CONSTRAINT PROGRAMMING FOR REAL

2009-05-27

Christian Schulte, KTH, ICT

Constraint Programming for Fun

- What is constraint programming?

Sudoku is constraint programming

- ... as a reminder ... for real, later

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			

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

- 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			

1,3,5,6,7,8

- 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			

1,3,6,7

- 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			

1,3,6

- 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, branching, exploration, ...
- Modeling
  - variables, values, constraints, heuristics, symmetries, ...

Constraint Programming for Real

- Key ideas and principles
  - constraint propagation and search
- Why does constraint programming matter?
- Excursions
  - capturing structure: distinct reconsidered
  - local reasoning: admitting failure
  - user-defined constraints: rostering
  - compositional modeling: scheduling [if time allows]
- Summary
  - strength and challenges
  - two entry pointers

Key Ideas and Principles

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 ∈ {0,…,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 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

distinct(x, y, z) $\qquad\qquad$ $x + y = 3$

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

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

Propagators

distinct(x, y, z)

$x + y = 3$

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

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

Propagators

distinct(x, y, z)

$x + y = 3$

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

- Iterate propagator execution until fixpoint
  - no more pruning possible

Propagation for SMM

- Results in store

  $S \in \{9\}$ $E \in \{4,\ldots,7\}$ $N \in \{5,\ldots,8\}$ $D \in \{2,\ldots,8\}$

  $M \in \{1\}$ $O \in \{0\}$ $R \in \{2,\ldots,8\}$ $Y \in \{2,\ldots,8\}$

- Propagation **alone** not sufficient!
  - decompose into simpler sub-problems
  - branching

Constraints and Propagators

- Constraints state relations among variables
  - which value combinations satisfy constraint
- Propagators implement constraints
  - prune values in conflict with constraint
  - freedom of what to implement (more later)
- Constraint propagation executes propagators
  - until no more pruning possible (fixpoint)

Well-behaved Propagators

- Semantic: propagator implements constraint
  - correct — no solution of constraint ever removed
  - complete — decision procedure for assignments
    propagation + search is complete
- Operational: constraint propagation works
  - contracting — values are removed
  - monotonic — stronger pruning only on stronger input
- No restriction on
  - strength — how much pruning
  - how — how propagator is implemented

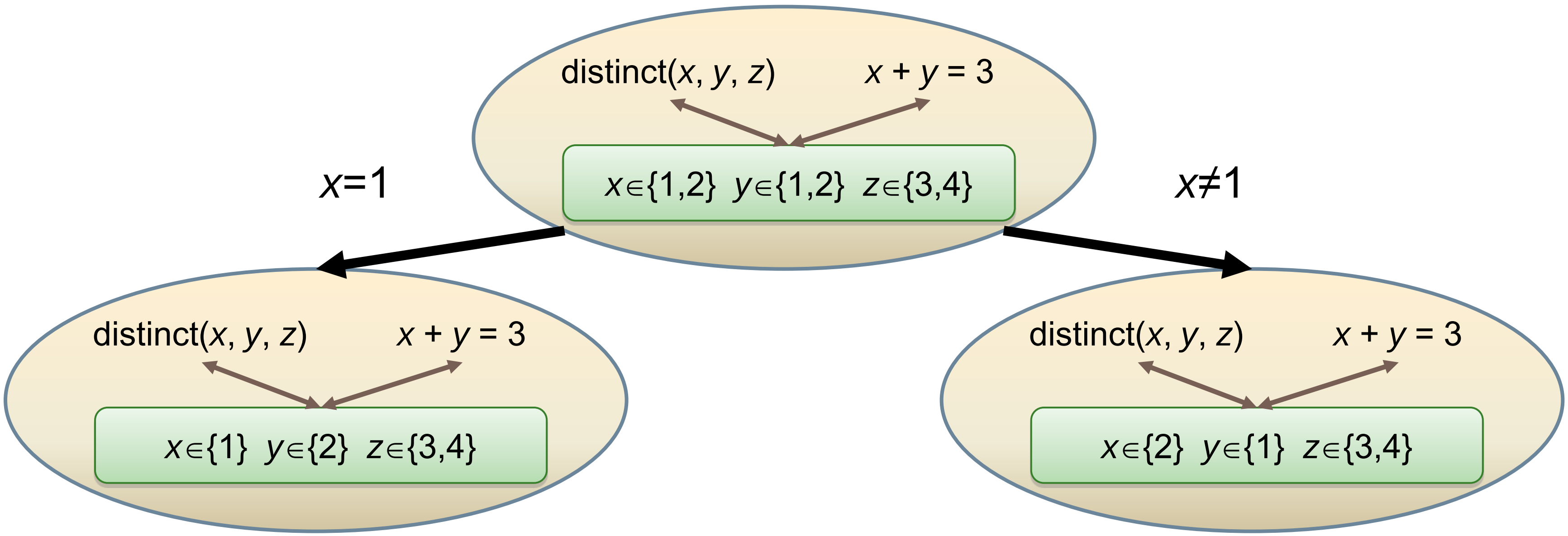
Search

branching

exploration

best solution search

Branching



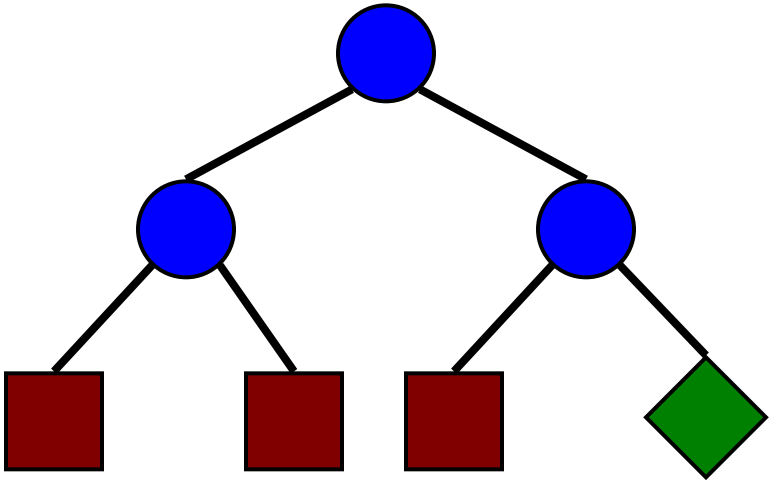
- Create subproblems with additional constraints
  - enables further propagation
  - defines search tree

Example Branching Strategy

- Pick variable x with at least two values
- Pick value n from domain of x
- Branch with

  $x=n$ and $x \neq n$

Exploration



- Iterate propagation and branching
- Orthogonal: branching ⇆ exploration
  - exploration: interactive, parallel, ...
- Nodes:
  - **unsolved**
  - **failed**
  - **solved**

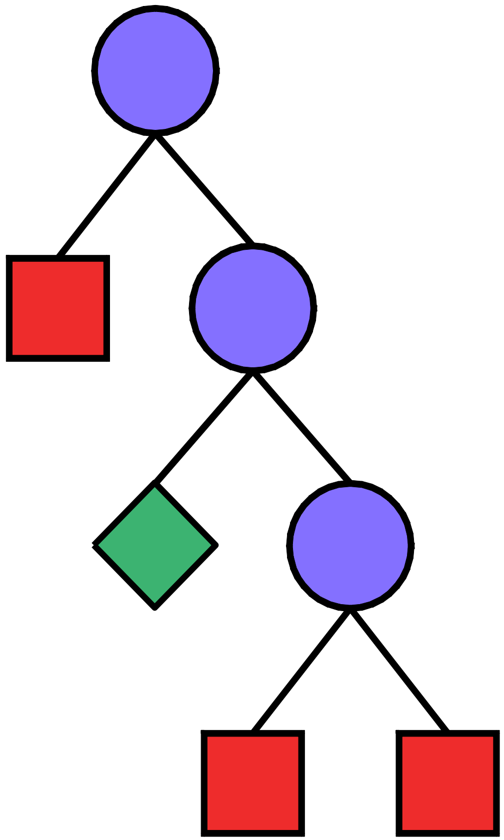
Heuristics for Branching

- Which variable
  - least possible values (first-fail)
  - application dependent heuristic
- Which value
  - minimum, median, maximum

    $x=n$ or $x \neq n$
  - split with median *n*

    $x<n$ or $x \geq n$
- Problem specific
  - most loaded resource, task with least slack, …
  - order tasks on same resource, ...

SMM: Solution With First-fail



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

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

Best Solution Search

- Naïve approach infeasible
  - compute all solutions
  - choose best

- Branch-and-bound approach
  - compute first solution
  - add “betterness” constraint to open nodes
  - next solution will be “better”
  - prunes search space

Summary

- Modeling
  - variables with domain
  - constraints to state relations
  - branching strategy
- Solving
  - constraint propagation
  - constraint branching
  - search tree exploration

Why Does CP Matter?

Widely Applicable

- Timetabling
- Scheduling
- 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
- …

Draws on Variety of Techniques

- Artificial intelligence
  - basic idea, search, ...
- Operations research
  - scheduling, flow, ...
- Algorithms
  - graphs, matchings, networks, ...
- Programming languages
  - programmability, extensionability, ...

Essential

Compositional middleware for combining

- smart algorithmic (solving)
- problem substructures (modeling)

components (propagators)

- scheduling, graphs, flows, …

while supporting

- essential extra constraints

- to be explored in the following excursions

Capturing Structure

distinct (alldifferent) reconsidered

Distinct Propagator

- Infeasible: no dedicated propagator
  - decompose `distinct(`x_1`, ..., `x_n`)`
  - into $x_i \neq x_j$ ($1 \leq i < j \leq n$) disequality propagators
  - too many propagators $O(n^2)$, propagation too weak
- Not much better: naive distinct propagator
  - wait until variable becomes assigned
  - remove value from all other variables
  - propagation too weak

Naïve Is Not Good Enough

- `distinct`(x, y, z)
  - decomposition: $x \neq y$ and $x \neq z$ and $y \neq z$
- $x \in \{1,2,3\}$, $y \in \{1,2\}$, $z \in \{1,2\}$
  - should propagate $x \in \{3\}$
- $x \in \{1,2\}$, $y \in \{1,2\}$, $z \in \{1,2\}$
  - should exhibit failure without search

Strong Distinct Propagator

- Strong - global - distinct propagator
  - only keep values appearing in a solution to constraint
  - essential for many problems (permutation problems)
  - takes global perspective on constraint
  - is strongest: domain-consistent, hyper-arc consistent, ...

- Can be propagated efficiently
  - O($n^{2.5}$) is efficient [Régin, 1994]

- Uses graph algorithms
  - solutions of constraint ⇔ properties of graph
  - characterize all solutions: prune excess values

Variable Value Graph

$s(x_0)=\{0,1\}$

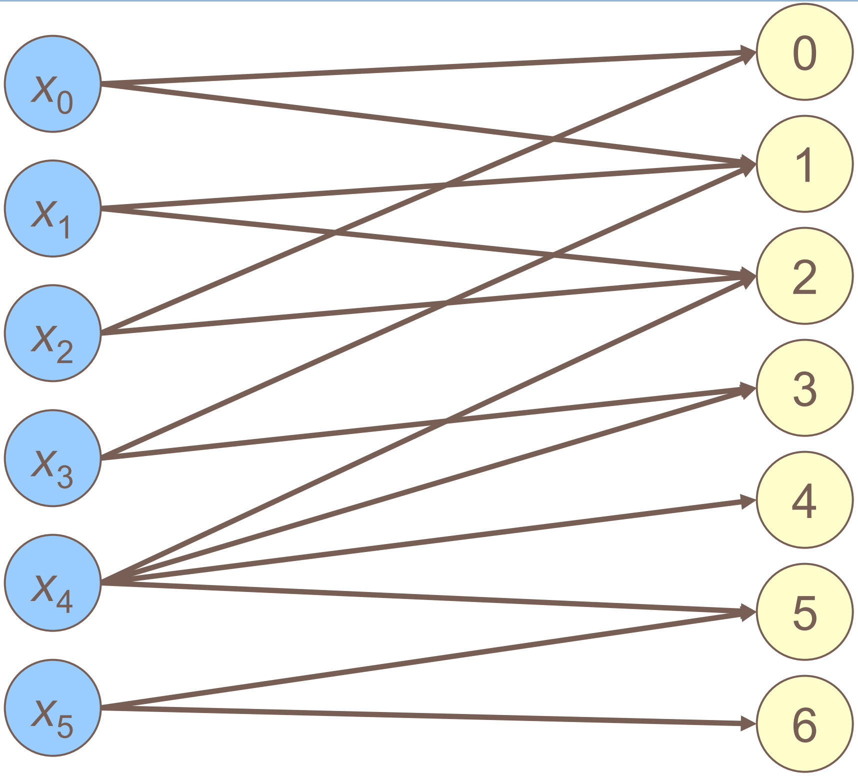
$s(x_1)=\{1,2\}$

$s(x_2)=\{0,2\}$

$s(x_3)=\{1,3\}$

$s(x_4)=\{2,3,4,5\}$

$s(x_5)=\{5,6\}$



- Bipartite graph
  - variable nodes → value nodes

Solution: Maximal Matching

$s(x_0)=\{0,1\}$

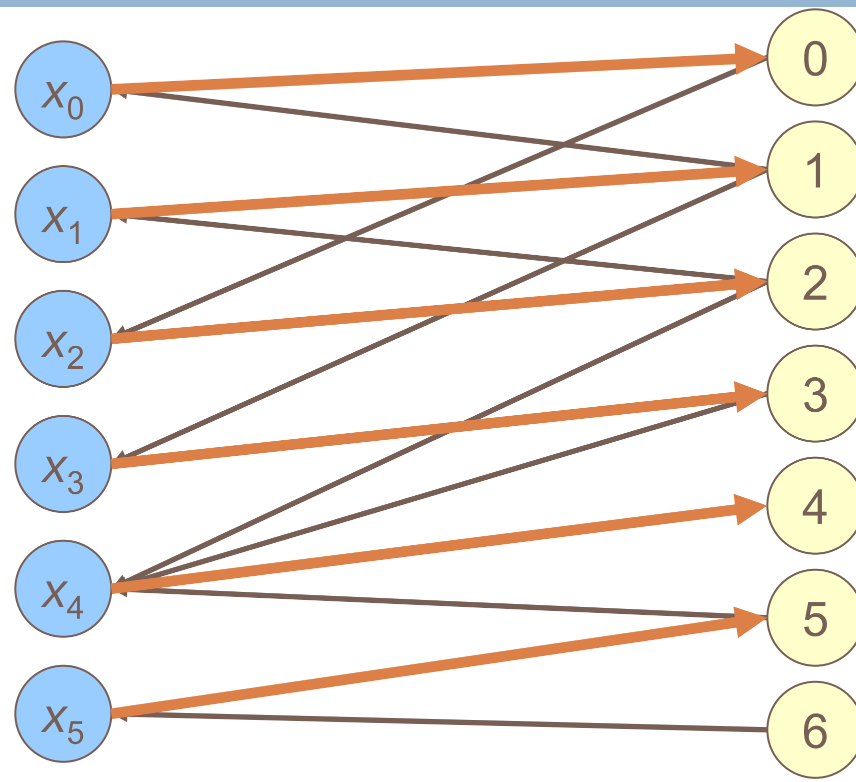
$s(x_1)=\{1,2\}$

$s(x_2)=\{0,2\}$

$s(x_3)=\{1,3\}$

$s(x_4)=\{2,3,4,5\}$

$s(x_5)=\{5,6\}$



- Compute single maximal matching
  - **matched edge** variable node → value node
  - **free edge** value node → variable node

Characterize All Solutions

$s(x_0)=\{0,1\}$

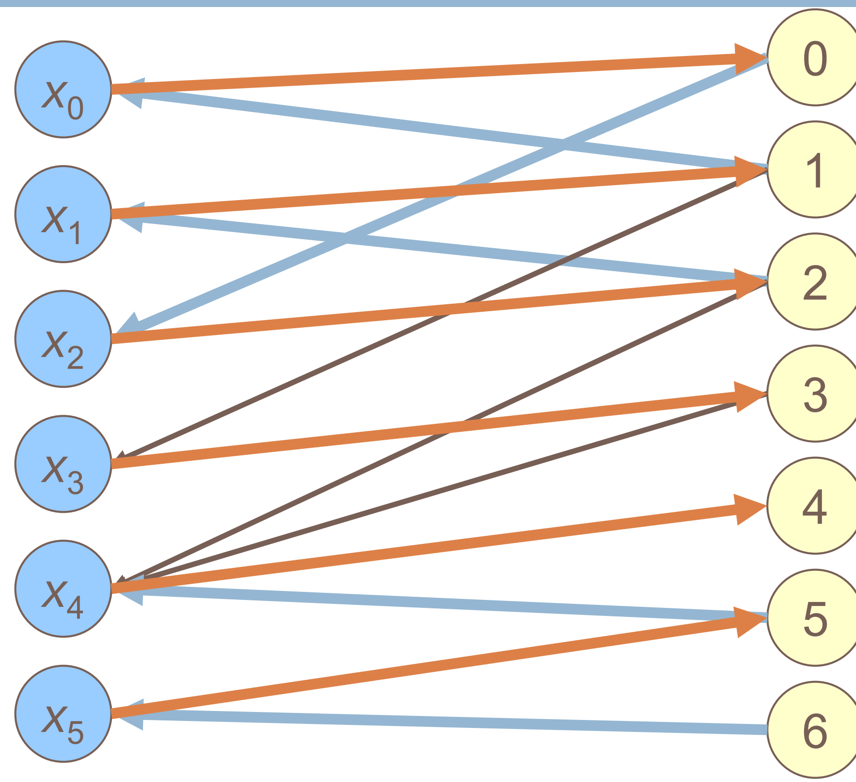
$s(x_1)=\{1,2\}$

$s(x_2)=\{0,2\}$

$s(x_3)=\{1,3\}$

$s(x_4)=\{2,3,4,5\}$

$s(x_5)=\{5,6\}$



- Edges that can appear in any matching
  - even alternating cycles $(x_0 \rightarrow 0 \rightarrow x_2 \rightarrow 2 \rightarrow x_1 \rightarrow 1 \rightarrow x_0)$
  - even alternating paths $(6 \rightarrow x_5 \rightarrow 5 \rightarrow x_4 \rightarrow 4)$

Characterize All Solutions

$s(x_0)=\{0,1\}$

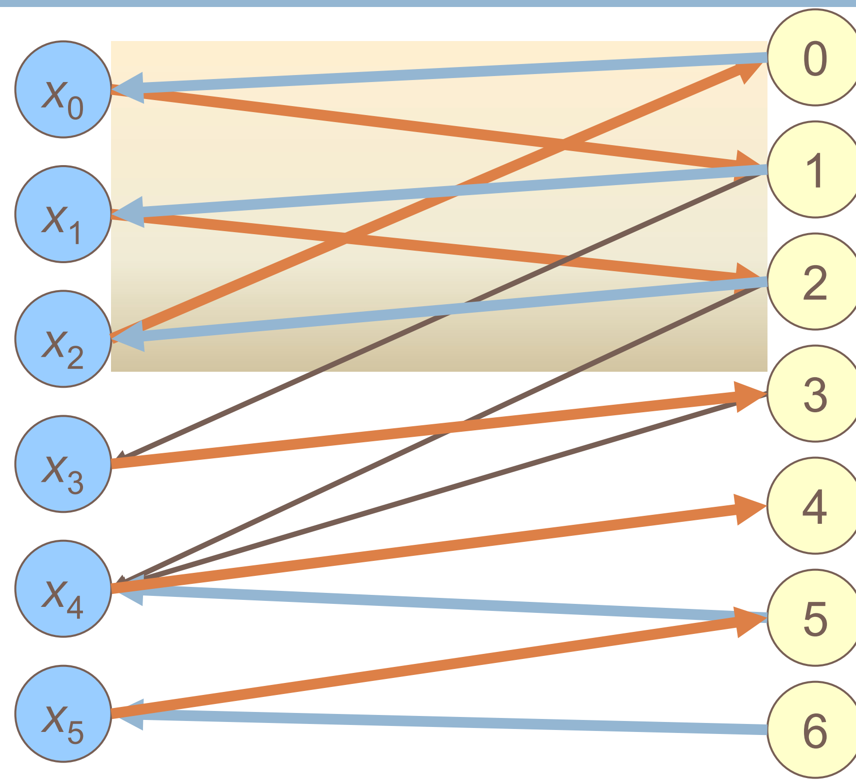
$s(x_1)=\{1,2\}$

$s(x_2)=\{0,2\}$

$s(x_3)=\{1,3\}$

$s(x_4)=\{2,3,4,5\}$

$s(x_5)=\{5,6\}$



- Edges that can appear in any matching
  - even alternating cycles $(x_0 \rightarrow 1 \rightarrow x_1 \rightarrow 2 \rightarrow x_2 \rightarrow 0 \rightarrow x_0)$
  - even alternating paths $(6 \rightarrow x_5 \rightarrow 5 \rightarrow x_4 \rightarrow 4)$

Characterize All Solutions

$s(x_0)=\{0,1\}$

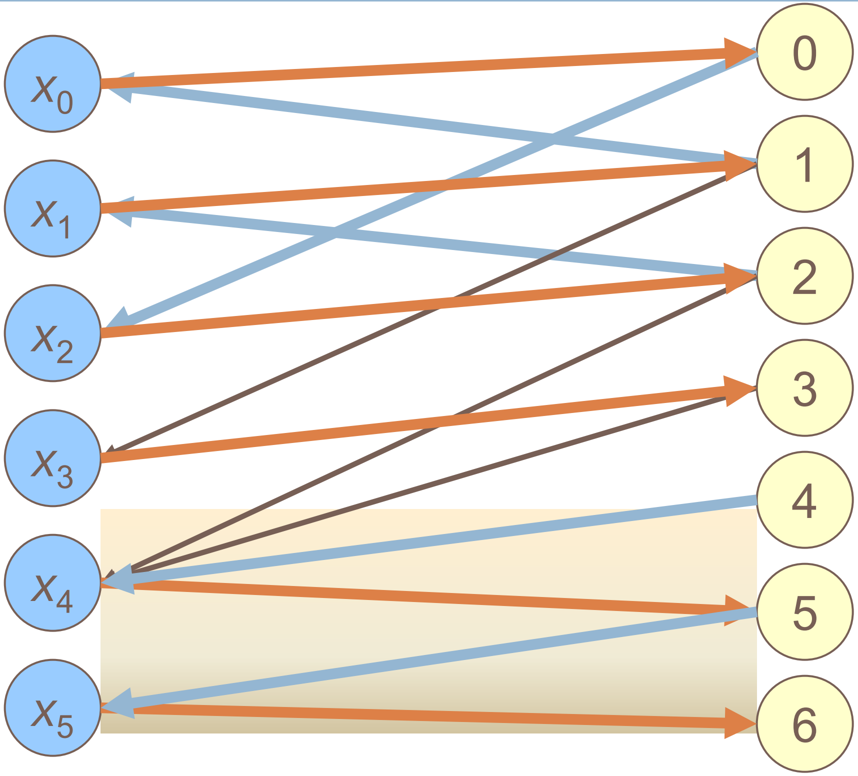
$s(x_1)=\{1,2\}$

$s(x_2)=\{0,2\}$

$s(x_3)=\{1,3\}$

$s(x_4)=\{2,3,4,5\}$

$s(x_5)=\{5,6\}$



- Edges that can appear in any matching
  - even alternating cycles $(x_0 \rightarrow 0 \rightarrow x_2 \rightarrow 2 \rightarrow x_1 \rightarrow 1 \rightarrow x_0)$
  - even alternating paths $(4 \rightarrow x_4 \rightarrow 5 \rightarrow x_5 \rightarrow 6)$

Prune Edges (Values)

$s(x_0)=\{0,1\}$

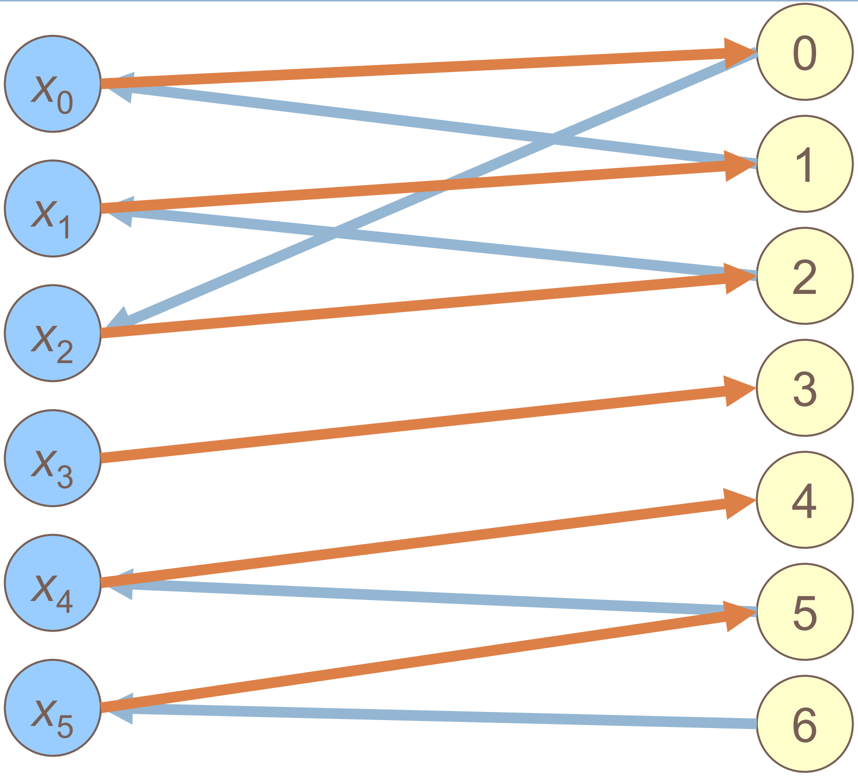
$s(x_1)=\{1,2\}$

$s(x_2)=\{0,2\}$

$s(x_3)=\{3\}$

$s(x_4)=\{4,5\}$

$s(x_5)=\{5,6\}$



- Prune edges that cannot appear in any matching
  - accordingly: prune values from variables

Global Constraints

- Reasons for globality: decomposition...
  - semantic: ...not possible
  - operational: ...less propagation
  - algorithmic: ...less efficiency

- Plethora available
  - scheduling, sequencing, cardinality, sorting, circuit, ...
  - systematic catalogue with hundreds available
    http://www.emn.fr/x-info/sdemasse/gccat/
  - sometimes not straightforward to pick the right one (strength versus efficiency, etc)

Summary

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

- Constraints as
  - reusable
  - powerful

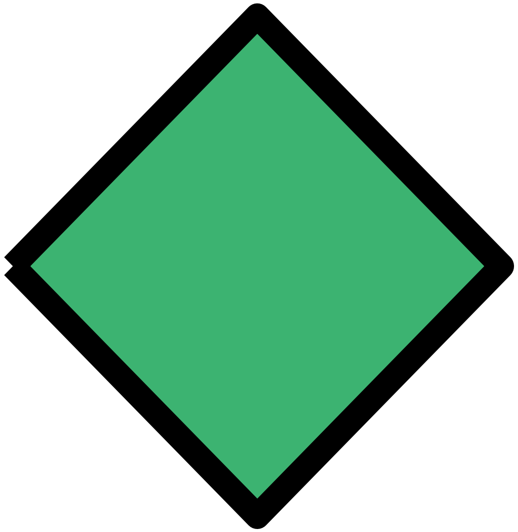
  software components

How to Deal with Distinct...

- Assume *n* variables, at most *d* values
- SAT (propositional formulae)
  - O(*nd*) clauses [Gent, Nightinggale, 2004]
  - other encodings possible
- MILP (mixed integer linear programs)
  - introduce O(*nd*) new 0/1 variables
  - decompose into O(*n*+*d*) linear (in)equations

    [Hooker, 2007, p 368]

SMM: Strong Propagation



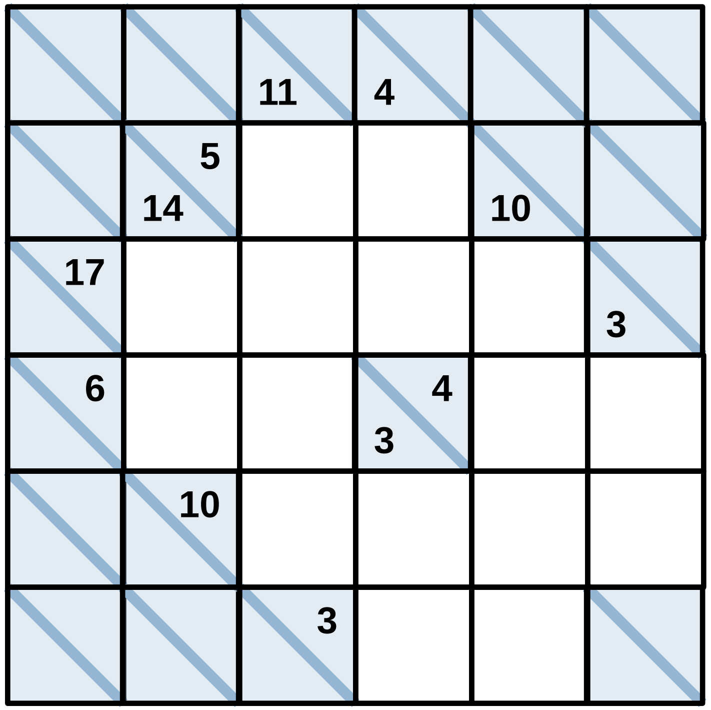
```
    SEND
+   MORE
--------
=  MONEY

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

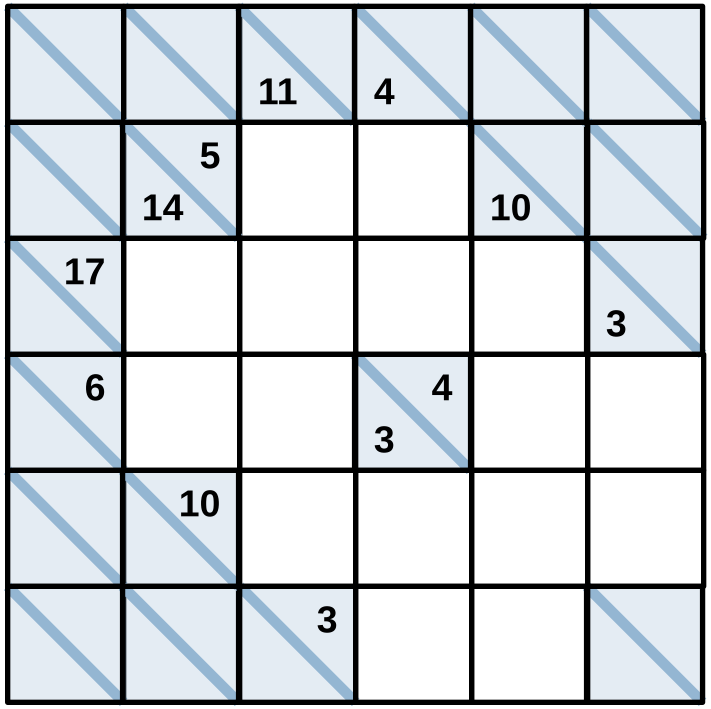
Local Reasoning

beauty and curse of
constraint programming

Kakuro

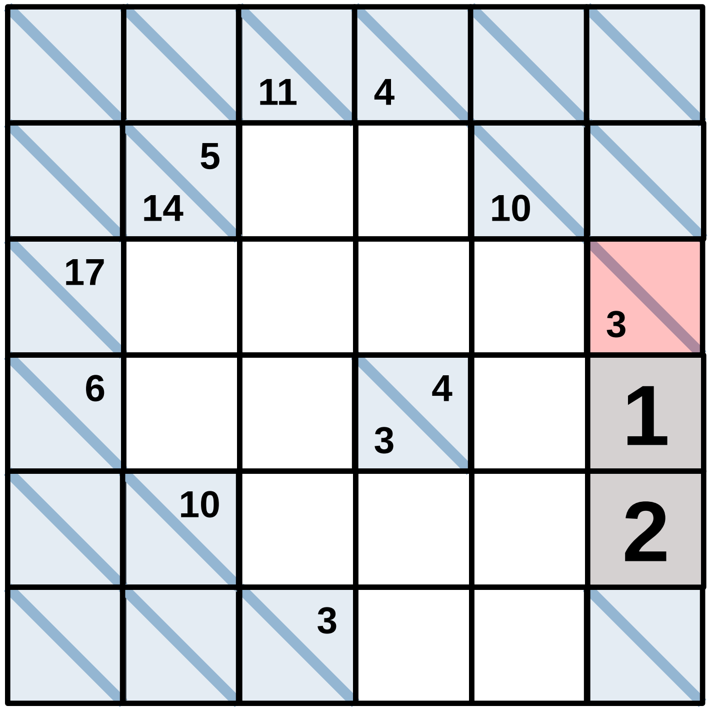


Kakuro



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

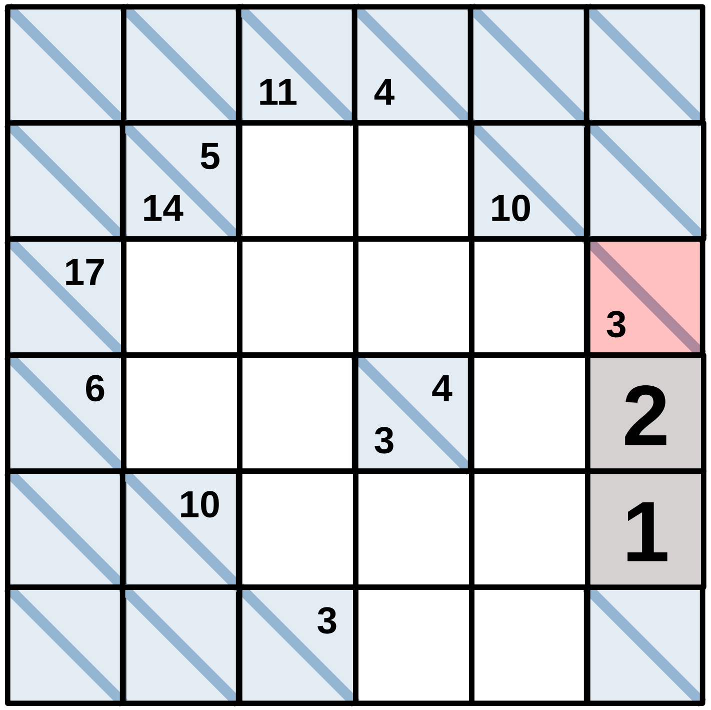
Kakuro



- For hint 3

  1 + 2

Kakuro



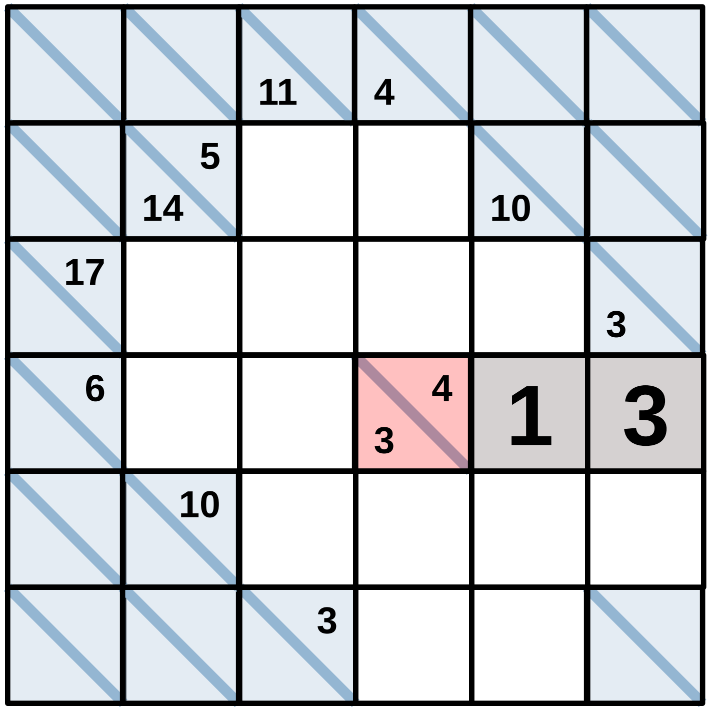
- For hint 3

  1 + 2

  or

  2 + 1

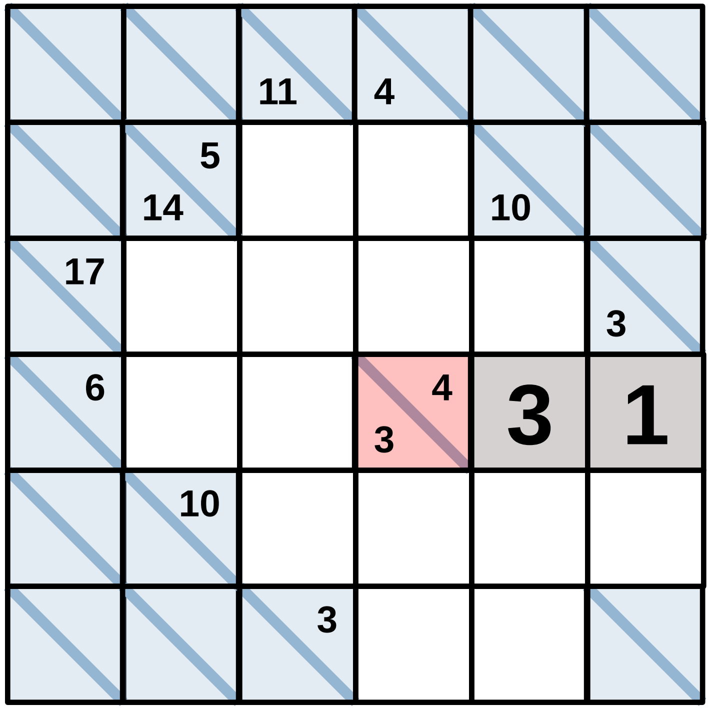
Kakuro



- For hint 4

  1 + 3

Kakuro



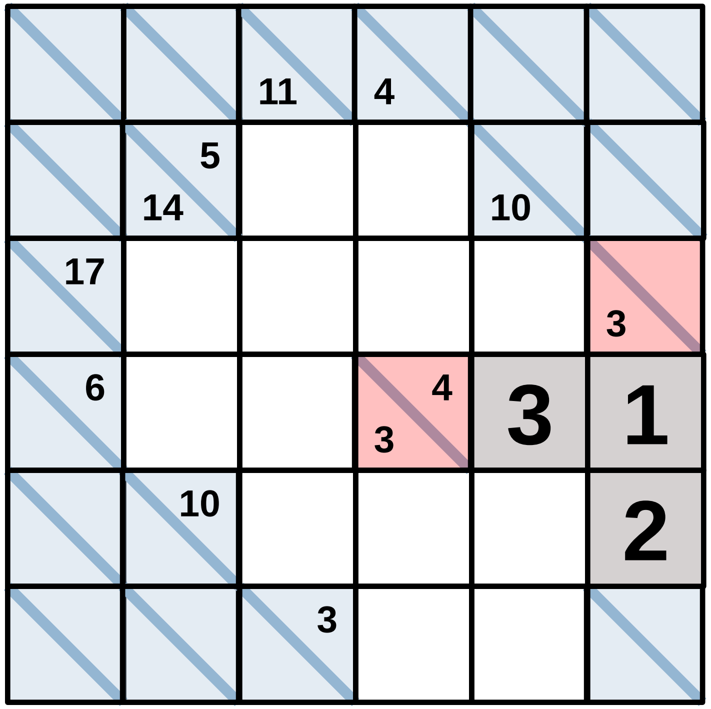
- For hint 4

  1 + 3

  or

  3 + 1

Kakuro



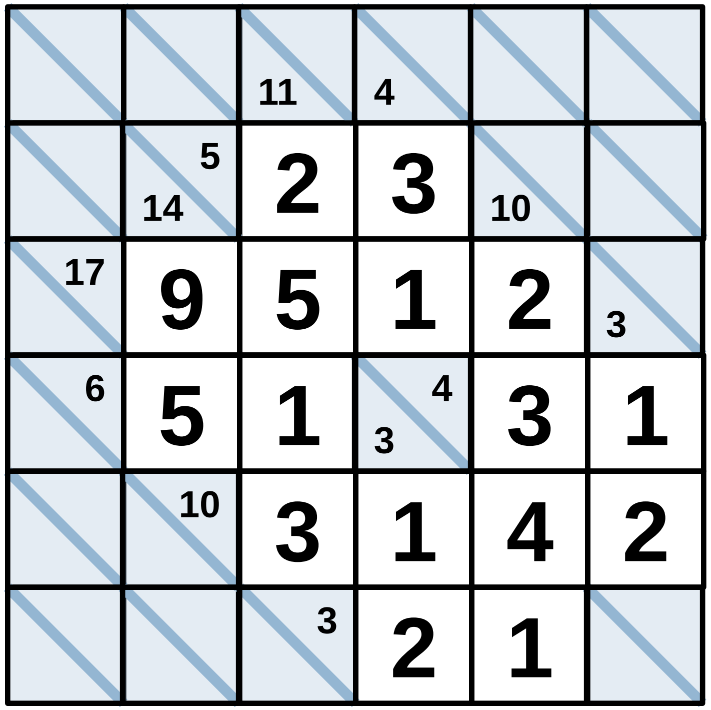
- For hint 3

  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, ? years

    years? centuries? eons?

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

workforce rostering

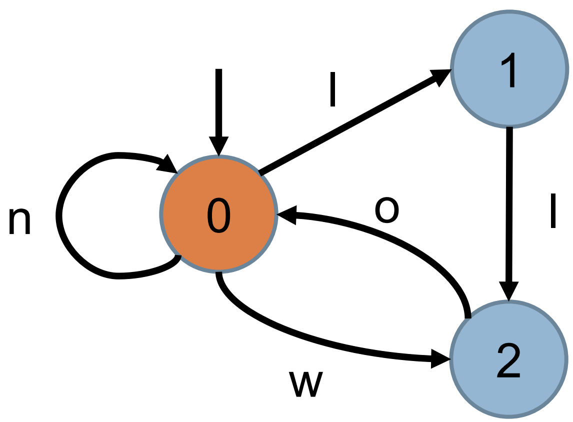
Kakuro reconsidered

Modeling Rostering: User-defined

- Personel rostering: example
  - one day off (o) after weekend shift (w)
  - one day off (o) after two consectuive 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)

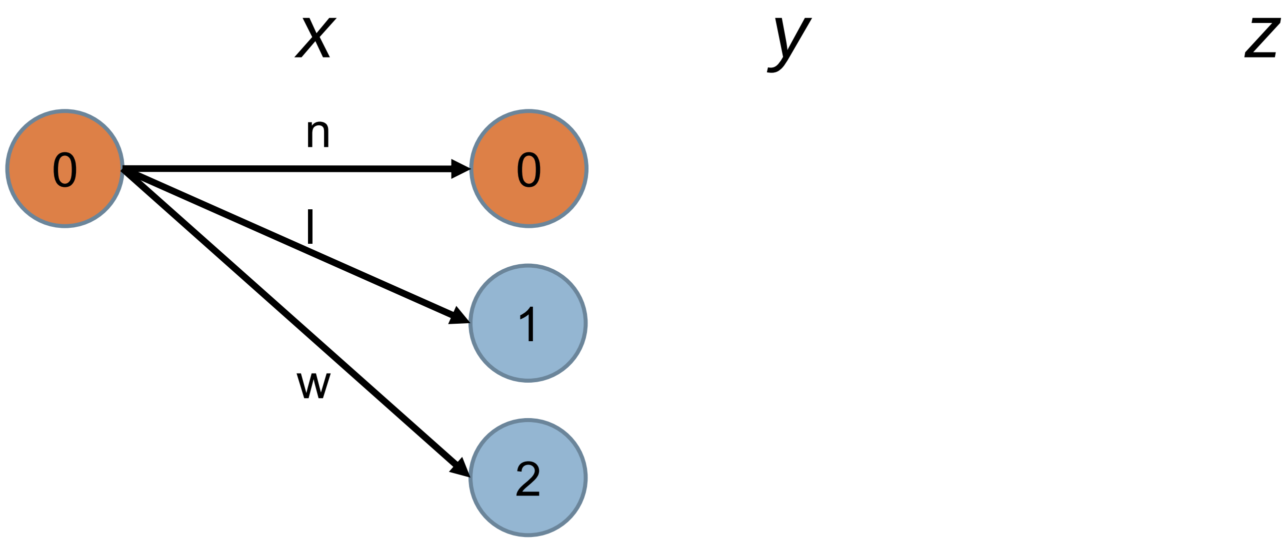
- $x_1 \dots x_n$ word in r
- or, accepted by DFA d for r

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

Propagating Regular

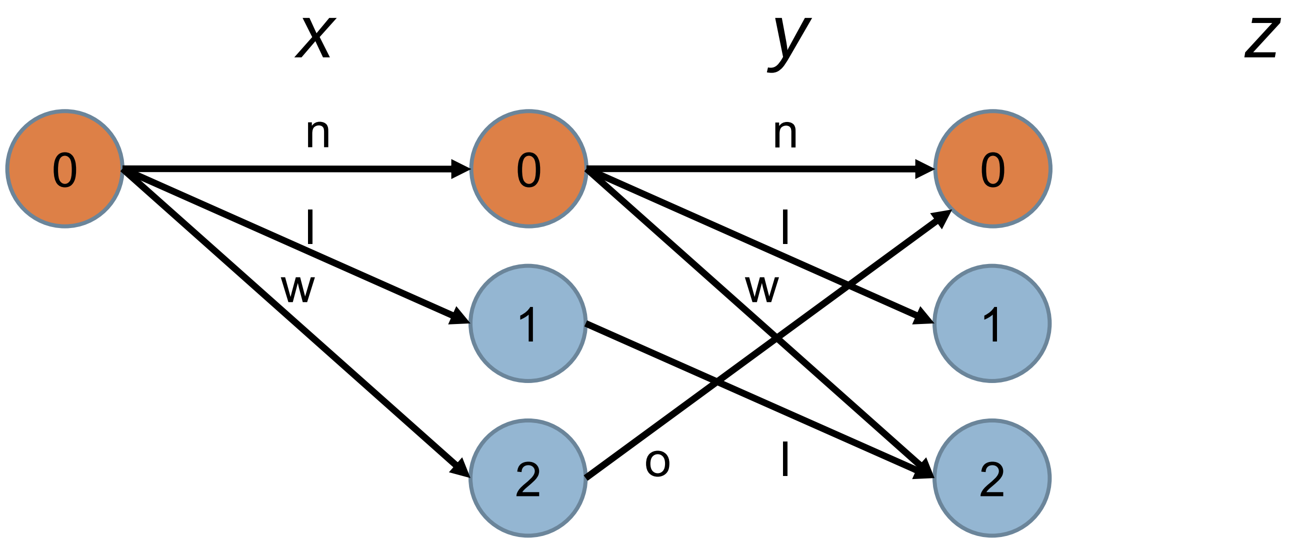
- Example: regular(x, y, z, d)
  - x, y, z in {w,o,l,n}
  - in reality: w=0, o=1, l=2, n=3

Propagating Regular



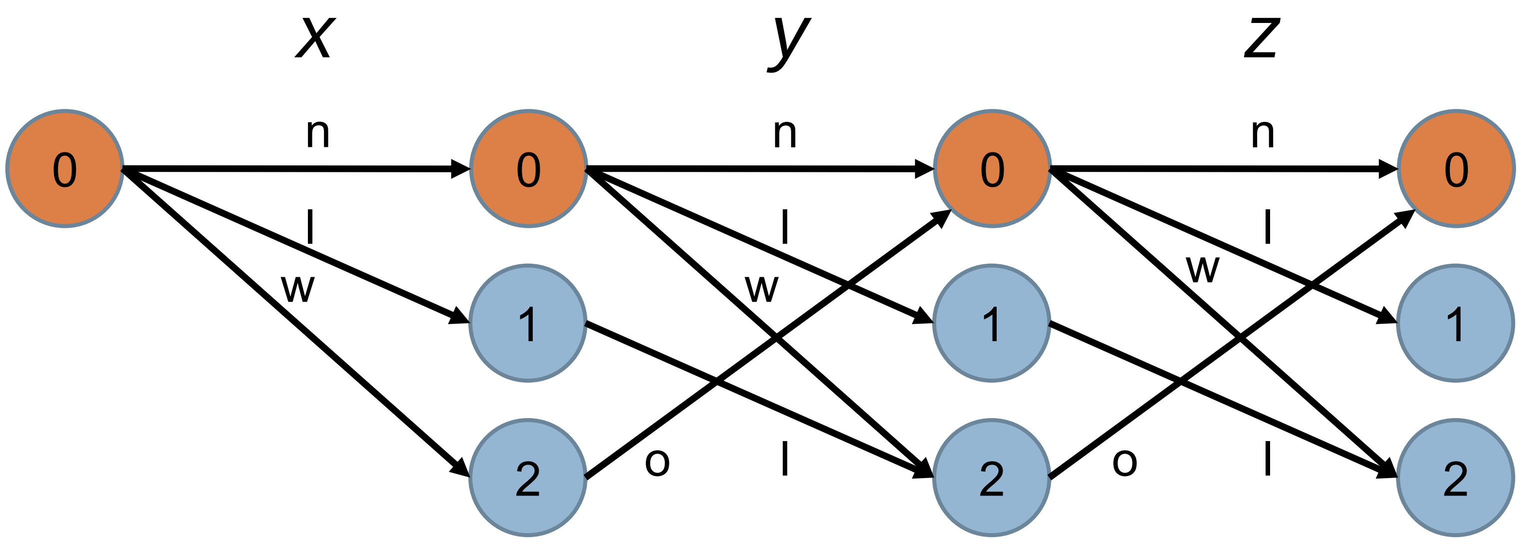
- Forward pass
  - all paths from start state

Propagating Regular



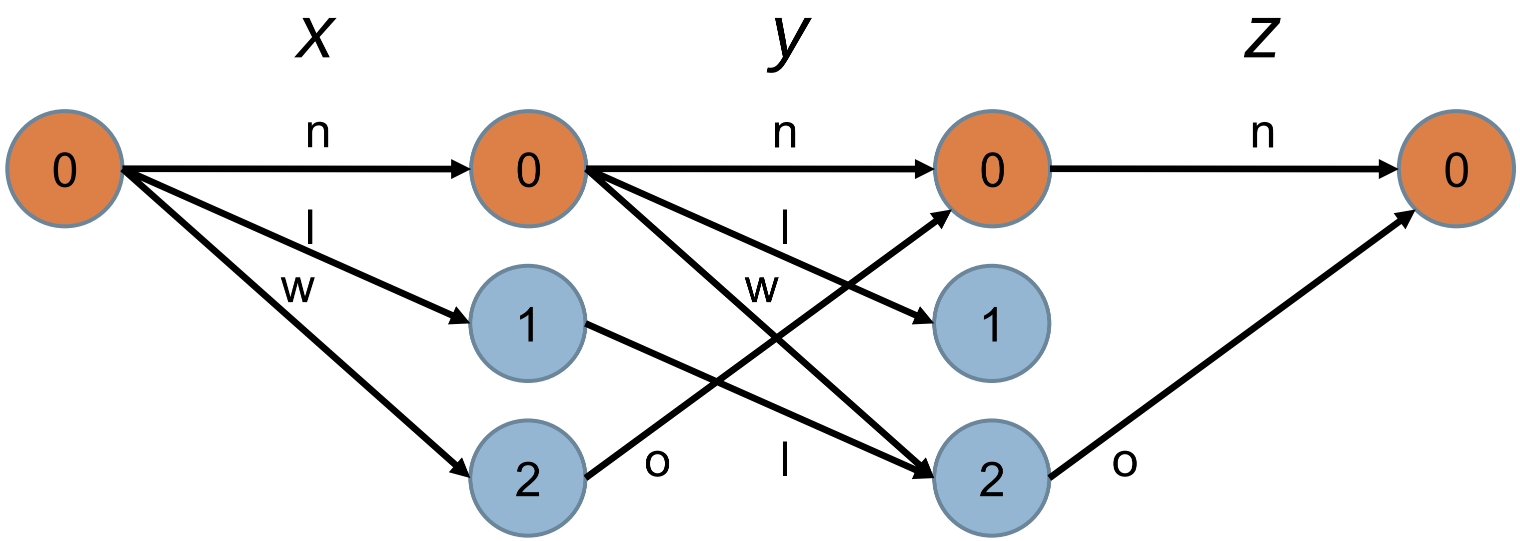
- Forward pass: optimization
  - each state at most once for each variable (“layer”)
  - several incoming/outgoing edges per state

Propagating Regular



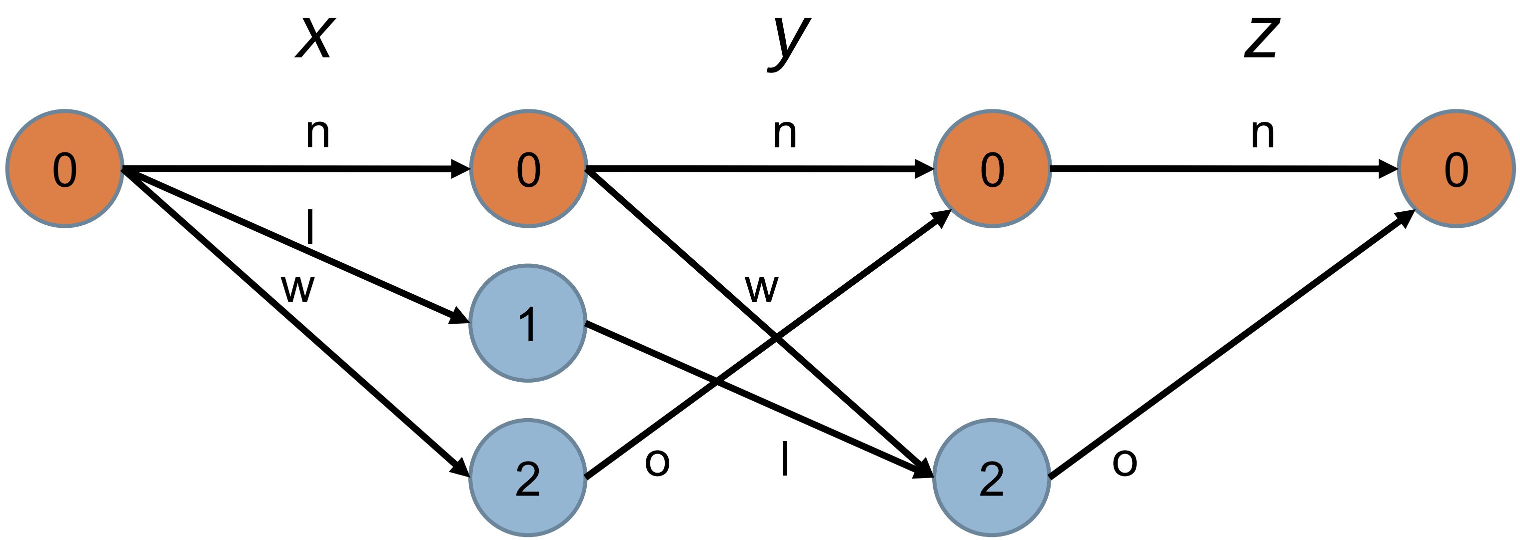
- Forward pass finished

Propagating Regular



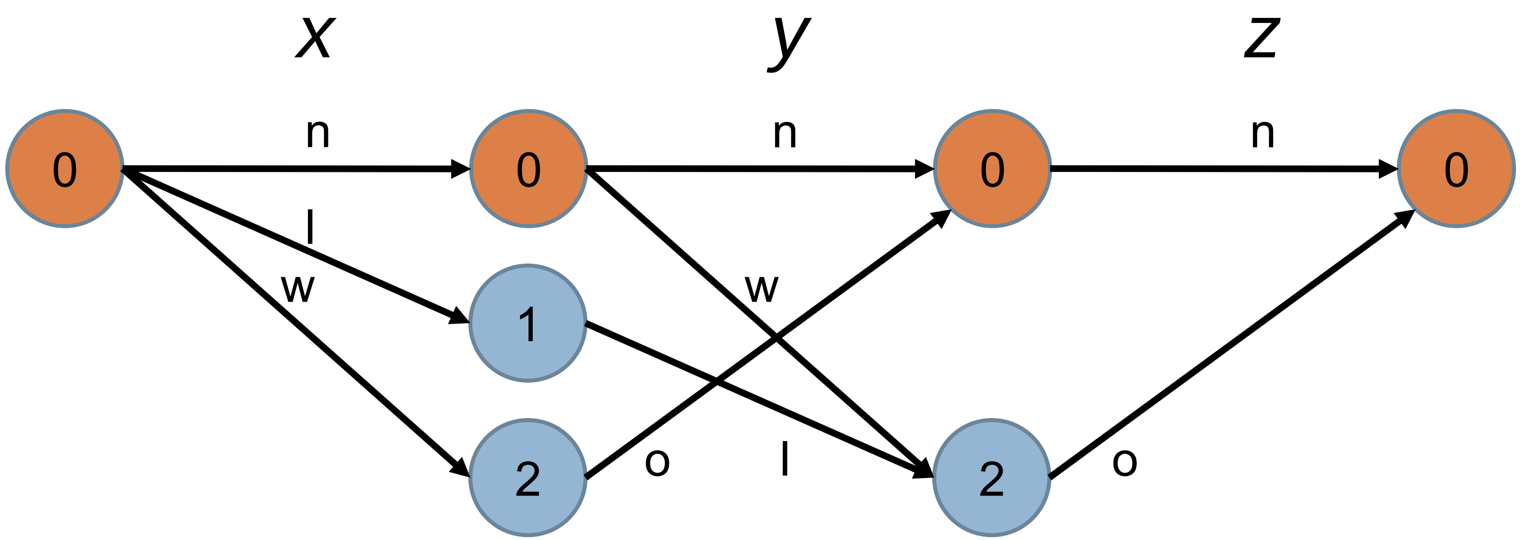
- Backward pass
  - start: remove non-final states for last layer

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]

AI's Legacy

- Original model for constraint propagation
  - constraints used for propagation in extension (list of solutions): no propagators
  - single algorithm for all constraints (consistency)
  - often restricted to binary constraints
- Beautiful model
  - insightful for understanding propagation
  - rich connections (complexity, relational databases, …)
  - rich notion of levels of pruning: arc consistency, path consistency, *k*-consistency, …

AI's Legacy: Solving for Real?

- Constraints used for propagation in extension
  - unable to exploit structure for efficient solving
  - unrealistic for large arity: distinct with *n* variables has *n*! solutions, ….
- Single algorithm for all constraints
  - infeasible in general: constraints may be NP-hard
  - no compromise between pruning and efficiency
- Often restricted to binary constraints
  - decomposition destroys propagation

The Best of Both Worlds

- Start from propagator-based constraint propagation
  - take advantage of dedicated algorithms
- Dedicated propagator for user-defined constraints
  - only pay, if needed
  - incredibly efficient: MDD-based propagator [Yap ea, 2008]

Kakuro Reconsidered

- Real model: for each hint
  - one regular constraint combining distinct and sum
  - 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
  - use specialized propagator rather than extensional framework
- Kakuro: decomposition is harmful [again]
  - capture essential structure by few constraints
  - best by single constraint

Compositional Modeling

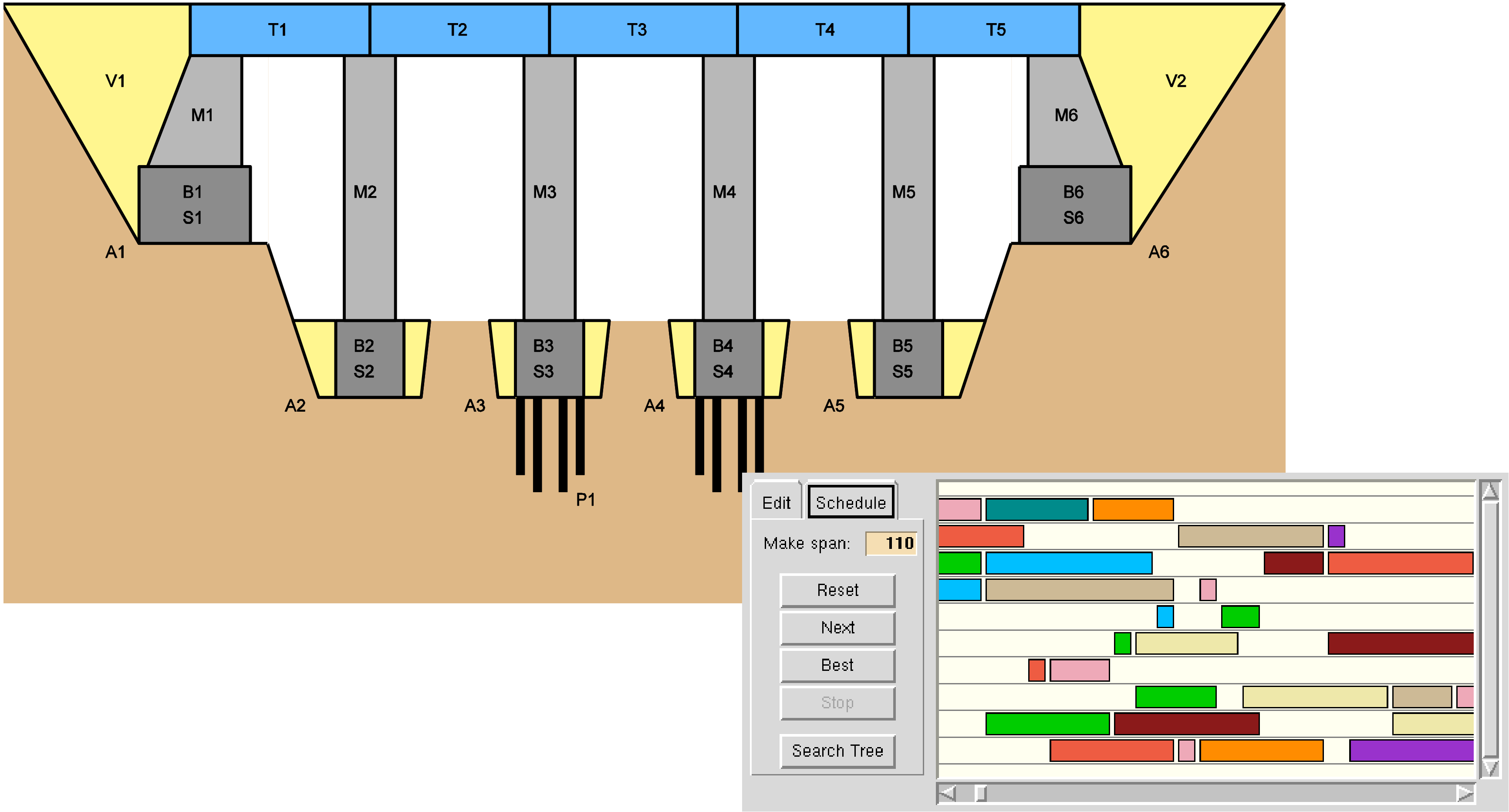
scheduling resources

Scheduling Resources: Problem

- Tasks
  - duration
  - resource
- Precedence constraints
  - determine order among two tasks
- Resource constraints
  - at most one task per resource

    [disjunctive, non-preemptive scheduling]

Scheduling: Bridge Example



Scheduling: Solution

- Start time for each task
- All constraints satisfied
- Earliest completion time
  - minimal make-span

Scheduling: Model

- Variable for start-time of task *a*

  start(*a*)

- Precedence constraint: *a* before *b*

  start(*a*) + dur(*a*) ≤ start(*b*)

Scheduling: Model

- Variable for start-time of task *a*

  start(*a*)

- Precedence constraint: *a* before *b*

  start(*a*) + dur(*a*) ≤ start(*b*)

- Resource constraint:

  a before *b*

  or

  b before *a*

Scheduling: Model

- Variable for start-time of task *a*

  start(*a*)

- Precedence constraint: *a* before *b*

  start(*a*) + dur(*a*) ≤ start(*b*)

- Resource constraint:

  start(*a*) + dur(*a*) ≤ start(*b*)

  or

  b before *a*

Scheduling: Model

- Variable for start-time of task *a*

  $\text{start}(a)$

- Precedence constraint: *a* before *b*

  $\text{start}(a) + \text{dur}(a) \leq \text{start}(b)$

- Resource constraint:

  $\text{start}(a) + \text{dur}(a) \leq \text{start}(b)$

  or

  $\text{start}(b) + \text{dur}(b) \leq \text{start}(a)$

  [use so-called reification for this]

Model: Easy But Too Naive

- Local view
  - individual task pairs
  - $O(n^2)$ propagators for n tasks
- Global view (again a global constraint)
  - all tasks on resource
  - single propagator
  - smarter algorithms possible

Edge Finding: Idea

- Assume a subset O of tasks and a task $t \in O$
  - compute earliest completion time of O

    ect(O)
  - compute latest completion time of O - {t}

    lct(O - {t})
  - if

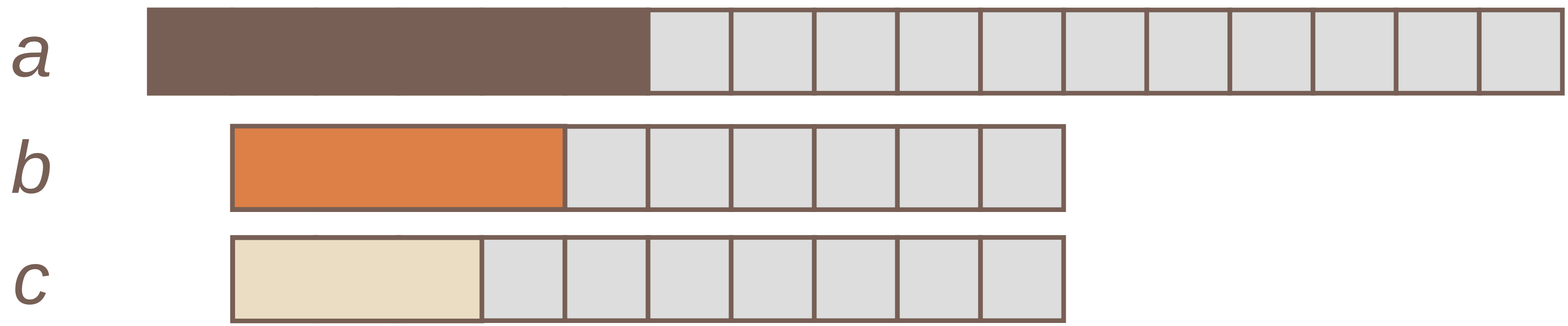
    ect(O) > lct(O - {t})

    then

    t must run last in O
- Can be done in O(n log n) for n tasks

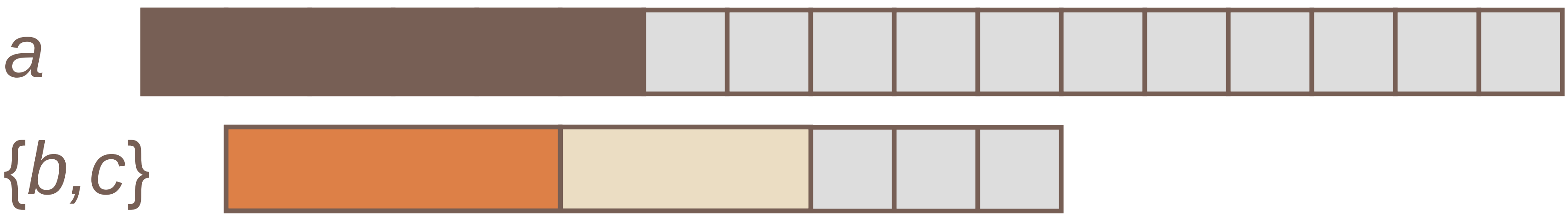
  [Carlier & Pinson, 1994] [Vilím ea., 2004]

Edge Finding



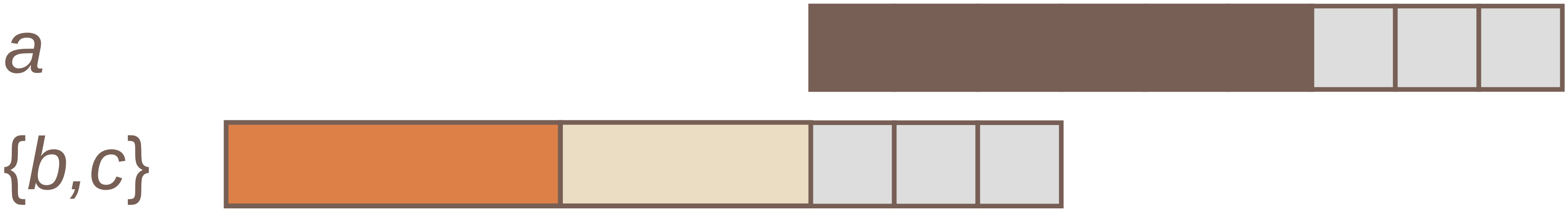
- Assume
  - start(a) ∈ {0,…,11} dur(a) = 6
  - start(b) ∈ {1,…,7} dur(b) = 4
  - start(c) ∈ {1,…,8} dur(c) = 3

Edge Finding



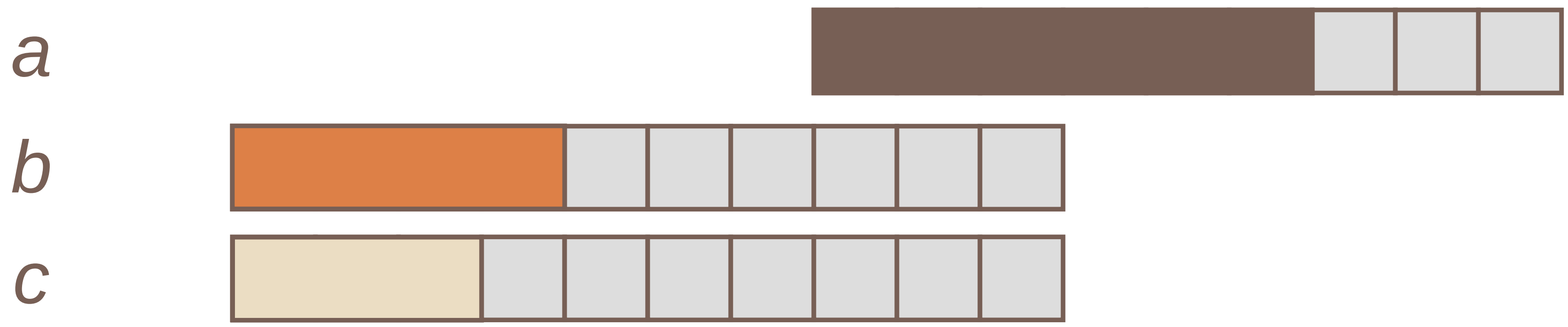
- Assume $O=\{a,b,c\}$, $t=a$
- Clearly, *a* must go last

Edge Finding



- Assume $O=\{a,b,c\}$, $t=a$
- Clearly, *a* must go last

Edge Finding



- Propagate
  - start(a) $\in$ {8,…,11}

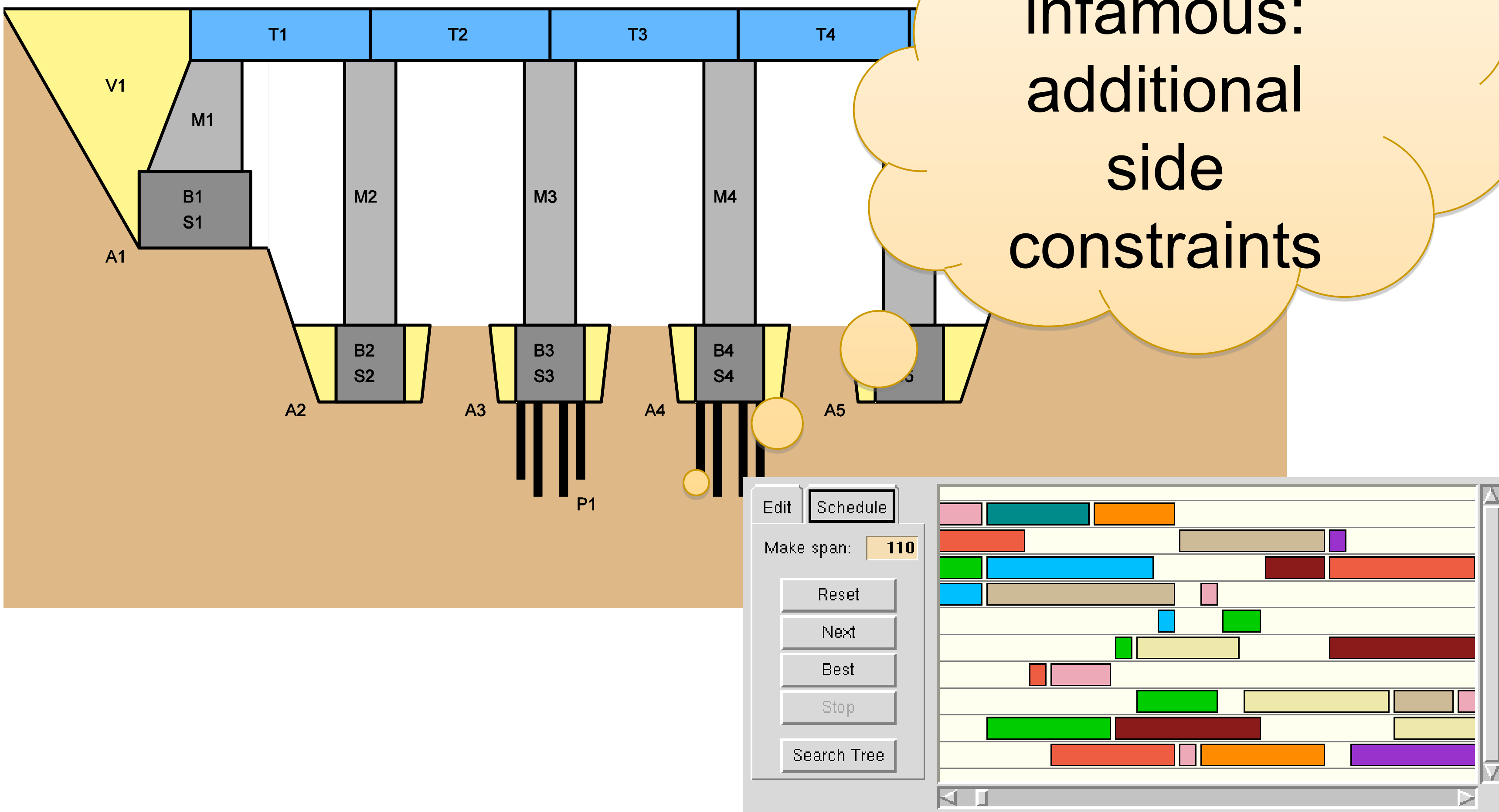
Constraint-based Scheduling

- Rich set of methods
  - propagation
  - branching heuristics
  - search methods

- Many variants
  - disjunctive, cumulative, elastic, preemptive, ...
  - batch processing, setup times, ...

  [Baptiste, Le Pape, Nuijten, Constraint-based Scheduling. Kluwer, 2001]

Scheduling: Bridge Example



infamous: additional side constraints

Summary

- Modeling is compositional
  - reasoning is too!
- Powerful global constraints...
  plus...
  essential additional side constraints
- Scheduling domain
  - show case of constraint programming

Strength And Challenges

Strength

- Captures structure
  - use structure for efficient reasoning
  - unique distinction from SAT and LP
- Flexible, compositional, reusable
  - add additional side constraints
  - add new algorithmic components
  - high return on investment into global constraints
- Simple
  - clear model based on propagators
- Efficient systems available
  - commercial and open source

Challenges

- Modeling: art not science
  - true to some extent for most approaches
  - here: identify substructures, know strength of different methods
  - array of techniques: symmetry breaking, implied constraints, heuristics, ...
- Search: mostly naive
  - local decision making
  - no global techniques such as learning (SAT), or strong branching, impact-based search (LP)
  - remedies in their infancy

The Essence

- Constraint programming is about…

  ...local reasoning exploiting structure
- Strength
  - simplicity, compositionality, exploiting structure
- Challenges
  - lack of global picture during search
  - difficult to find global picture due to rich structure
- Future
  - part of hybrid solutions

Resources

- Complete and recent 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.