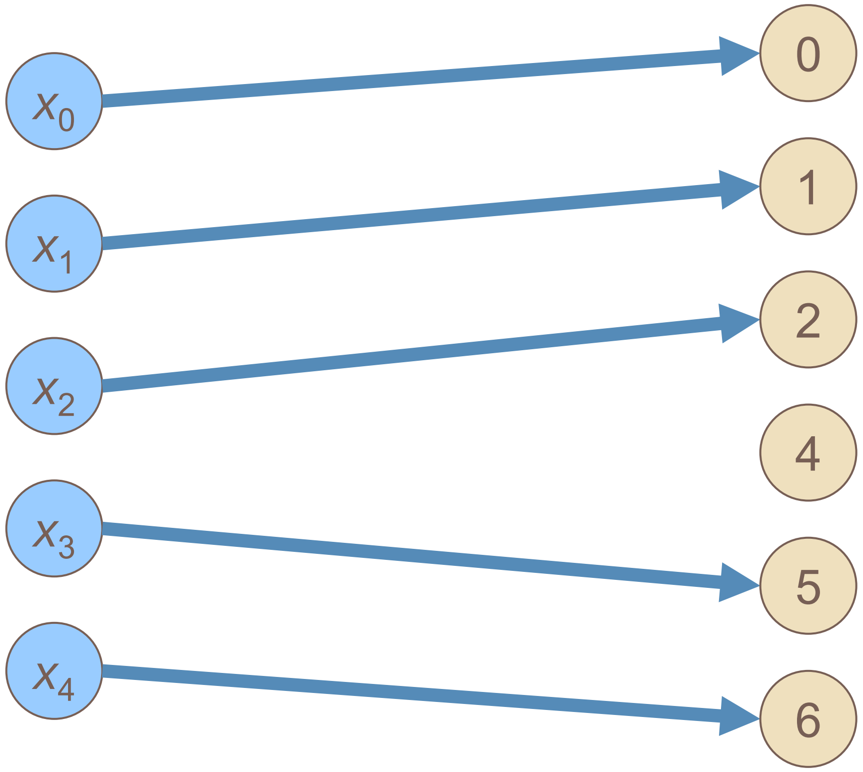
$x_0 \in \{0\}$

$x_1 \in \{1\}$

$x_2 \in \{2\}$

$x_3 \in \{5\}$

$x_4 \in \{6\}$



- Solutions maximal matchings in variable value graph
  - variable nodes → value nodes

Graph Solution (4)

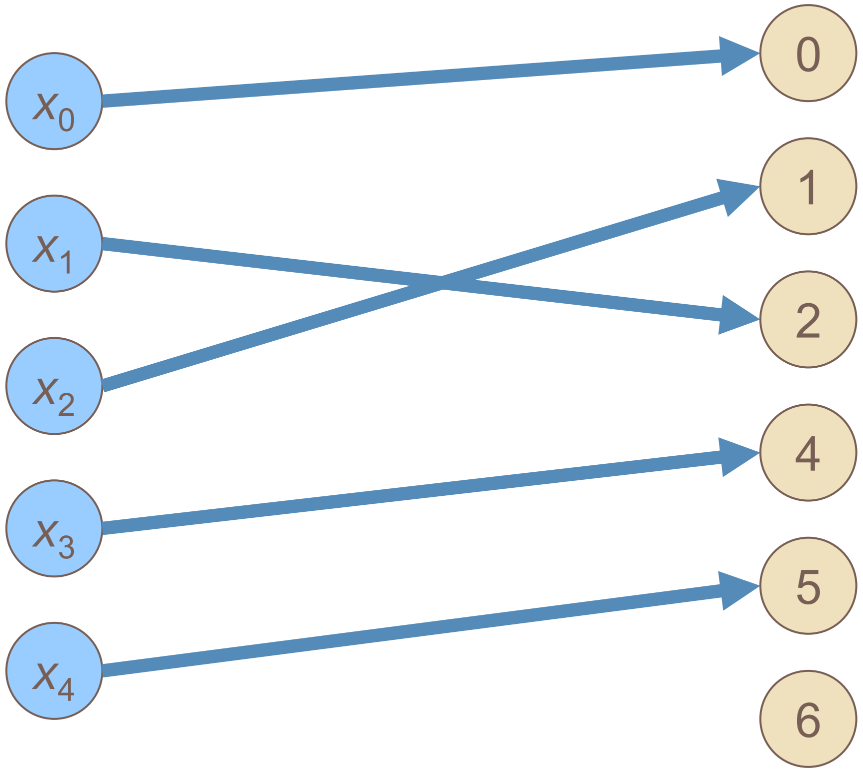
$x_0 \in \{0\}$

$x_1 \in \{2\}$

$x_2 \in \{1\}$

$x_3 \in \{4\}$

$x_4 \in \{5\}$



- Solutions maximal matchings in variable value graph
  - variable nodes → value nodes

Graph Solution (5)

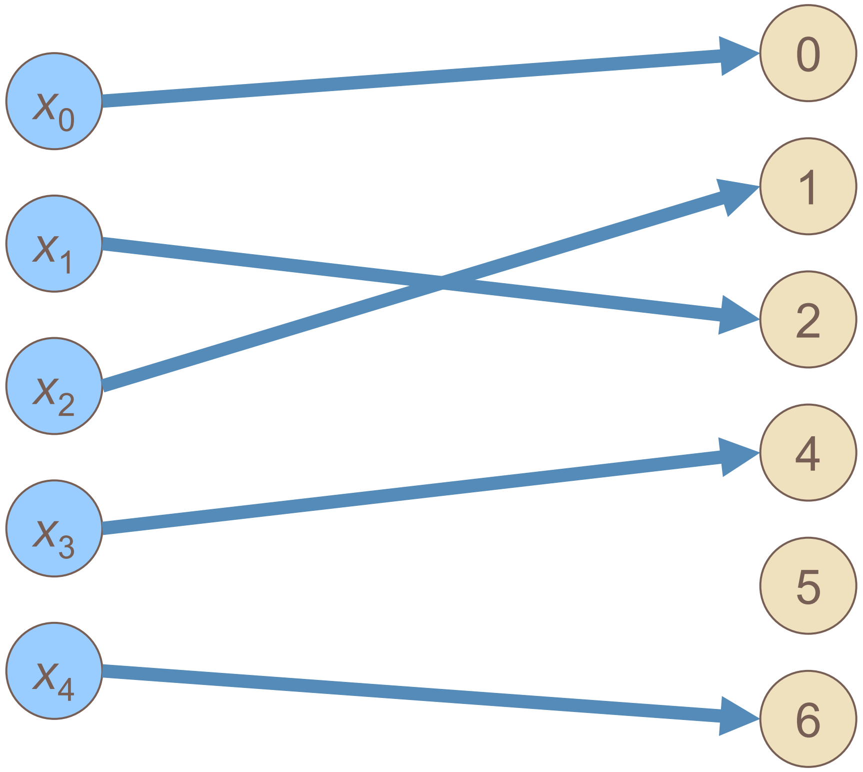
$x_0 \in \{0\}$

$x_1 \in \{2\}$

$x_2 \in \{1\}$

$x_3 \in \{4\}$

$x_4 \in \{6\}$



- Solutions maximal matchings in variable value graph
  - variable nodes → value nodes

Graph Solution (6)

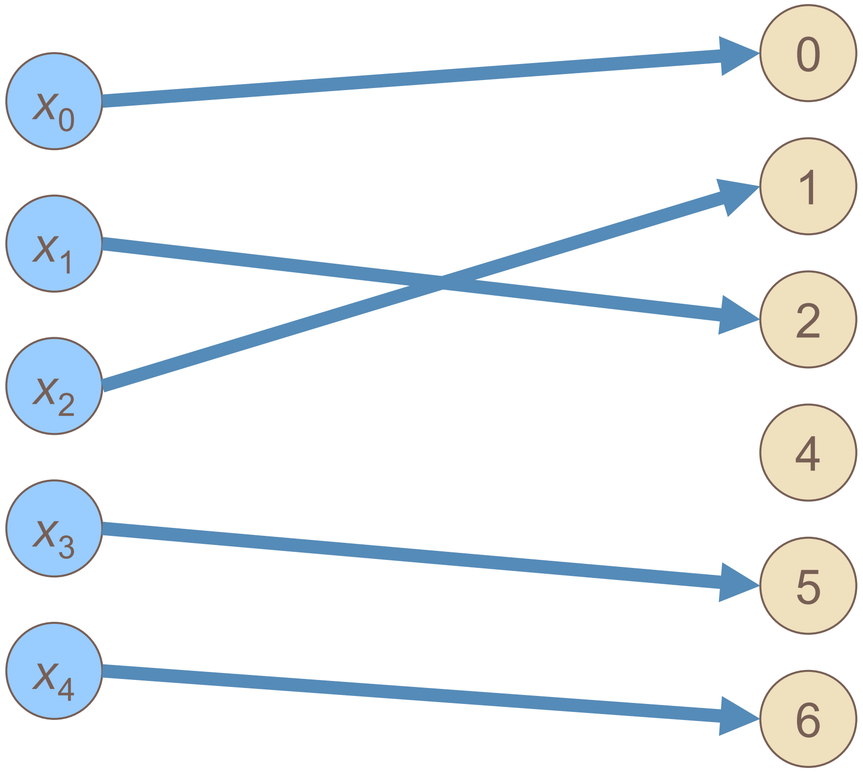
$x_0 \in \{0\}$

$x_1 \in \{2\}$

$x_2 \in \{1\}$

$x_3 \in \{5\}$

$x_4 \in \{6\}$

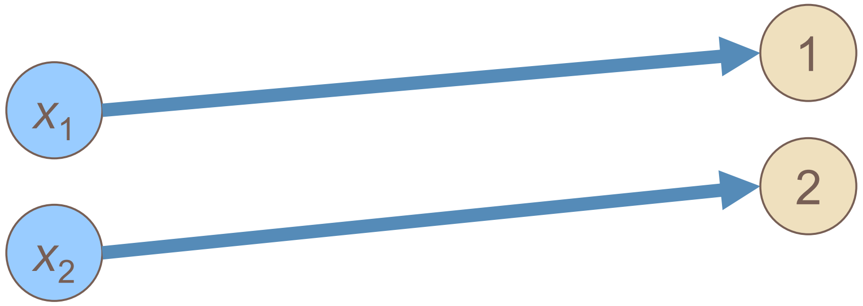


- Solutions maximal matchings in variable value graph
  - variable nodes → value nodes

Characterizing All Solutions

$x_1 \in \{1,2\}$

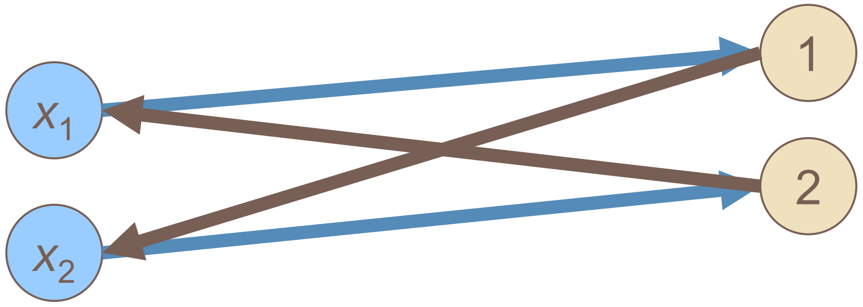
$x_2 \in \{1,2\}$



Characterizing All Solutions

$x_1 \in \{1,2\}$

$x_2 \in \{1,2\}$



- Non-matched edges in alternating cycle with matched edges...

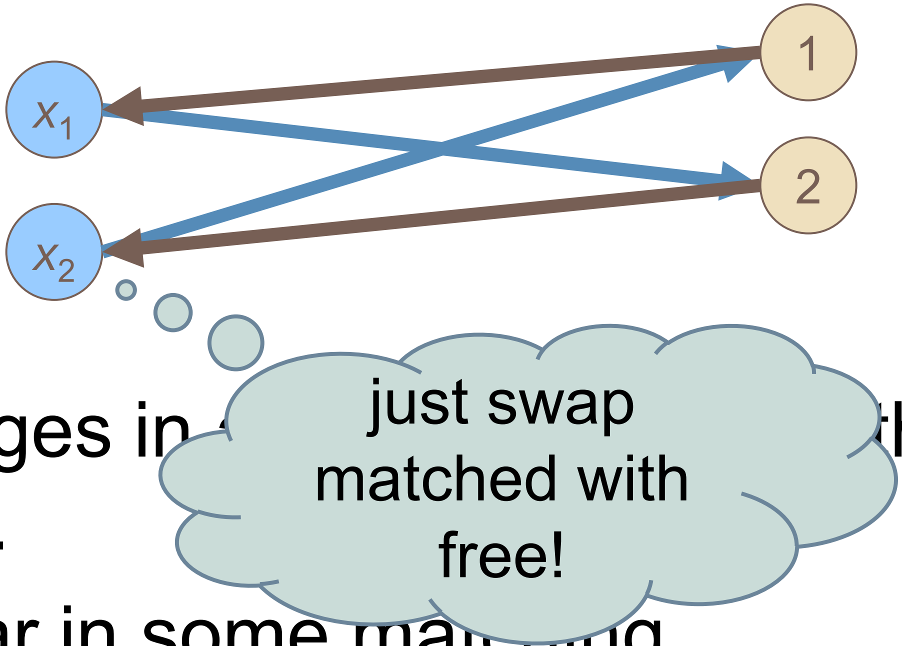
  ...appear in some matching

  ...part of some solution

Characterizing All Solutions

$x_1 \in \{1,2\}$

$x_2 \in \{1,2\}$



- Non-matched edges in [illegible] th matched edges...

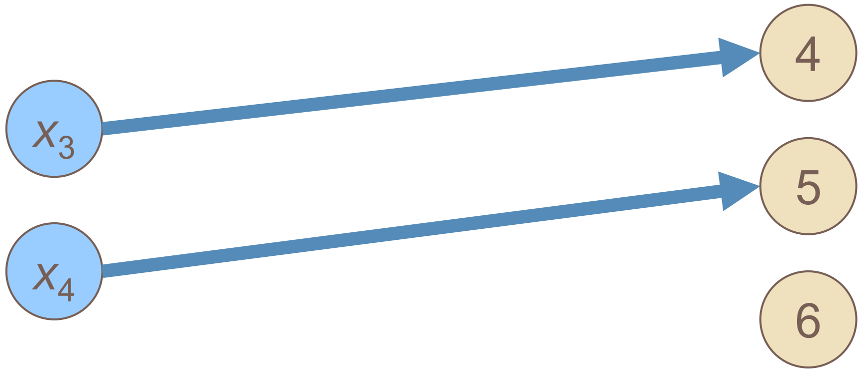
  ...appear in some matching

  ...part of some solution

Characterizing All Solutions

$x_3 \in \{2,4,5\}$

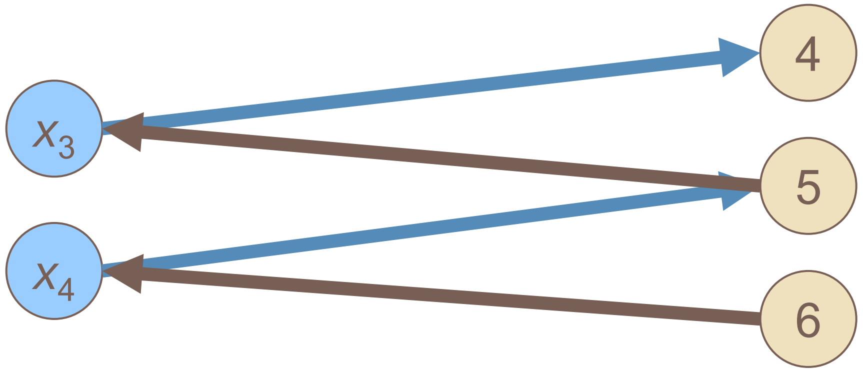
$x_4 \in \{5,6\}$



Characterizing All Solutions

$x_3 \in \{2,4,5\}$

$x_4 \in \{5,6\}$



- Non-matched edges in alternating path from unmatched node ...

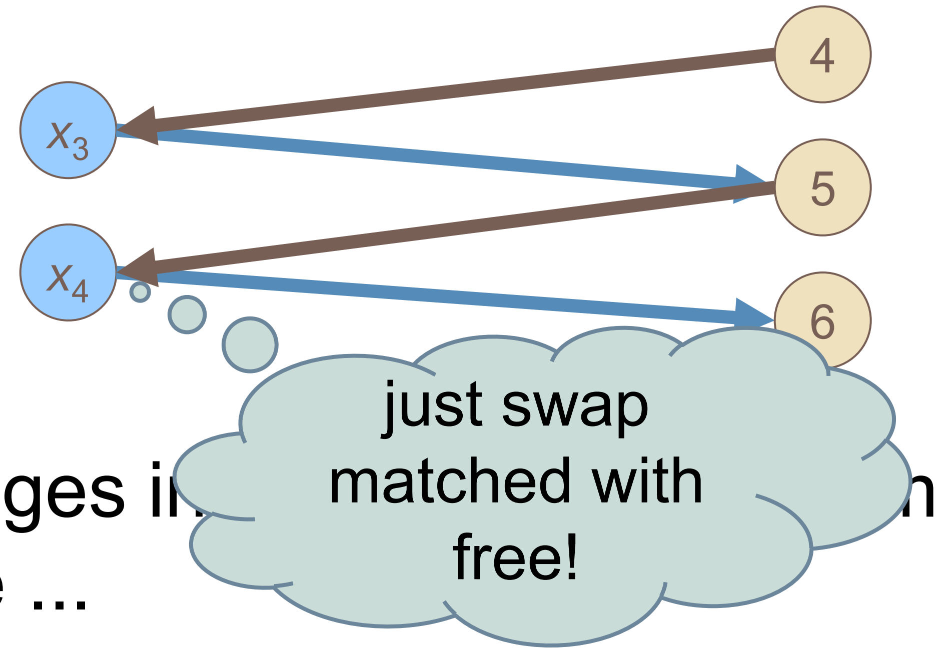
  ...appears in some matching

  ...part of some solution

Characterizing All Solutions

$x_3 \in \{2,4,5\}$

$x_4 \in \{5,6\}$



- Non-matched edges i… n unmatched node ...

  ...appears in some matching

  ...part of some solution

Variable Value Graph

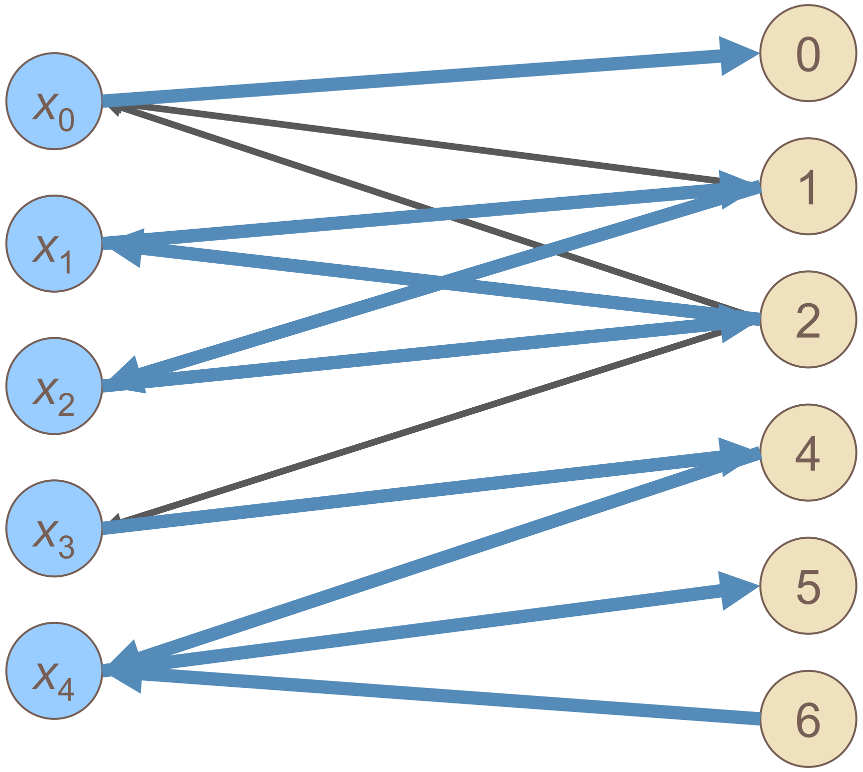
$x_0 \in \{0,1,2\}$

$x_1 \in \{1,2\}$

$x_2 \in \{1,2\}$

$x_3 \in \{2,4,5\}$

$x_4 \in \{5,6\}$



- Start from any matching (just one)
- Mark edges that can be part of a matching

Variable Value Graph

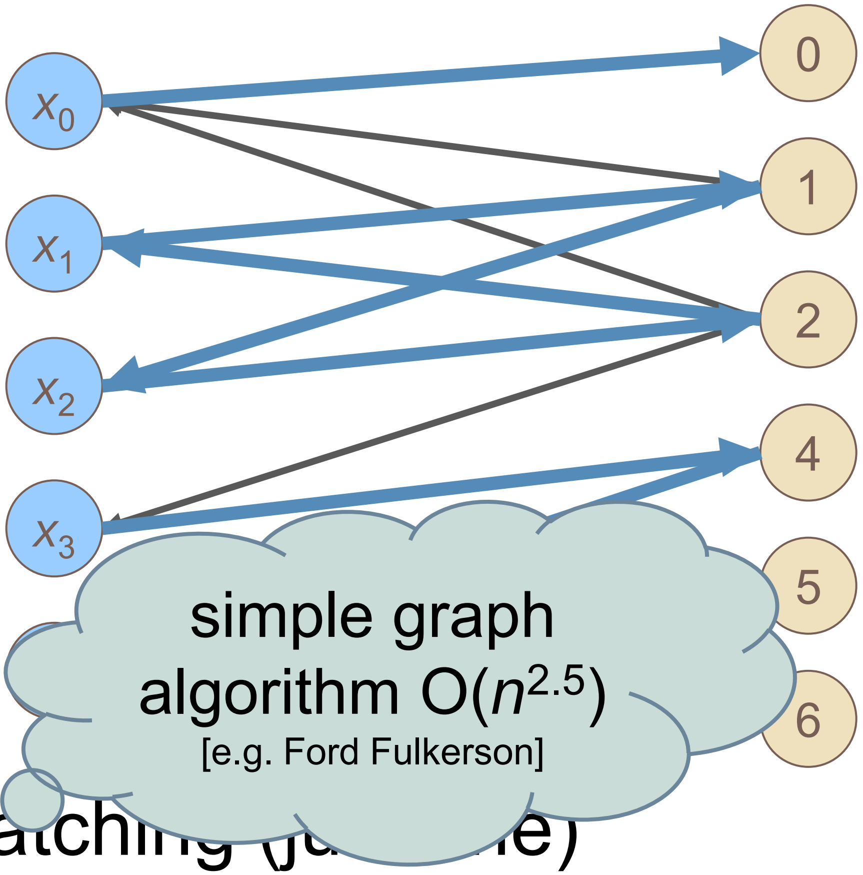
$x_0 \in \{0,1,2\}$

$x_1 \in \{1,2\}$

$x_2 \in \{1,2\}$

$x_3 \in \{2,4,5\}$

$x_4 \in \{5,6\}$



- Start from any matching (just one)
- Mark edges that can be part of a matching

Variable Value Graph

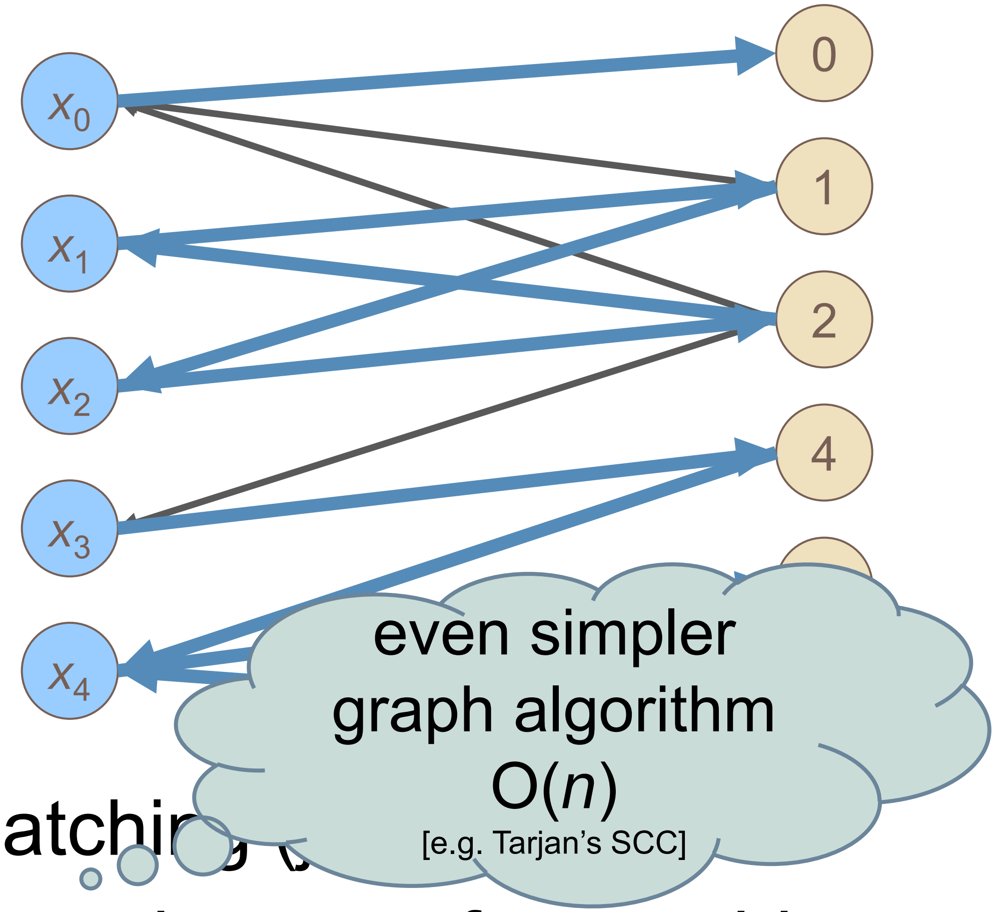
$x_0 \in \{0,1,2\}$

$x_1 \in \{1,2\}$

$x_2 \in \{1,2\}$

$x_3 \in \{2,4,5\}$

$x_4 \in \{5,6\}$



- Start from any matchin
- Mark edges that can be part of a matching

Propagation, Finally!

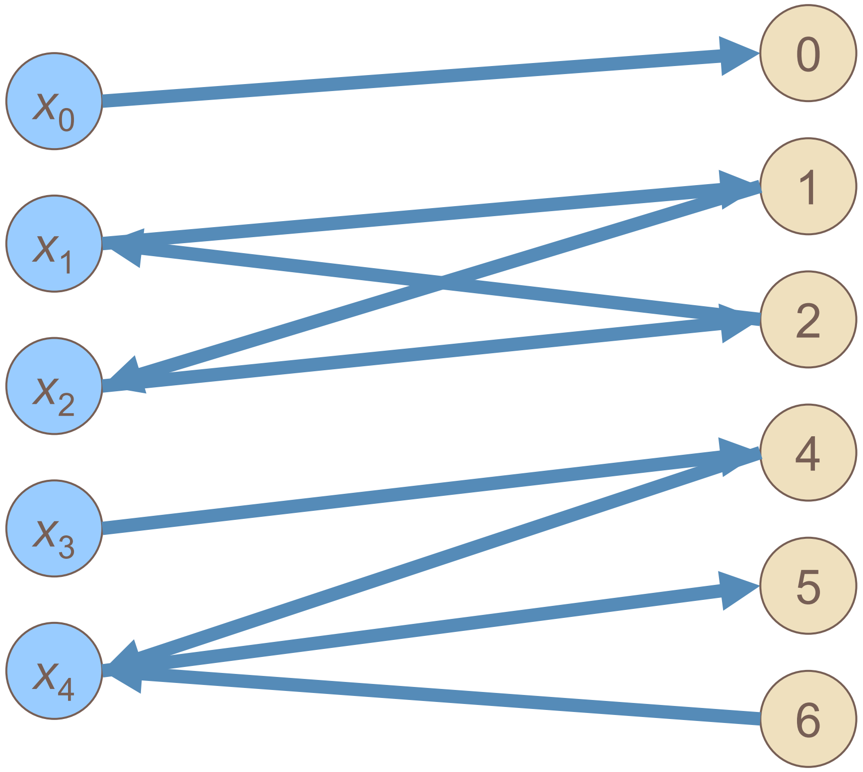
$x_0 \in \{0\}$

$x_1 \in \{1,2\}$

$x_2 \in \{1,2\}$

$x_3 \in \{4,5\}$

$x_4 \in \{5,6\}$



- Prune unmarked edges...

…and their corresponding values

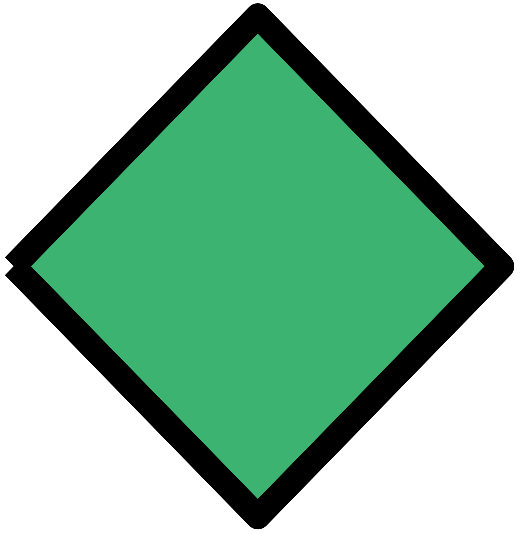
Summary

- Constraints capture problem structure (”global”)
  - ease modeling (commonly recurring structures)
  - enable solving (efficient and strong algorithms available)

- Constraints as
  - reusable
  - powerful

software components in the toolbox

SMM: Strong Propagation



```
    SEND
+   MORE
--------
=  MONEY

    9567
+   1085
--------
=  10652
```

Branching Heuristics

bin packing

Branching Heuristics

- CP advantage: programmable heuristics
  - application domain dependent: scheduling, assignment, bin-packing, …
  - requires deep insight into problem structure
  - limited reuse even though recurring principles
- CP disadvantage: universal heuristics just emerging
  - CP solver as “black box” tool
  - ultimate goal: robust and autonomous search
  - contrast to SAT and MIP
- Here: bin packing as case study for programmable heuristics

First-Fail Principle

- Could be paraphrased as:

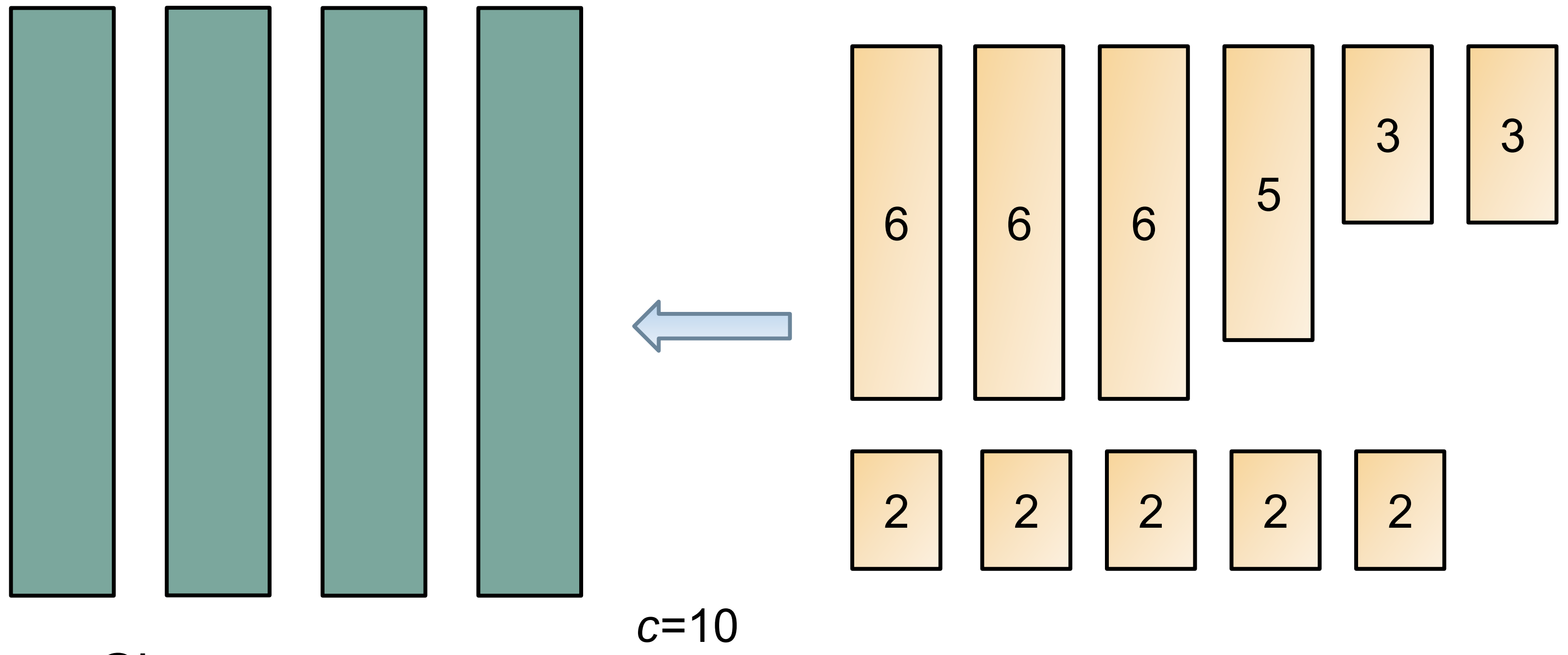
  to succeed, try first where you are most likely to fail!

  - minimize cost to find out that decision is in fact wrong
  - cost = amount of search needed (depth-first search)

- Avoid thrashing
  - make wrong decision: search will have to find out
  - make many unrelated or non-difficult decisions
  - takes ages to find that decision was wrong!

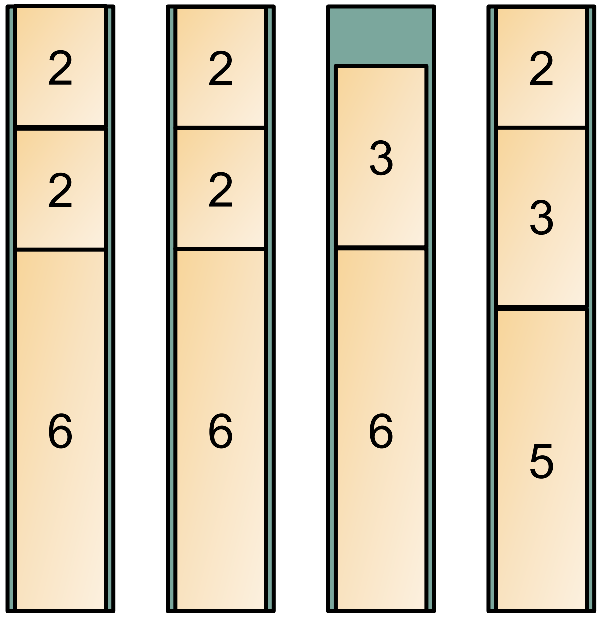
[Haralick, Elliott. Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence, 1980]

Bin Packing Problem



- Given
  - bins of capacity *c*
  - *n* items of size s_i
- Sought
  - find least number of bins such that each item packed into bin

Bin Packing Problem



c=10

- Given
  - bins of capacity c
  - n items of size s_i
- Sought
  - find least number of bins such that each item packed into bin

Simplify Problem

- Repeat simpler problem for *m* such that:

  is it possible to pack *n* items into *m* bins?

- Restrict *m* by lower bound

  $l = \lceil (s_1 + \ldots + s_n) / c \rceil$

  and upper bound

  u = #bins from some (non-optimal) packing

- Try *m* between *l* and *u*: least feasible *m* optimal

Constraint Model: Variables

- Bin variable b_i for each item i $\quad b_i \in \{1, \ldots, n\}$
  - into which bin is item i packed
- Load variable l_j for each bin j $\quad l_j \in \{0, \ldots, c\}$
  - size of items packed into bin j
- Packing variable x_{ij} $\quad x_{ij} \in \{0,1\}$
  - whether item i is packed into bin j

Constraint Model: Constraints

- Total load is size of all items

  $l_1 + \ldots + l_m = s_1 + \ldots + s_n$

- Load corresponds to items packed into bin *j*

  $l_j = s_1 \cdot x_{1j} + \ldots + s_n \cdot x_{nj}$

- Bin variables correspond to packing variables

  $x_{ij} = 1$ if and only if $b_i = j$

Constraint Model: Improved

- Use dedicated bin packing constraint

  binpacking($\langle b_1,\ldots,b_n\rangle$, $\langle s_1,\ldots,s_n\rangle$, $\langle l_1,\ldots,l_m\rangle$)

  - no packing variables needed
  - much stronger propagation

- If items *i* and *j* with *i*<*j* have same size

  $b_i \leq b_j$

  - reduce search space ("symmetry breaking")

- Assign large items ($s_i > c/2$) to fixed bins
- ...

[Shaw. A Constraint for Bin Packing. CP 2004]

How To Branch?

- Branch over the bin variables b_i
  - that is: assign items to bins
- Which item to pick first: largest!
- Which bin to pick first: tightest!
  best fit (least slack)!
- “Easy” to express with standard heuristics…
  …can programming do more?

Programming Heuristic

- Avoid search
  - perfect fit of item *i* to bin *b*: assign *i* to *b* (no search)
  - all bins have same slack: assign *i* to some *b*

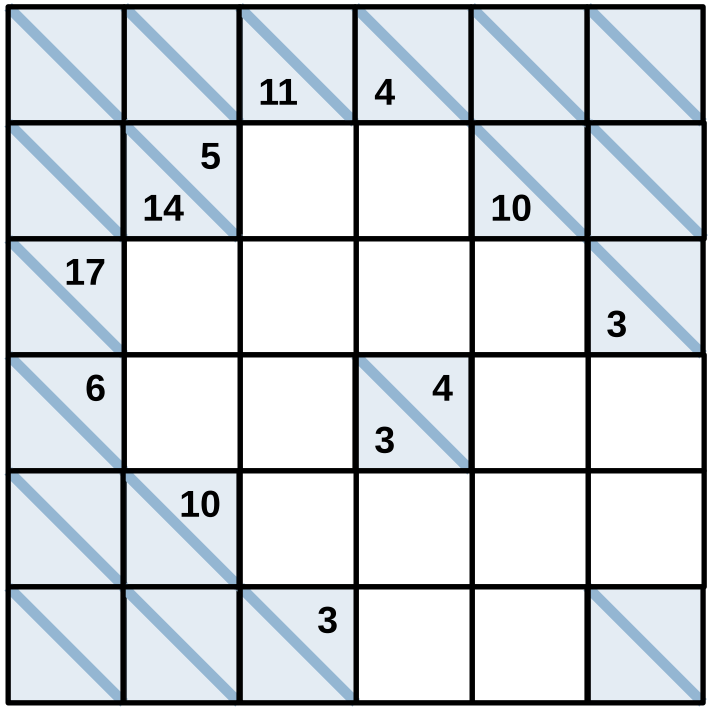
- Learn from failure
  - try to assign item *i* to bin *b*
  - if search fails: no other item *j* with $s_i = s_j$ can go to *b*
  - if search fails: item *i* cannot go to bin with same slack (also for items *j* with $s_i = s_j$)
  - "symmetry breaking during search"
  - known as CDBF: complete decreasing best-fit

[Gent, Walsh. From approximate to optimal solutions: constructing pruning and propagation rules. IJCAI 1997.]

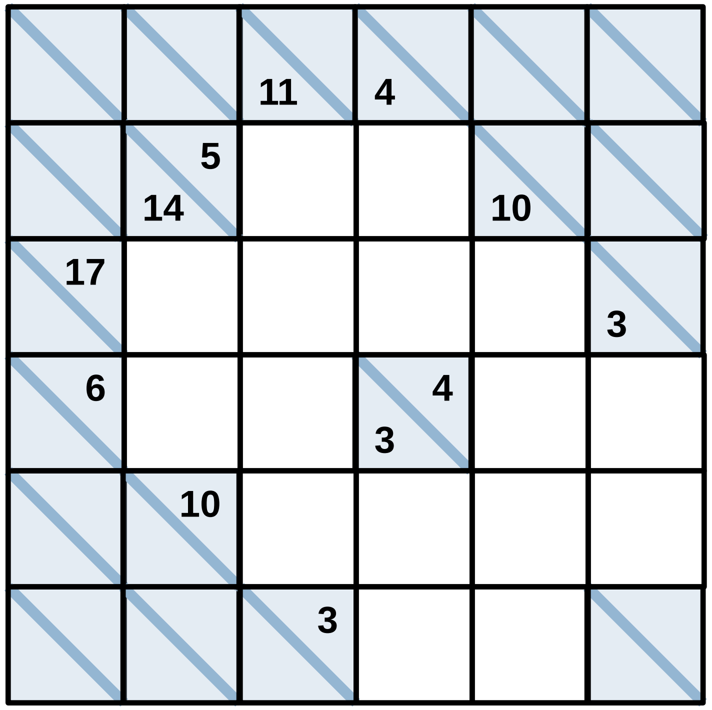
Local Reasoning

beauty and curse of
constraint programming

Kakuro

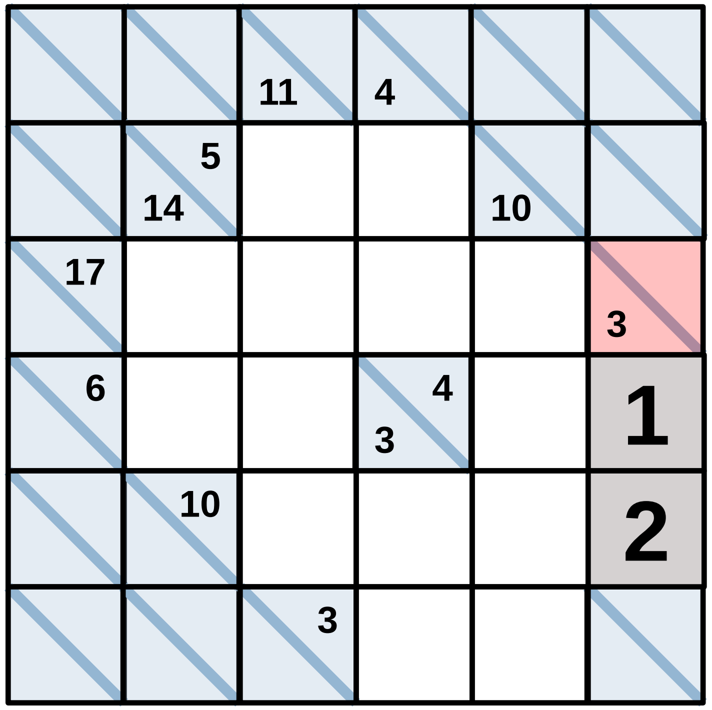


Kakuro



- Fields take digits
- Hints describe
  - for row or column
  - digit sum must be hint
  - digits must be distinct

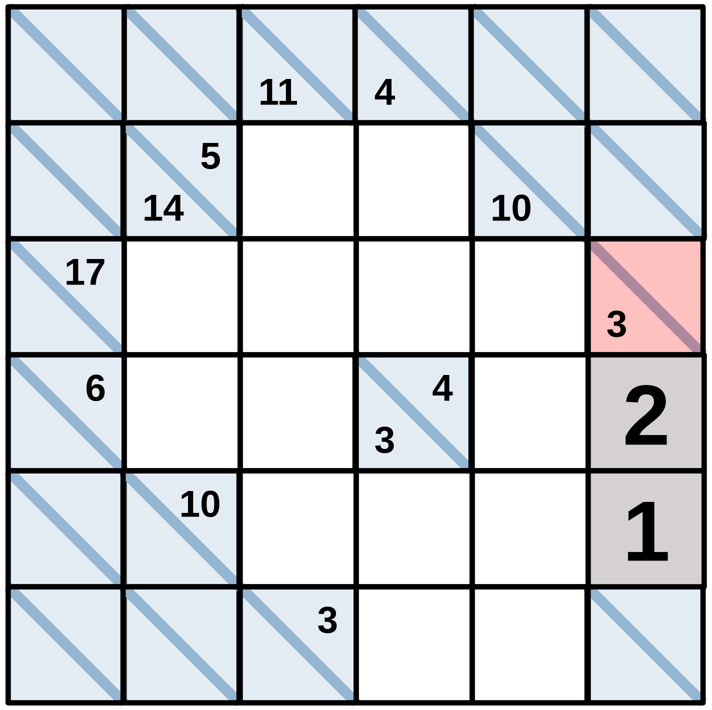
Kakuro



- For hint 3

  1 + 2

Kakuro



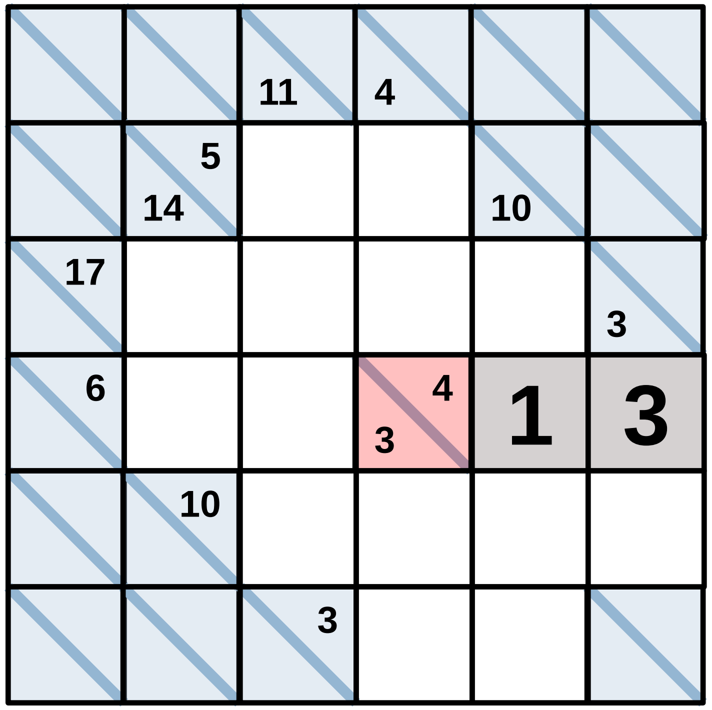
- For hint 3

  1 + 2

  or

  2 + 1

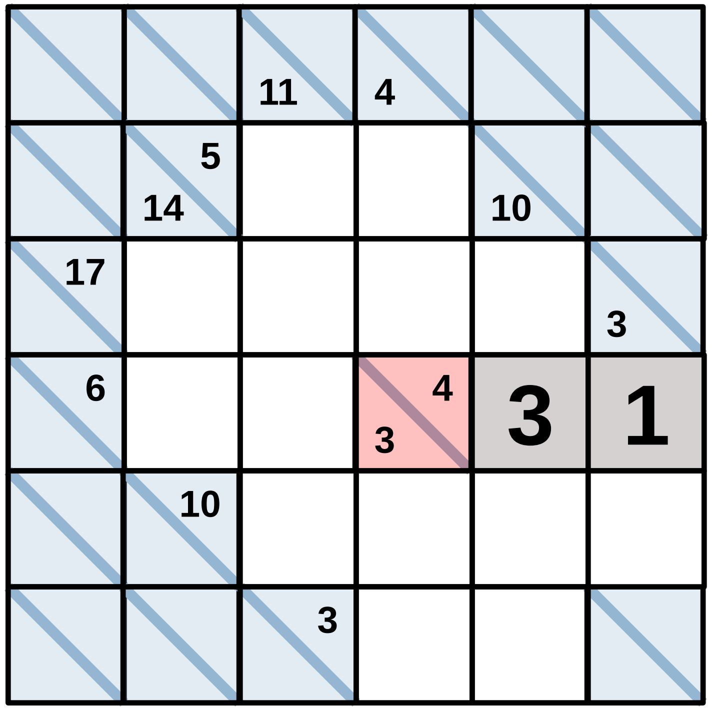
Kakuro



- For hint 4

  1 + 3

Kakuro



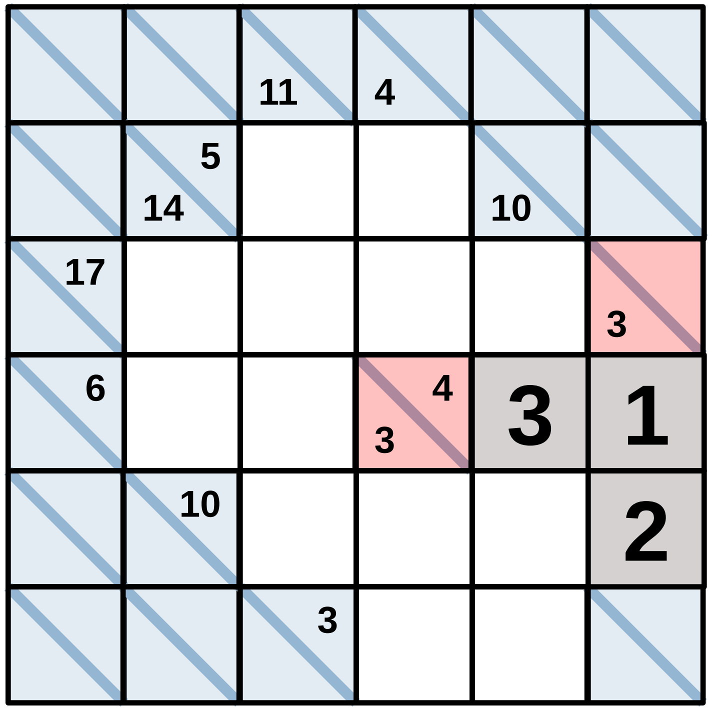
- For hint 4

  1 + 3

  or

  3 + 1

Kakuro

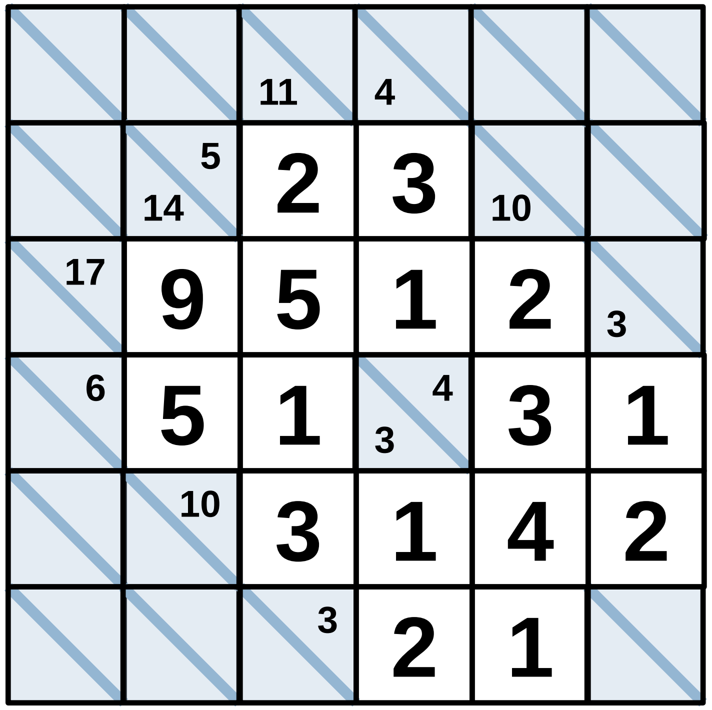


- For hint 3

  1 + 2
- For hint 4

  1 + 3

Kakuro Solution



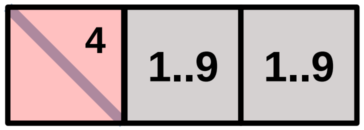
Modeling and Solving Kakuro

- Obvious model: for each hint
  - distinct constraint
  - sum constraint
- Good case... (?)
  - few variables per hint
  - few values per variable
- Let’s try it...
  - 22×14, 114 hints: 9638 search nodes, 2min 40sec
  - 90×124, 4558 hints: ? search nodes, ? minutes

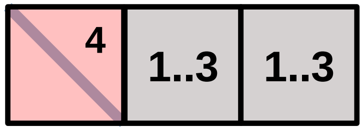
    years? centuries? eons?

Local Reasoning: Decomposition

- Possible values
  - = all digits

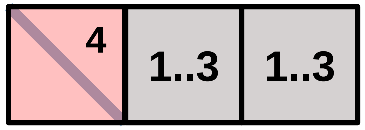


- Propagating sum = 4
  - in isolation!



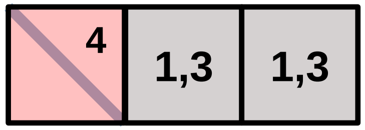
all solutions:
⟨1,3⟩ ⟨2,2⟩ ⟨3,1⟩

- Propagating distinct
  - in isolation!



all solutions:
⟨1,2⟩ ⟨1,3⟩ ⟨2,1⟩
⟨2,3⟩ ⟨3,1⟩ ⟨3,2⟩

- Propagating both
  - in combination!
  - but how?
  - where is the tool (constraint) for it?



all solutions:
⟨1,3⟩ ⟨3,1⟩

Failing for Kakuro...

- Beauty of constraint programming
  - local reasoning
  - propagators are independent
  - variables as simple communication channels

- Curse of constraint programming
  - local reasoing
  - propagators are independent
  - variables as simple communication channels

User-defined Constraints

personnel rostering

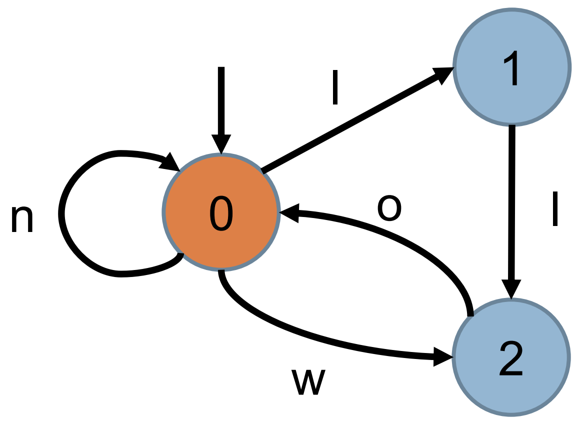
Kakuro reconsidered

Modeling Rostering: User-defined

- Personnel rostering: example (nonsensical)
  - one day off (o) after weekend shift (w)
  - one day off (o) after two consectutive long shifts (l)
  - normal shifts (n)
- Infeasible to implement propagator for ever-changing rostering constraints
- User-defined constraints: describe legal rosters by regular expression
  - (wo | llo | n)*

Regular Constraint

(wo | llo | n)*



regular(x_1, …, x_n, r)

- x_1 … x_n word in r
- or, accepted by DFA d for r

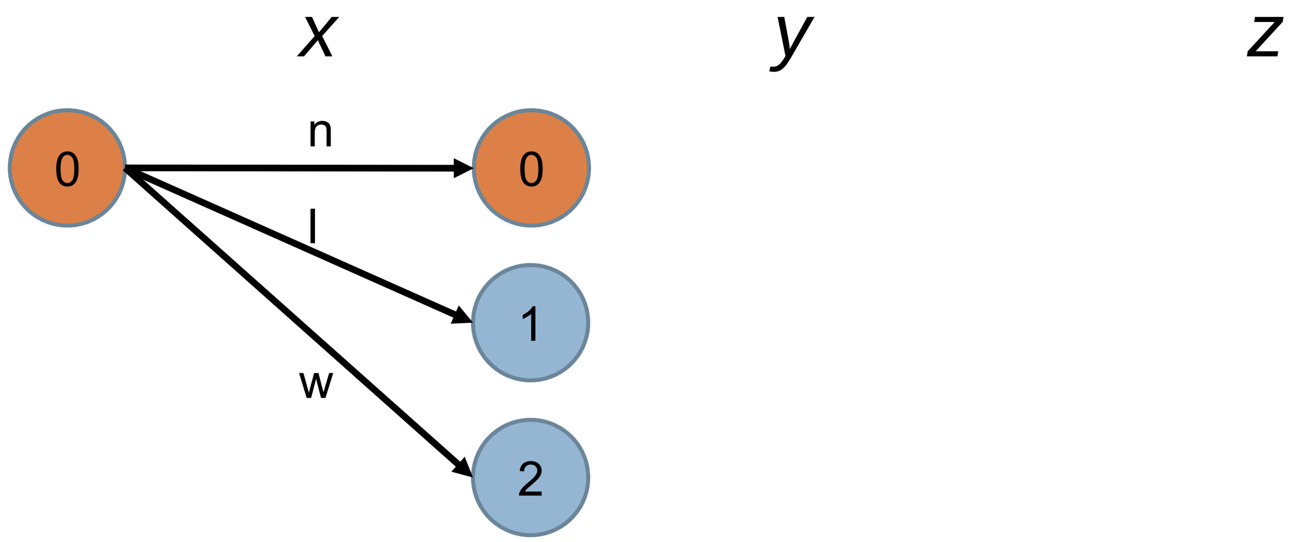
- Propagation idea: maintain all accepting paths in DFA
  - from start state (0) to a final state (0): solutions!
  - symbols on transitions comply with variable values

  [Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. CP 2004]

Propagating Regular

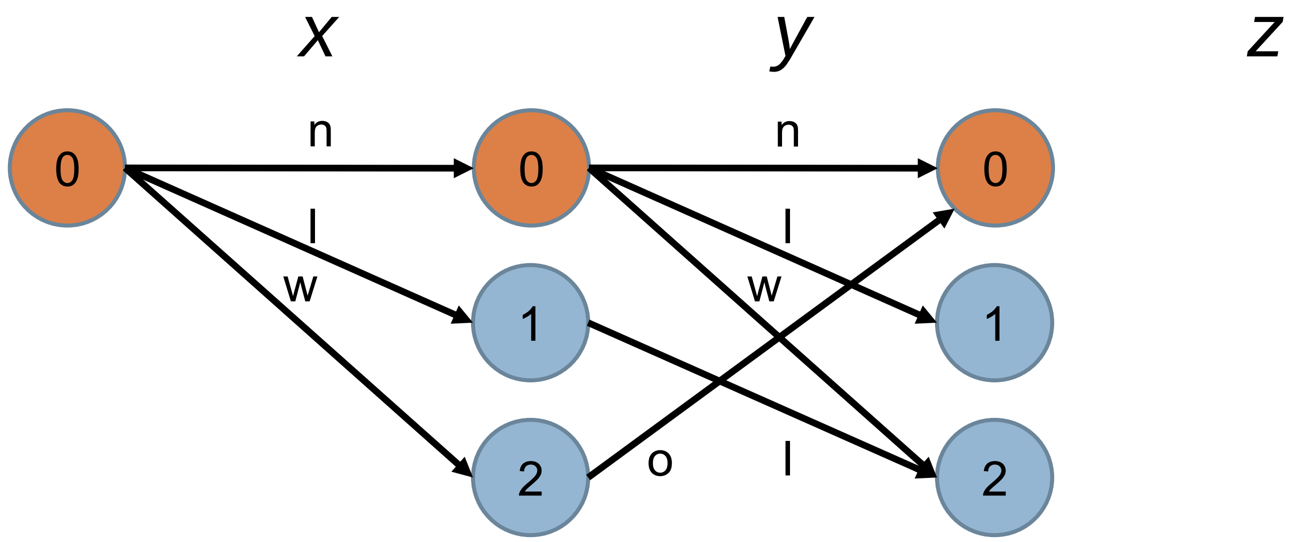
- Example: regular(x, y, z, d)
  - x, y, z in {w,o,l,n}
  - in reality: w=0, o=1, l=2, n=3

Propagating Regular



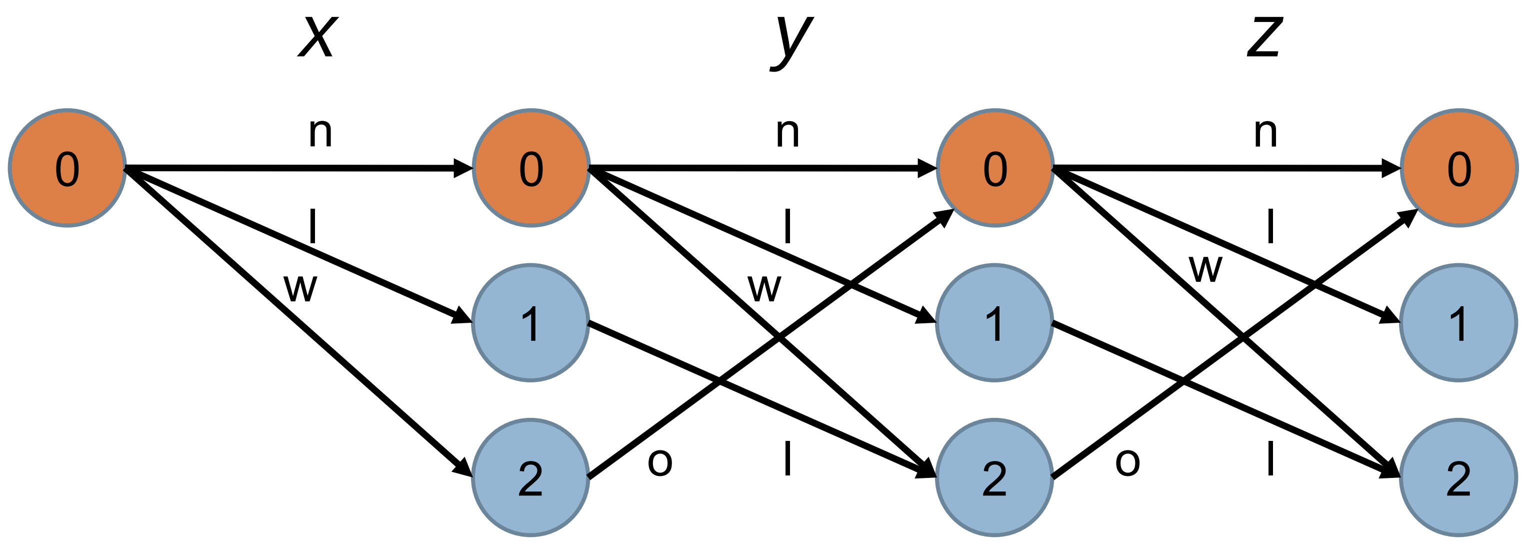
- Forward pass
  - all paths from start state

Propagating Regular



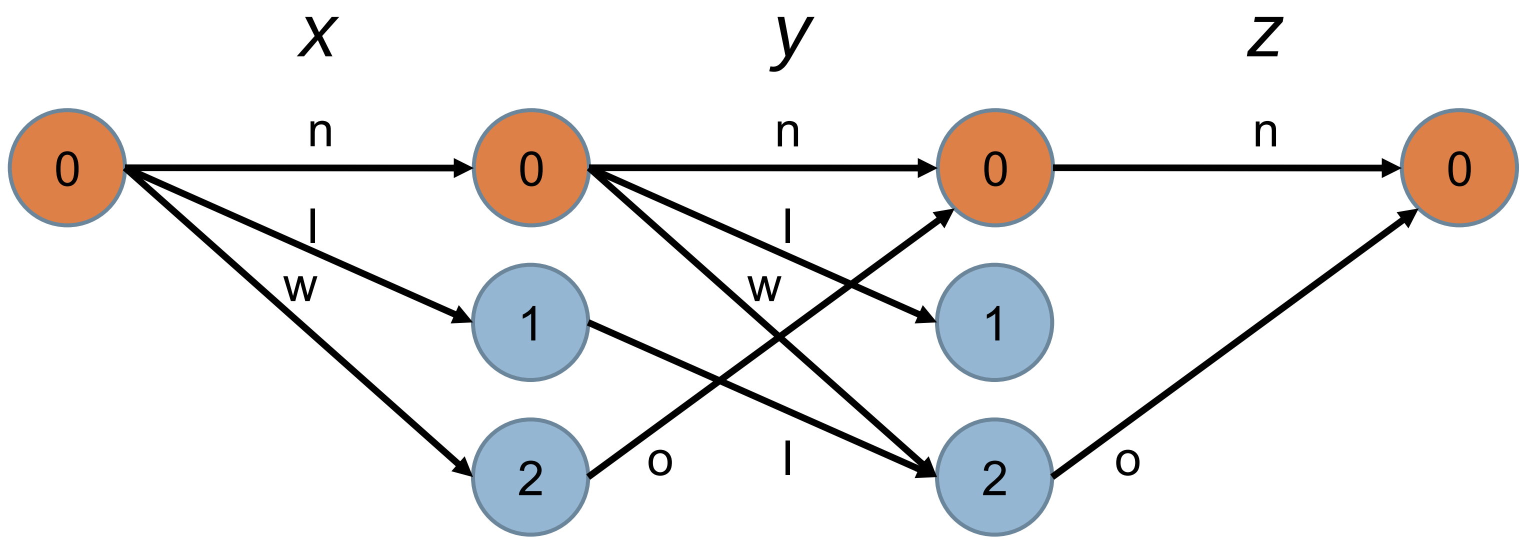
- Forward pass: optimization
  - each state at most once for each variable (“layer”)
  - several incoming/outgoing edges per state

Propagating Regular



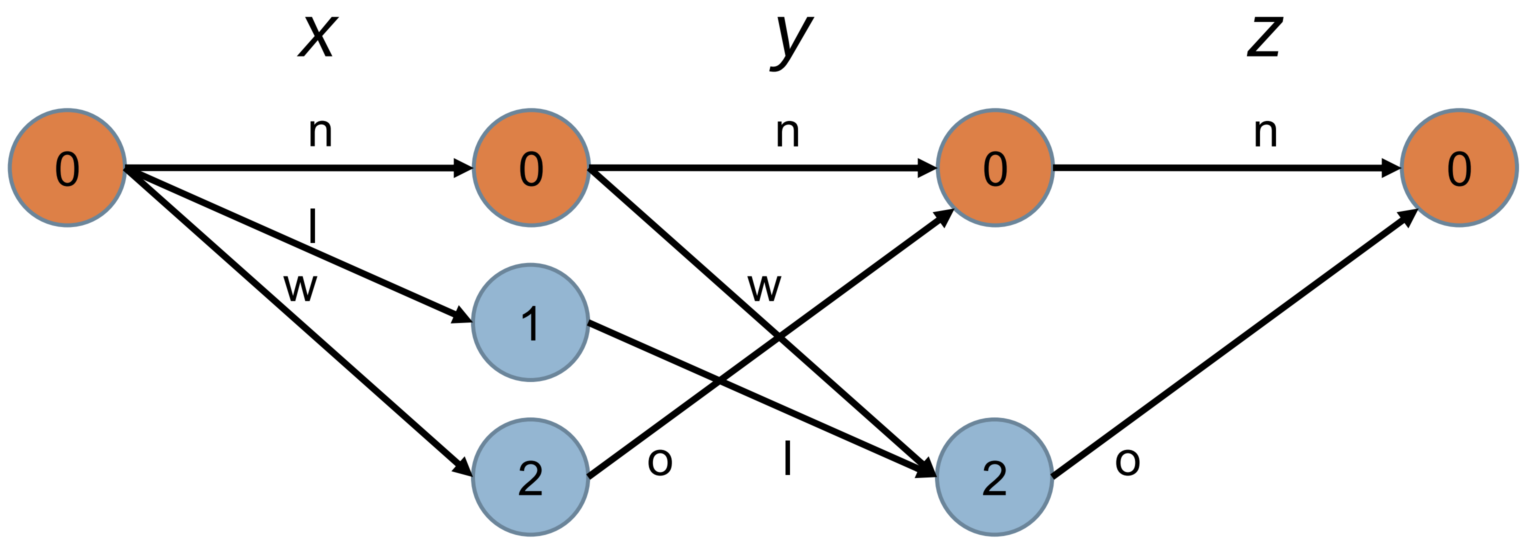
- Forward pass finished

Propagating Regular



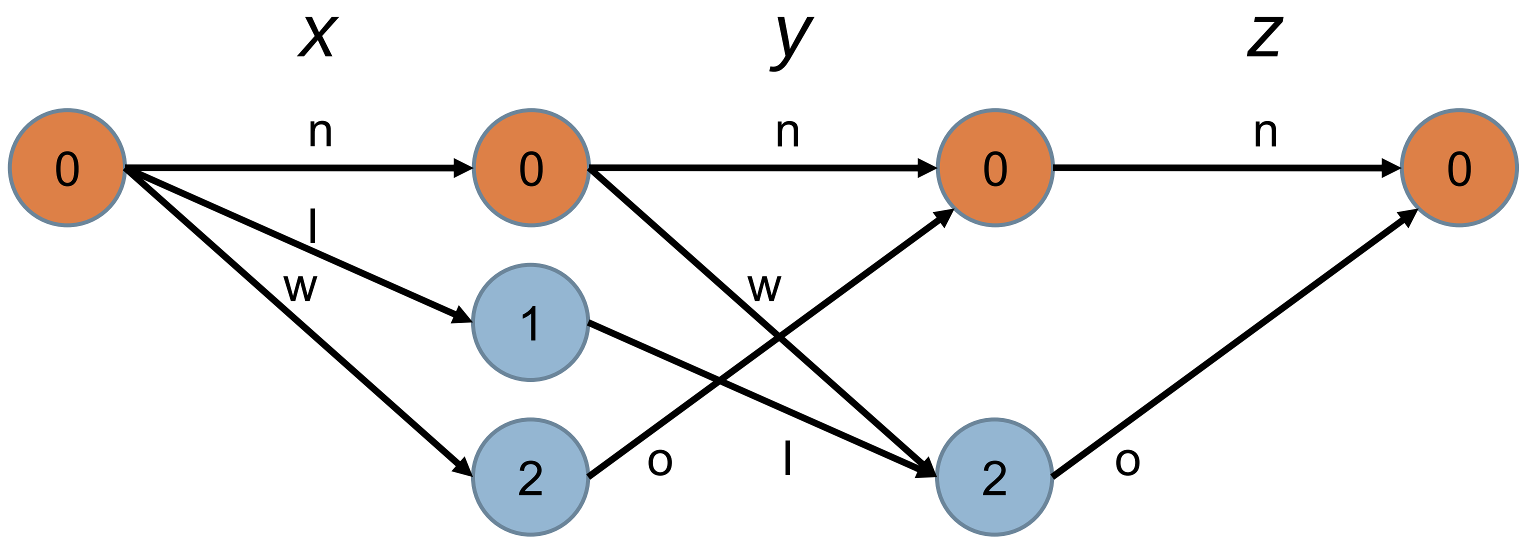
- Backward pass
  - start: remove non-final states for last layer

Propagating Regular



- Backward pass
  - start: remove non-final states for last layer
  - continue: remove states with no outgoing edges

Propagating Regular



- Pruning

  $x \in \{n,l,w\}$ $y \in \{n,l,w,o\}$ $z \in \{n,o\}$

Getting Even Better

- Variants of regular constraint
  - original regular constraint [Pesant, 2004]
  - use way more efficient MDD instead of DFA [Yap ea, 2008]
  - cost-based variants available [Pesant, ea, 2007]

Kakuro Reconsidered

- Real model: for each hint
  - one regular constraint combining distinct and sum
  - example: regular expression for hint 5 with two fields

    14 | 23 | 32 | 41
  - precompute when model is setup
- Good case...
  - few solutions for combined constraint
- Let's try again (precomputation time included)
  - 22×14, 114 hints: 0 search nodes, 28 msec
  - 90×124, 4558 hints: 0 search nodes, 345 msec

Summary

- User-defined constraints
  - high degree of flexibility
  - efficient and perfect propagation
  - limited to medium-sized constraints

- Kakuro: decomposition is harmful [again]
  - capture essential structure by few constraints
  - best by single constraint

Summary

Essence

- Constraint programming is about…
  ...local reasoning exploiting structure
  ...an array of modeling tools for solving

- Strength
  - simplicity, compositionality, exploiting structure
  - rich toolbox of techniques

- Challenges
  - lack of global picture during search
  - difficult to find global picture due to rich structure

Resources

- Overview
  - Rossi, Van Beek, Walsh, eds. Handbook of Constraint Programming, Elsevier, 2006 (around 950 pages).
- National perspective
  - Flener, Carlsson, Schulte. Constraint Programming in Sweden, *IEEE Intelligent Systems*, pages 87-89. IEEE Press, March/April, 2009.
  - SweConsNet: Swedish network for people interested in constraints. Yearly workshops, see:
    www.it.uu.se/research/SweConsNet/
- Advanced (ID2204)/graduate (ID3005) course
  - taught by me
  - period 4, 2014