Constraint Programming

Introduction, State of the Art & Trends

Christian Schulte
cschulte@kth.se

Department of Electronic, Computer and Software Systems
School of Information and Communication Technology
KTH – Royal Institute of Technology
Sweden

KTH Information and Communication Technology

Talk Overview

- What is Constraint Programming?

Sudoku is Constraint Programming

- ... more 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

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			

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

Propagation

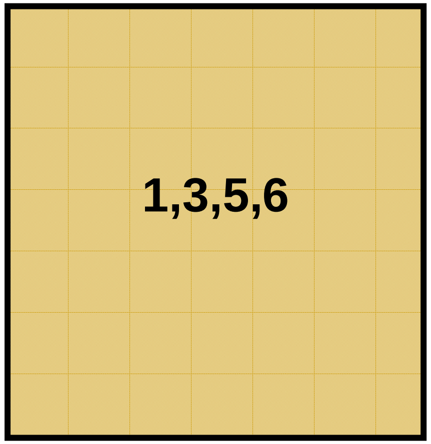
			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
			6		8			

1,3,5,6,7

- 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 variables

- Modelling: 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, ...
- Modelling
  - variables, values, constraints, heuristics, symmetries, ...

Remainder Overview

- Key ideas and principles
  - constraint propagation
  - search: branching and exploration
- Why does constraint programming matter
- State of the art and trends

- Excursions
  - constraint propagation revisited
  - scheduling resources
  - strong propagation

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 $\in$ {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

Important Concepts

- Constraint store
- Propagator
- Constraint propagation

Constraint Store

$x \in \{3,4,5\}$ $y \in \{3,4,5\}$

- Maps variables to possible values

Constraint Store

finite domain constraints

$x \in \{3,4,5\}$ $y \in \{3,4,5\}$

- Maps variables to possible values
- Others: finite sets, intervals, trees, ...

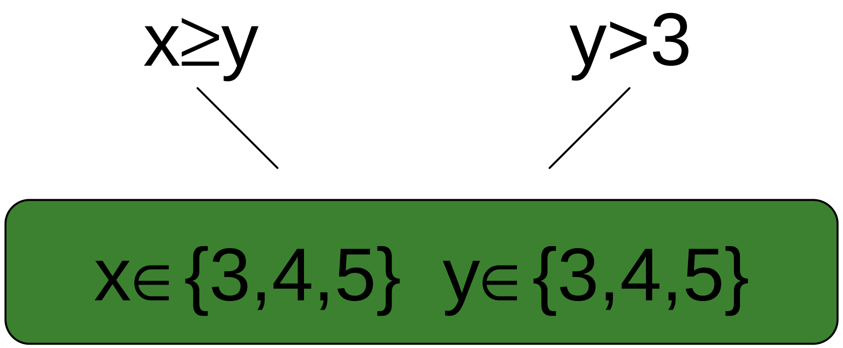
Propagators

- Implement constraints

  $\texttt{distinct}(x_1,\ldots,x_n)$

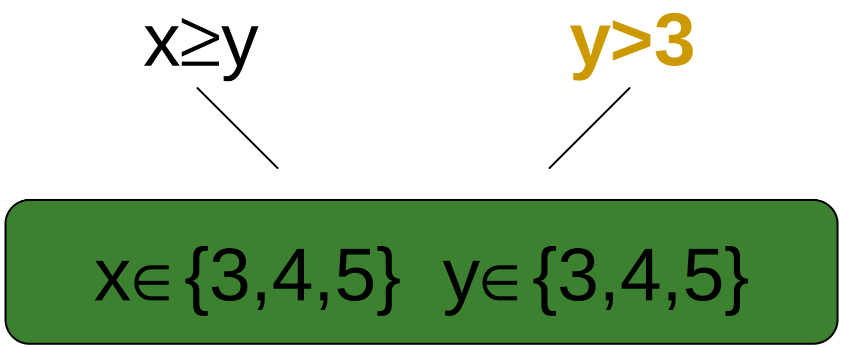
  $x + 2\times y = z$

Propagators



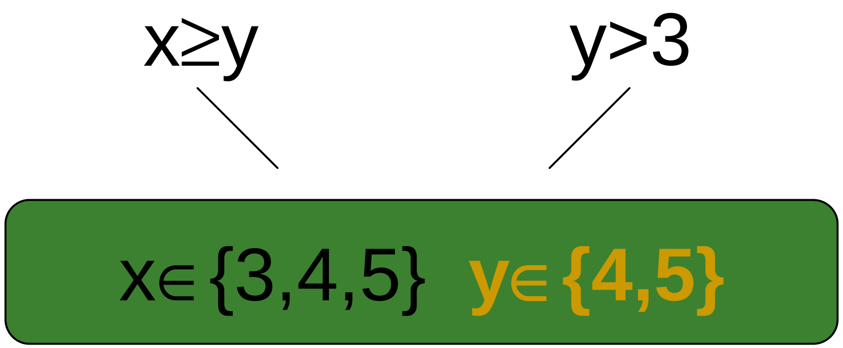
- Amplify store by constraint propagation

Propagators



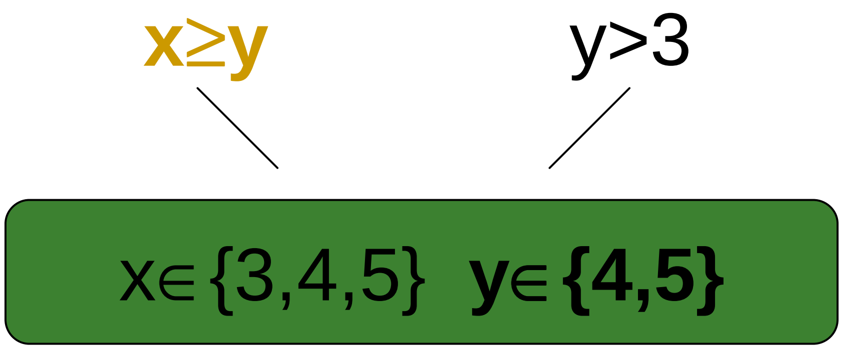
- Amplify store by constraint propagation

Propagators



- Amplify store by constraint propagation

Propagators



- Amplify store by constraint propagation

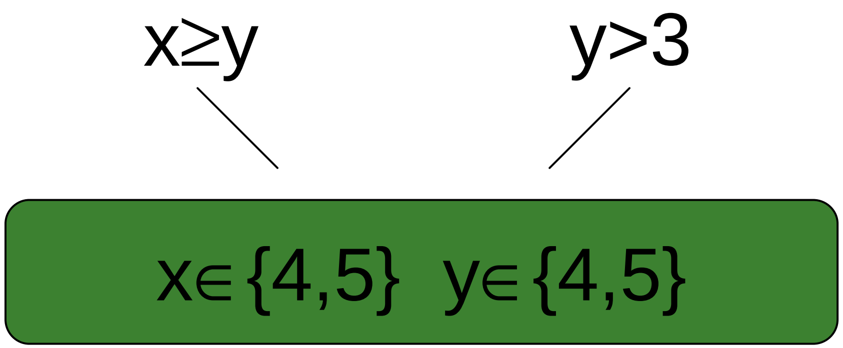
Propagators

$x \geq y$ $\quad$ $y > 3$

$x \in \{4,5\}$ $\quad$ $y \in \{4,5\}$

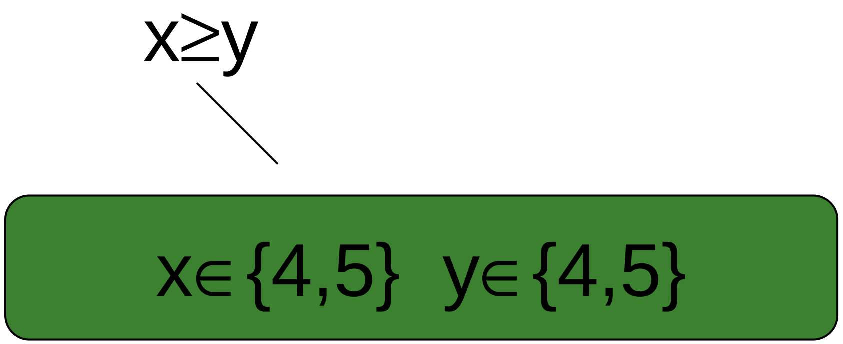
- Amplify store by constraint propagation

Propagators



- Amplify store by constraint propagation
- Disappear when done (subsumed, entailed)
    - no more propagation possible

Propagators



- Amplify store by constraint propagation
- Disappear when done (subsumed, entailed)
  - no more propagation 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}

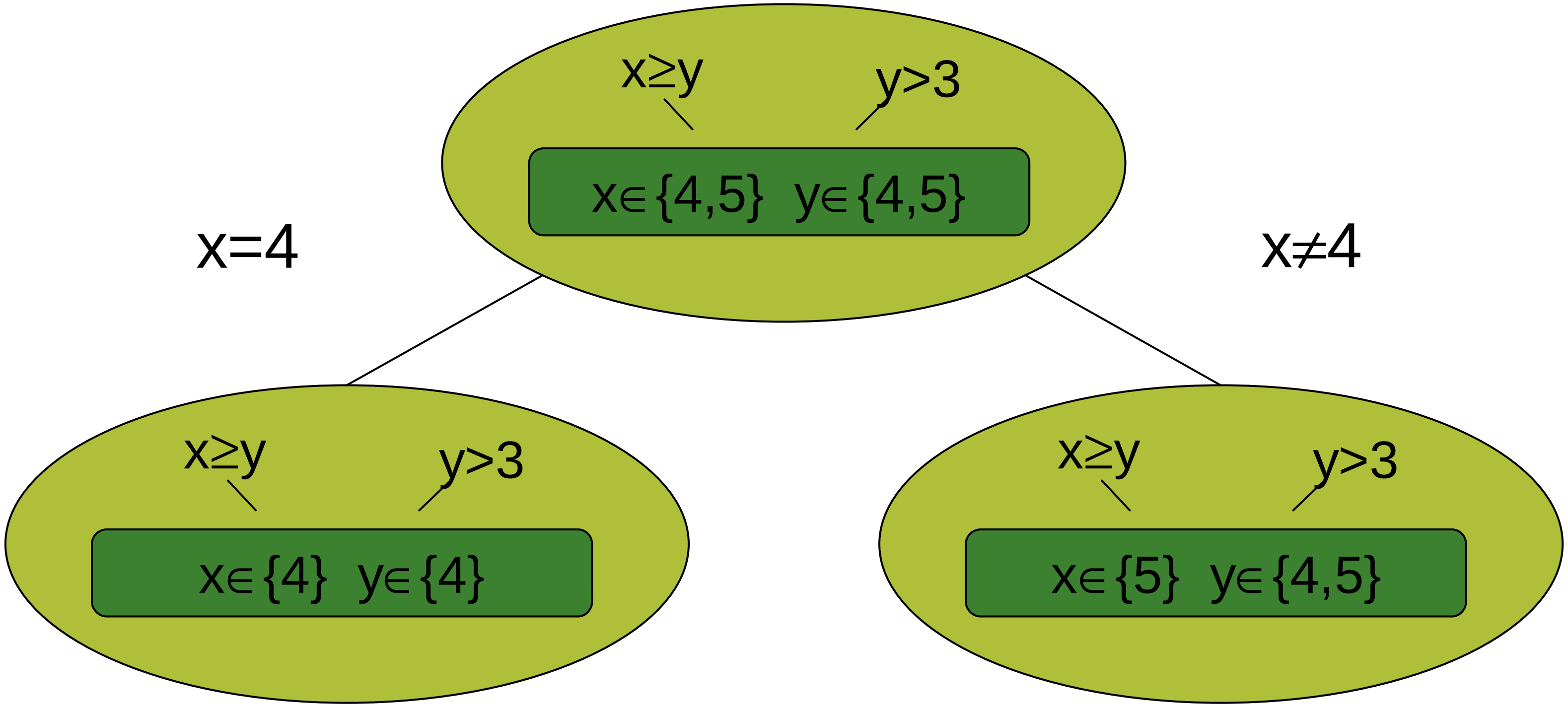
- Propagation **alone** not sufficient!
  - create simpler sub-problems
  - branching

Search

Important Concepts

- Branching
- Exploration
- Branching heuristics
- Best solution search

Search: Branching



- Create subproblems with additional information
  - enable further constraint propagation

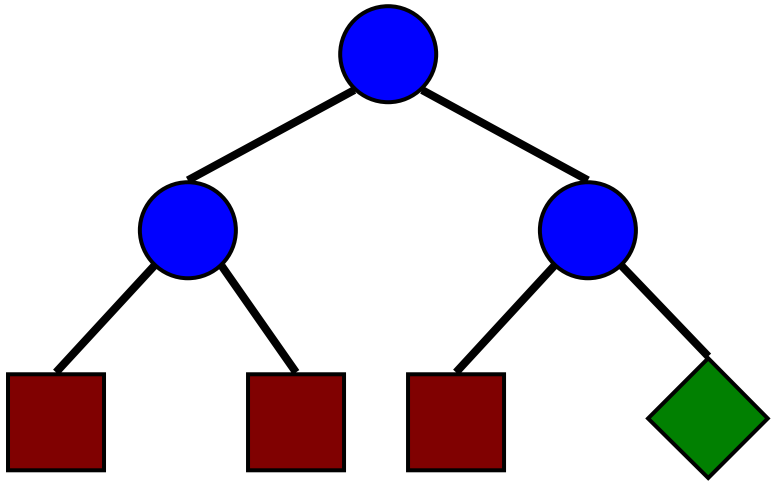
Example Branching Strategy

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

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

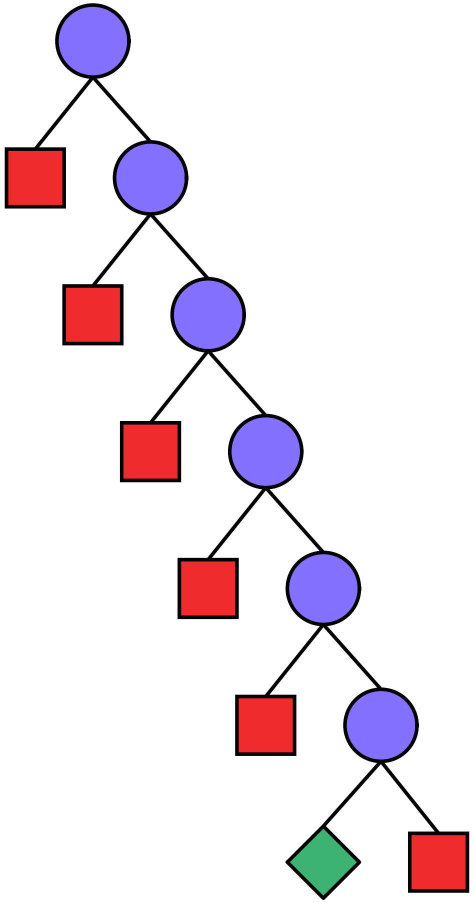
- Part of model

Search: Exploration



- Iterate propagation and branching
- Orthogonal: branching ⇆ exploration
- Nodes:
  - **Unsolved**
  - **Failed**
  - **Succeeded**

SMM: Solution



```
    SEND
+   MORE
= MONEY

    9567
+   1085
=  10652
```

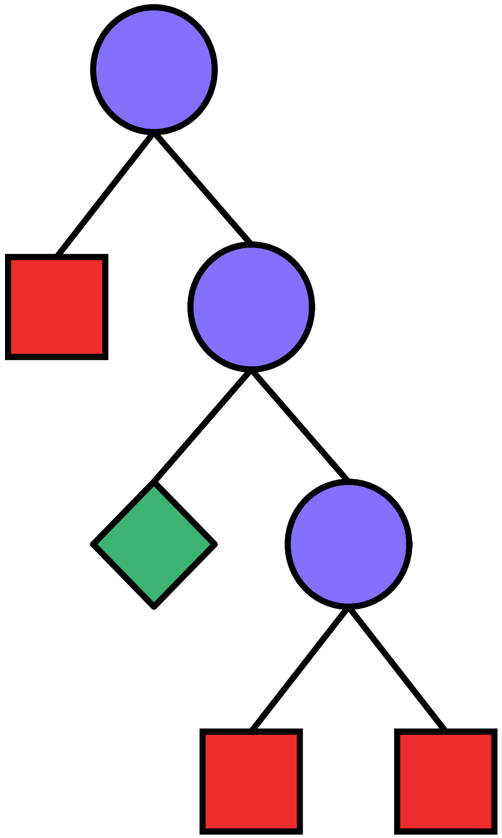
Heuristics for Branching

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

    `x=m` or `x≠m`
  - split with median m

    `x<m` or `x≥m`
- Problem specific

SMM: Solution With First-fail



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

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

Send Most Money (SMM++)

- Find distinct digits for letters, such that

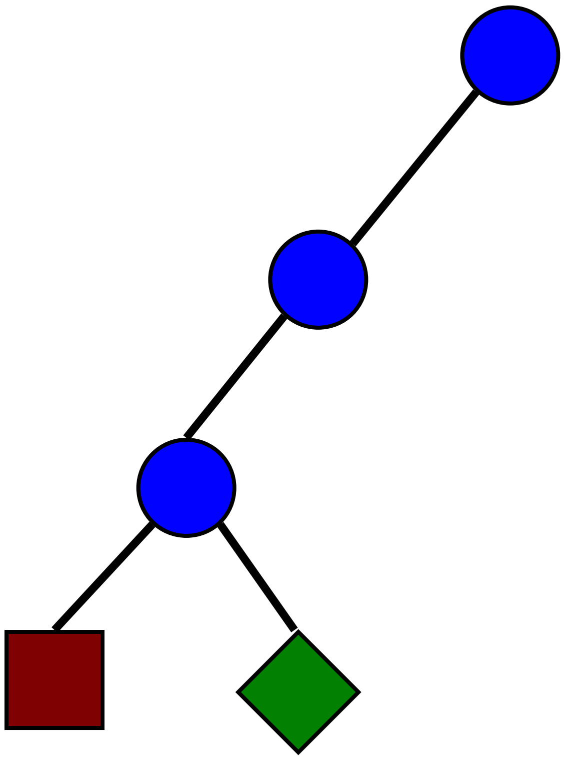
```
      SEND
+     MOST
= MONEY
```

and **MONEY** maximal

Best Solution Search

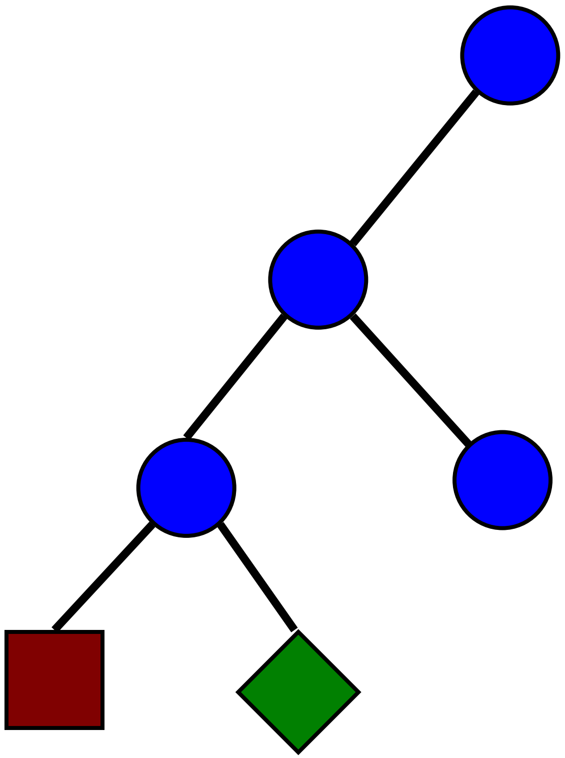
- Naïve approach:
  - compute all solutions
  - choose best
- Branch-and-bound approach:
  - compute first solution
  - add “betterness” constraint to open nodes
  - next solution will be “better”
  - prunes search space

Branch-and-bound Search



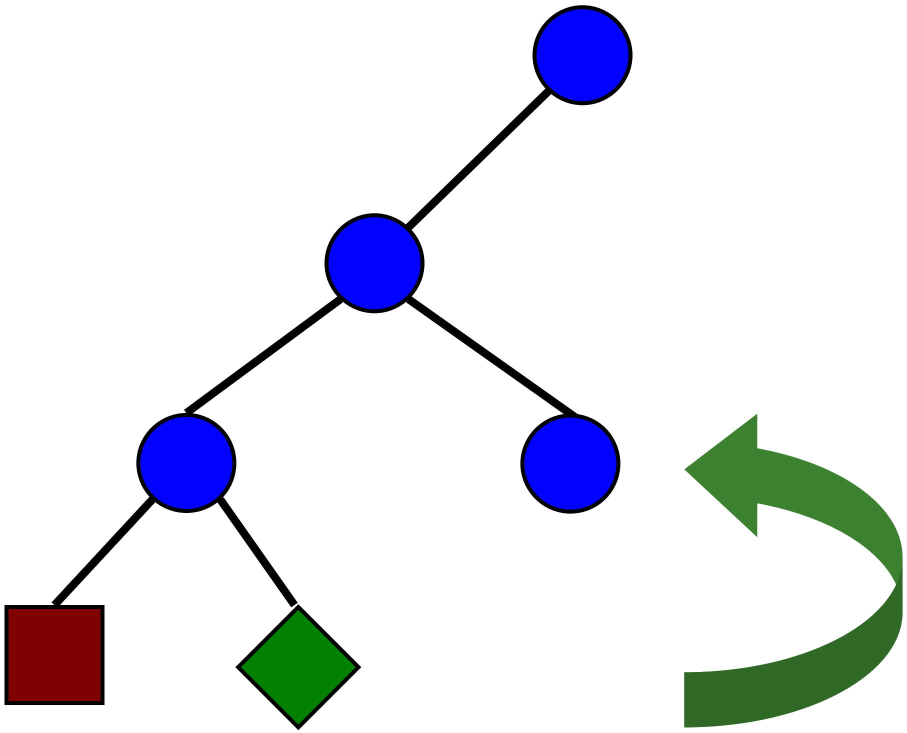
- Find first solution

Branch-and-bound Search



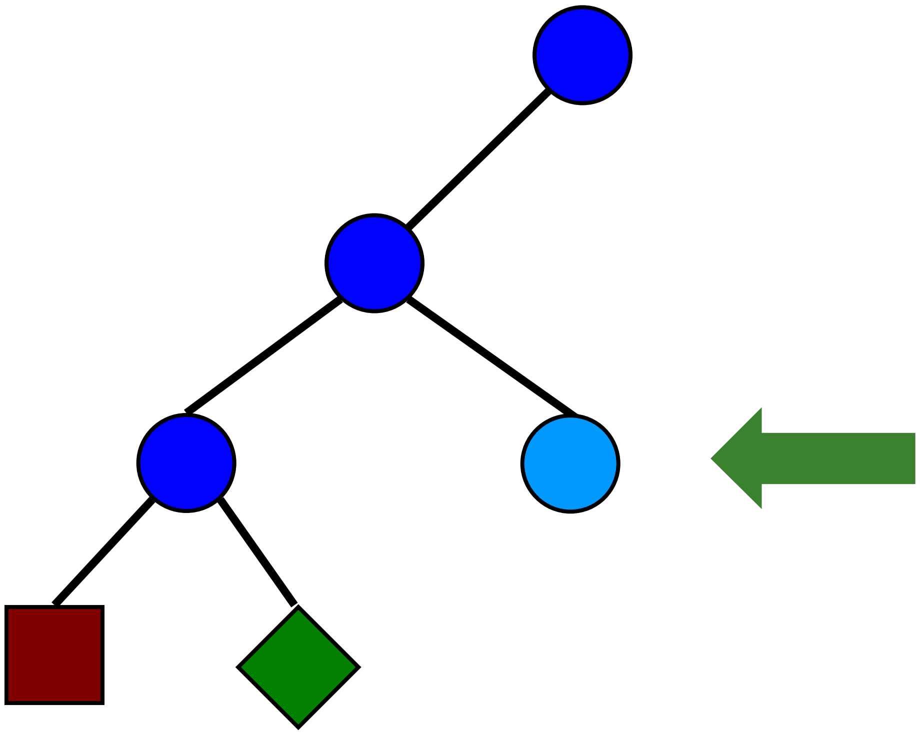
- Explore with additional constraint

Branch-and-bound Search



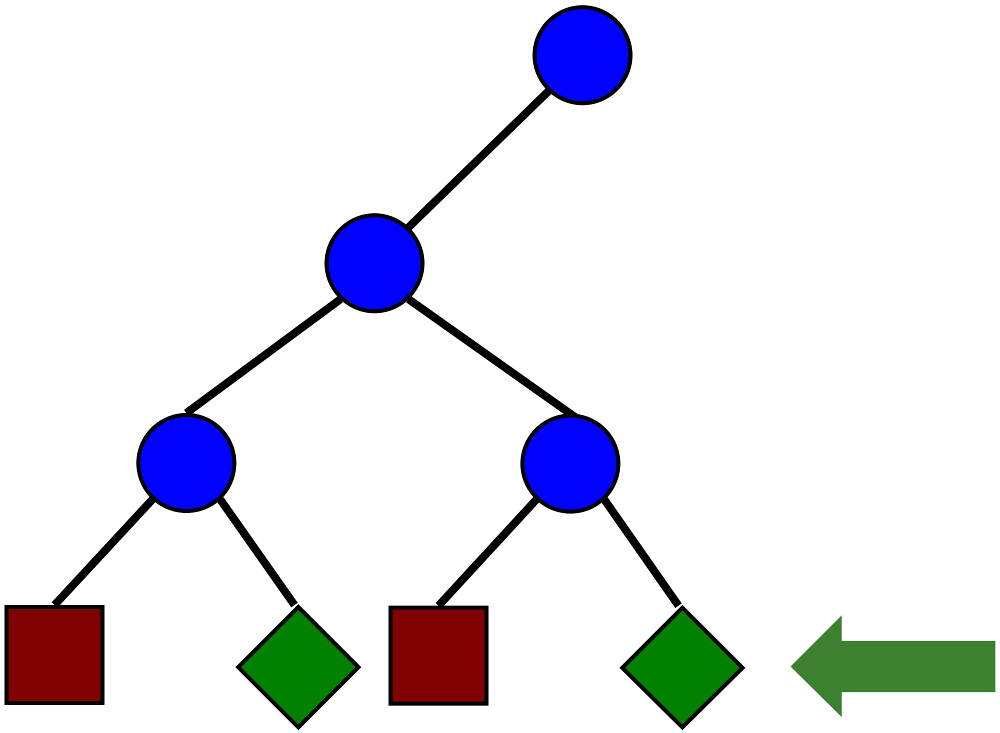
- Explore with additional constraint

Branch-and-bound Search



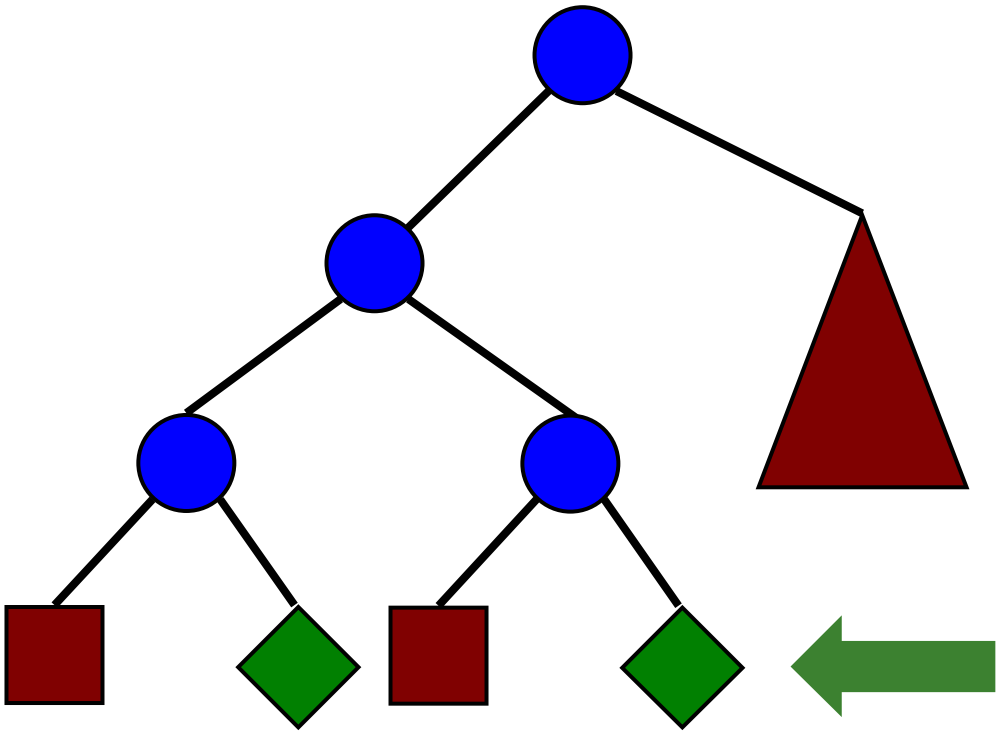
- Guarantees better solutions

Branch-and-bound Search



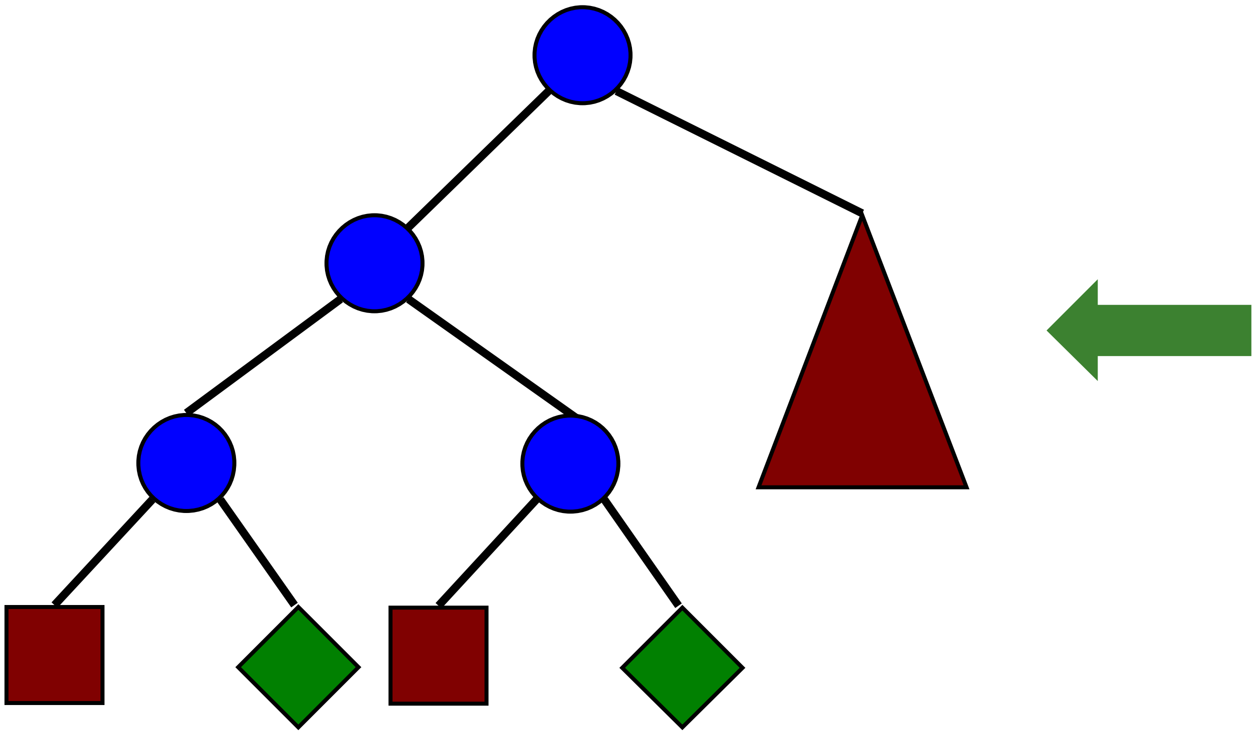
- Guarantees better solutions

Branch-and-bound Search



- Last solution best

Branch-and-bound Search



- Proof of optimality

Modelling SMM++

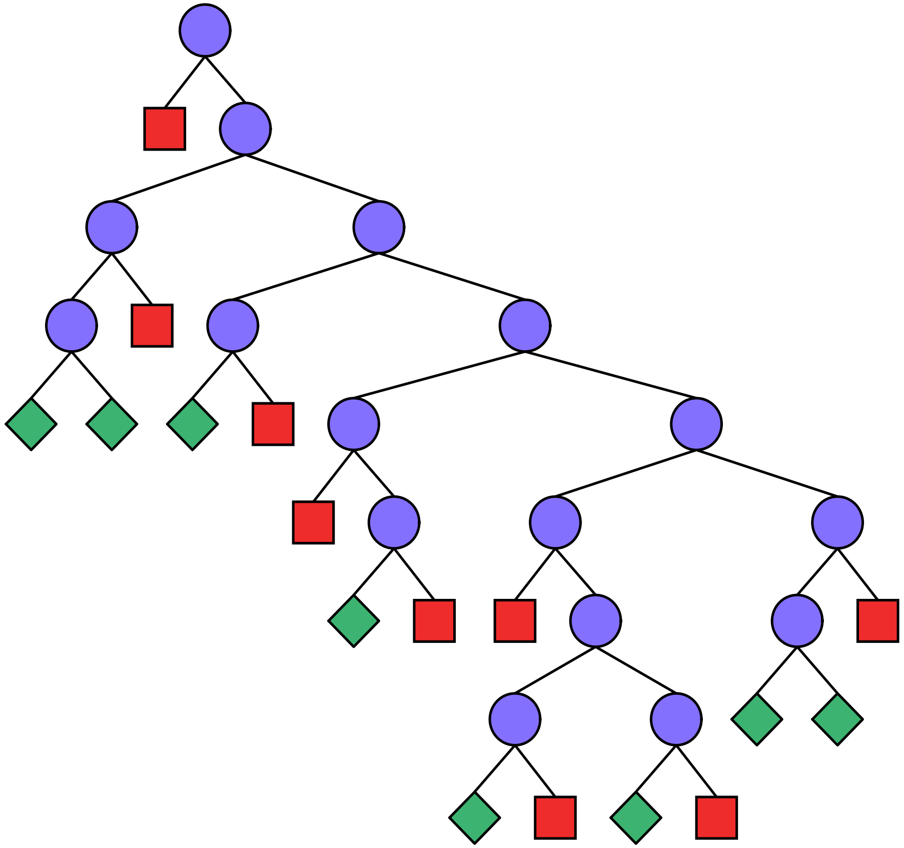
- Constraints and branching as before
- Order among solutions with constraints
  - so-far-best solution **S,E,N,D,M,O,T,Y**
  - current node **S,E,N,D,M,O,T,Y**
  - constraint added

$$10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y$$

$$<$$

$$10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y$$

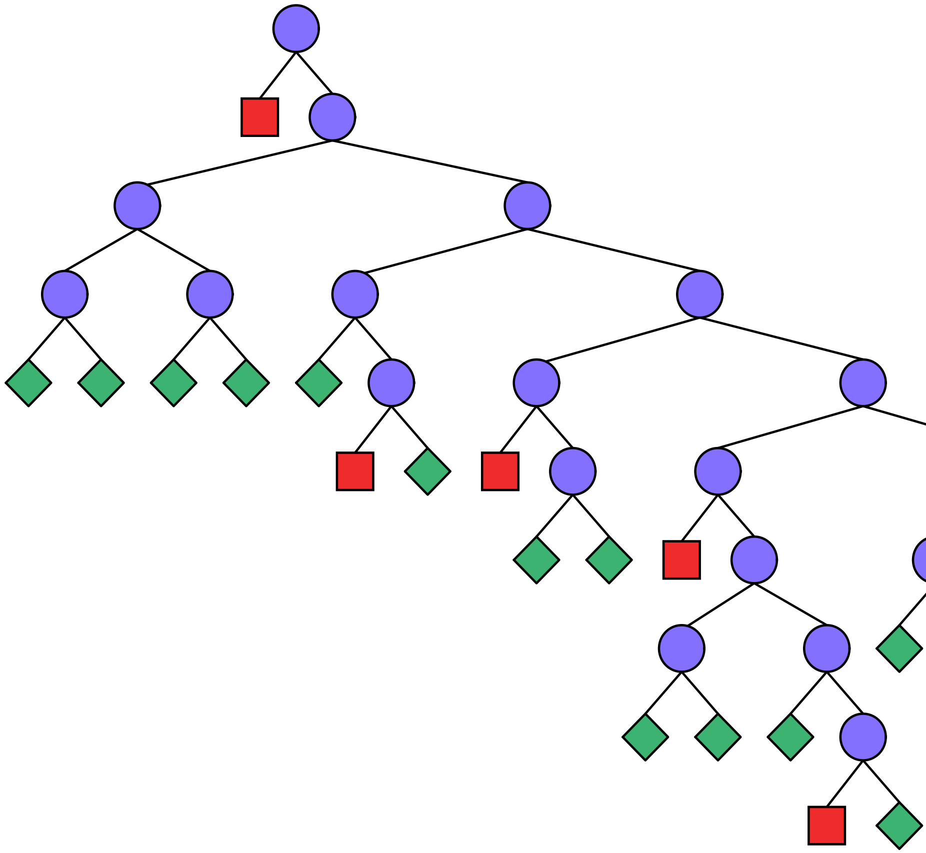
SMM++: Branch-and-bound



```
   SEND
+  MOST
-------
= MONEY

   9782
+  1094
-------
= 10876
```

SMM++: All Solution Search



```
    SEND
+   MOST
--------
=  MONEY

    9782
+   1094
--------
=  10876
```

Summary: Key Ideas and Principles

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

applications

principles

Excursion
Constraint Propagation
Revisited

Constraint Propagation

- Variables (as members of store)
  - feature variable domain (here: finite set of integers)
- Propagators
  - implement constraints
- Propagation loop
  - execute propagators until simultaneous fixpoint

Propagator

- Propagator *p* is procedure
  - implements constraint con(*p*)
    its semantics (set of tuples)
  - computes on set of variables var(*p*)
- Execution of propagator *p*
  - narrows domains of variables in var(*p*)
  - signals failure

Propagators Are Intensional

- Propagators implement narrowing
  - also: filtering, propagation, domain reduction
- No extensional representation of con(*p*)
  - impractical in most cases (space)
- Extensional representation of constraint
  - can be provided by special propagator
  - often: “element” constraint, “relation” constraint, …

Propagator Properties

- Propagator *p* is
  - correct: no solution of con(*p*) is removed
  - assignment complete: failure at latest for assignments
    - compatibility with search
- Propagator *p* is
  - contracting: variable domains are narrowed
  - monotonic: application to smaller domains will result in smaller domains than application to larger domains

Propagation Loop

- Largest simultaneous fixpoint of propagators
  - fixpoint: propagators cannot narrow any further
  - largest: no solutions lost
- Guaranteed
  - termination: domains finite
    propagators contracting
  - largest fixpoint: propagators monotonic

Detailed study with proofs: [Apt 00]

Fix and Runnable Propagators

- Propagator is either
  - fix: has reached fixpoint
  - runnable: not known to have reached fixpoint
- Propagation loop maintains propagator sets
  - all propagators *Prop*
  - runnable propagators *Run*
  - initially *Run := Prop*

Sketch of Propagation Loop

```
while (Run ≠ ∅) {
        pick and remove p from Run;
        execute p;
        ModVar := { x | x modified by p };
        DepProp := { q | x∈var(q), x∈ModVar };
        Run := join(DepProp, Run);
}
```

Sketch of Propagation Loop

while ($Run \neq \emptyset$) {

pick and remove p from Run;

execute p;

$ModVar$:= { x | x modified by p };

$DepProp$:= { q | $x \in \mathrm{var}(q)$, $x \in ModVar$ };

Run := join($DepProp$, Run);

}

Loop invariant: p is fix $\Leftrightarrow$ $p \in (Prop - Run)$

Sketch of Propagation Loop

```
while (Run ≠ ∅) {
        pick and remove p from Run;
        execute p;
        ModVar := { x | x modified by p };
        DepProp := { q | x∈var(q), x∈ModVar };
        Run := join(DepProp, Run);
}
```

Termination ($Run = \emptyset$): p is fix $\Leftrightarrow$ $p \in Prop$

Sketch of Propagation Loop

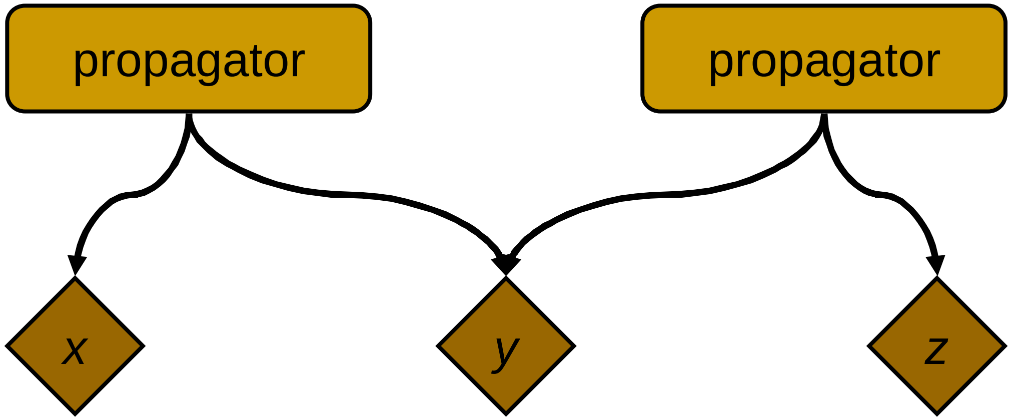
```
while (Run ≠ ∅) {
        pick and remove p from Run;
        execute p;
        ModVar := { x | x modified by p };
        DepProp := { q | x∈var(q), x∈ModVar };
        Run := join(DepProp, Run);
}
```

Ignored: failure (signaled by *p*)

Implementing *ModVar* and *DepProp*

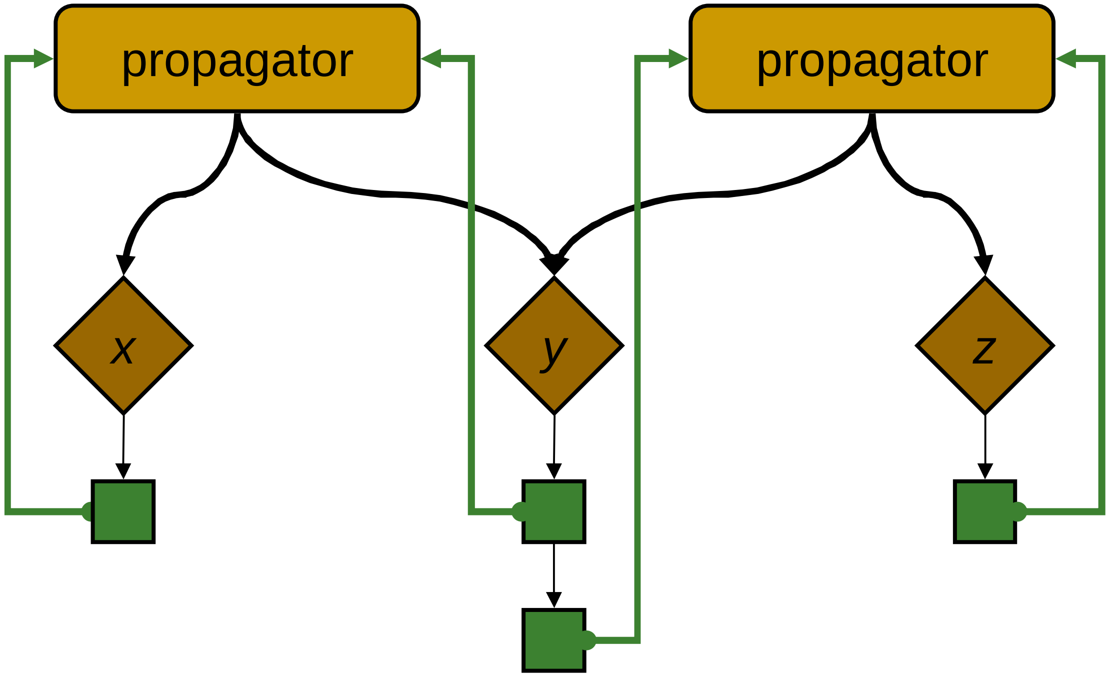
- Variable-centered approach
  - each variable *x* knows dependent propagators
  - typically organized as list (*suspension list*)
  - propagator *p* included in list of *x* ⇔ $x \in \mathrm{var}(p)$
- Upon propagator creation
  - propagator subscribes to its variables
  - becomes runnable

Propagators ⇨ Variables



- **Propagators** know their **variables**
  - to perform domain modifications
  - passed as parameters to propagator creation

Variables ⇨ Propagators



- **Variables** know dependent **propagators**
  - to perform efficient computation of dependent propagators

Modifying a Variable

- Traverse suspension list
  - add propagators to *Run*
- Optimization
  - mark runnable propagators
  - that is: propagators already in *Run*
- Multiple variable modification by propagator
  - explicitly maintain *ModVar* (as in model)
  - only after propagator execution: process *ModVar*
  - suspension list traversed only once per variable

Idempotent Propagators

- Idempotent propagator
  - always computes fixpoint
- Propagation loop perspective
  - no need to include in *Run*
  - more efficient: saves one invocation of propagator
- Propagator perspective
  - must compute fixpoint itself
  - more efficient: specific method for computing fixpoint
  - might be more challenging

Propagator Entailment

- Propagator will never contribute anything
  - fixpoint property preserved by narrowing
- Delete propagator, if entailment detected
  - remove from suspension-list, or
  - mark as dead, delegate removal to garbage collection

Summary: Constraint Propagation Revisited

- Variables
  - domain, suspension list
- Propagators
  - intensional, correct, contracting, monotone, …
  - know variables for narrowing
- Propagation loop
  - computes largest simultaneous fixpoint

Why Does Constraint Programming 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, matching, networks, ...
- Programming languages
  - programmability, extensionability, ...

Essential Aspect

- Compositional middleware for combining
  - smart algorithmic
  - problem substructures

  components (propagators)
  - scheduling
  - graphs
  - flows
  - …

  plus
  - essential extra constraints

Significance

- Constraint programming identified as a strategic direction in computer science research

  [ACM Computing Surveys, December 1996]

Excursion
Scheduling Resources

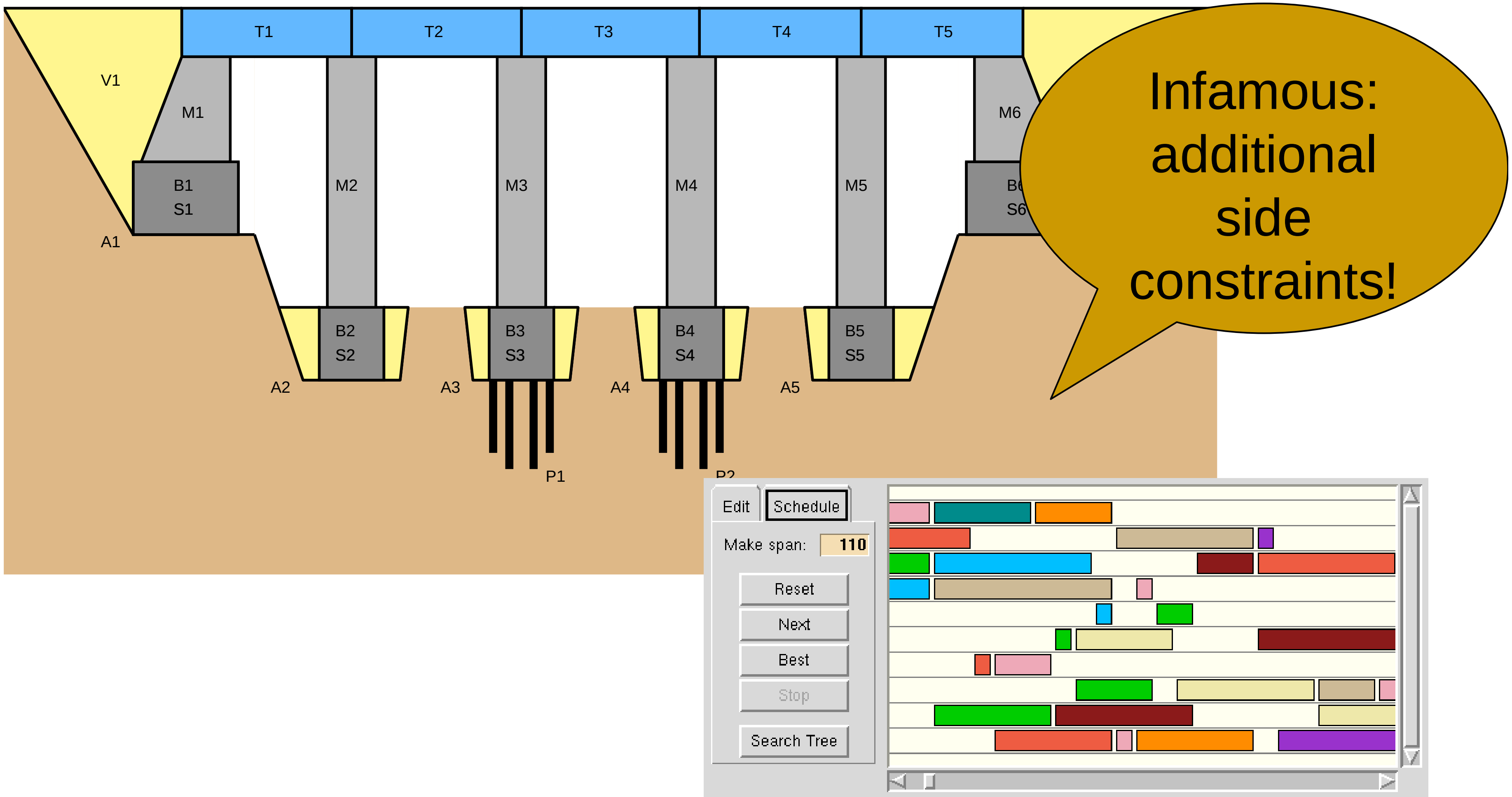
- Modelling
- Propagation
- Strong propagation

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*

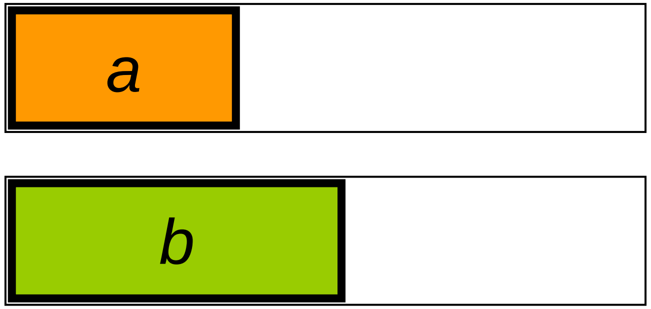
  $start(a)$

- Precedence constraint: *a* before *b*

  $start(a) + dur(a) \leq start(b)$

Propagating Precedence

a before *b*

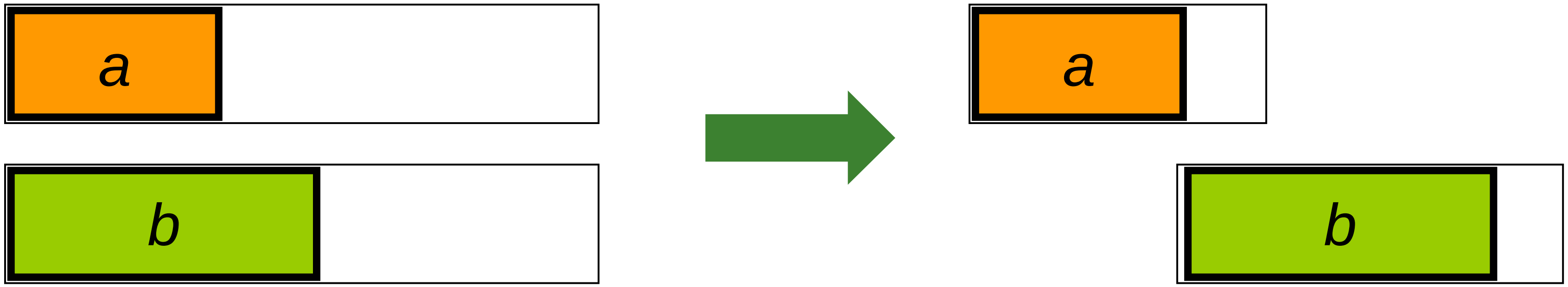


start(*a*)∈{0,…,7}

start(*b*)∈{0,…,5}

Propagating Precedence

a before *b*



start(*a*)∈{0,…,7}
start(*b*)∈{0,…,5}

start(*a*)∈{0,…,2}
start(*b*)∈{3,…,5}

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

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

Reifited Constraints

- Use control variable $b \in \{0,1\}$

$$c \quad \leftrightarrow \quad b=1$$

- Propagate
  - c holds $\Rightarrow$ propagate $b=1$
  - $\neg c$ holds $\Rightarrow$ propagate $b=0$
  - $b=1$ holds $\Rightarrow$ propagate c
  - $b=0$ holds $\Rightarrow$ propagate $\neg c$

Reified Constraints

- Use control variable $b \in \{0,1\}$

$$c \quad \leftrightarrow \quad b=1$$

- Propagate
  - c holds $\Rightarrow$ p
  - $\neg c$ holds $\Rightarrow$ pr
  - b=1 holds $\Rightarrow$ propagate c
  - b=0 holds $\Rightarrow$ propagate $\neg c$

not easy!

Reification for Disjunction

- Reify each precedence

  $$[\text{start}(a) + \text{dur}(a) \leq \text{start}(b)] \leftrightarrow b_0{=}1$$

  and

  $$[\text{start}(b) + \text{dur}(b) \leq \text{start}(a)] \leftrightarrow b_1{=}1$$

- Model disjunction

  $$b_0 + b_1 \geq 1$$

Model Is Too Naive

- Local view
  - individual task pairs
  - $O(n^2)$ propagators for *n* tasks

- Global view ("global" constraints)
  - all tasks on resource
  - single propagator
  - smarter algorithms possible

Example: Edge Finding

- Find ordering among tasks ("edges")
- For each subset of tasks $\{a\} \cup B$
  - assume: *a* before *B*

    deduce information for *a* and *B*
  - assume: *B* before *a*

    deduce information for *a* and *B*
  - join computed information
  - can be done in $O(n^2)$

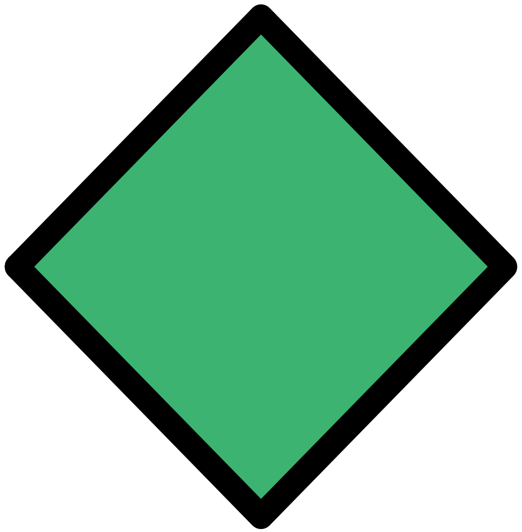
Summary

- Modelling
  - easy but not always efficient
  - constraint combinators (reification)
  - global constraints
  - smart heuristics
- More on constraint-based scheduling

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

Excursion
Strong Propagation

SMM: Strong Propagation



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

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

Example: Distinct Propagator

- Infeasible: decomposition
  - O(n^2) disequality propagators
- Naive distinct propagator
  - wait until variable becomes assigned
  - remove value from all other variables
- Strong distinct propagator
  - only keep values appearing in a solution to constraint
  - essential for many problems

Distinct Propagator: Hall Sets

- Direct approach: Hall sets
  - Van Beek, Quimper, et. al. [CP 2004]
- Set $\{x_1, ..., x_n\}$ of variables Hall set, iff set of values $s(x_1) \cup ... \cup s(x_n)$ has cardinality n
- Pruning
  - find Hall set H
  - prune values in H from all other variables

Strong Distinct Propagator

- Can be propagated efficiently
  - O($n^{2.5}$) is efficient
  - breakthrough: Régin, A filtering algorithm for constraints of difference in CSPs, AAAI 1994.
- Uses graph algorithms
  - insight on problem structure
  - relation between solutions of constraint and properties of graph

Régin's Approach

- Construct a variable-value graph
  - bipartite graph: variable nodes → value nodes
- Characterize solutions in graph
  - maximal matchings
- Use matching theory
  - one matching can describe all matchings
- Remove edges not representing solutions

Variable Value Graph

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

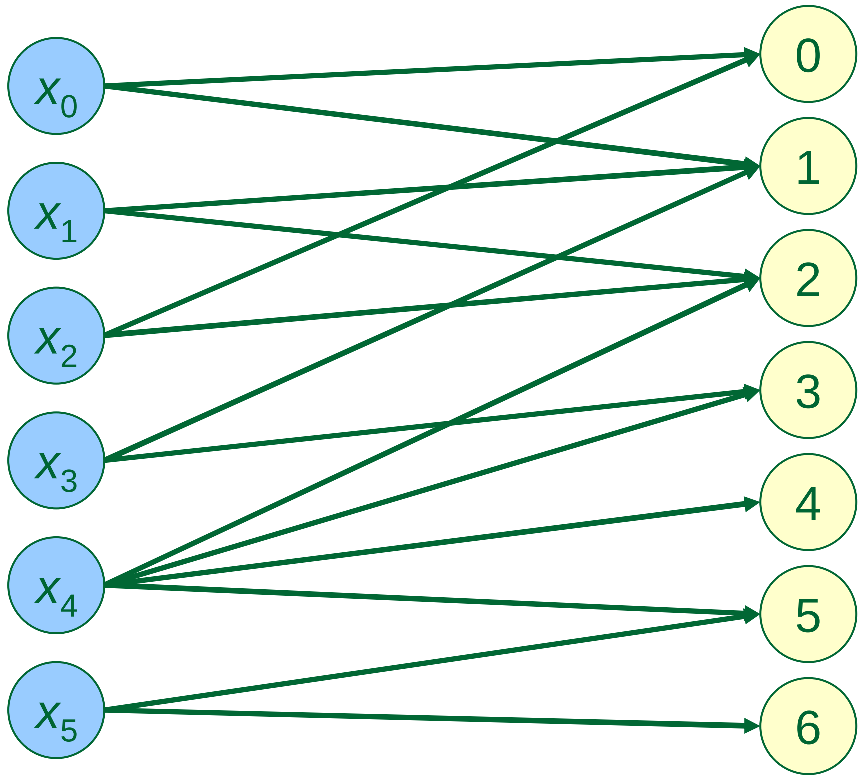
$s(x_1)=\{1,2\}$

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

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

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

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



Maximal Matching Are Solutions

$a(x_0)=0$

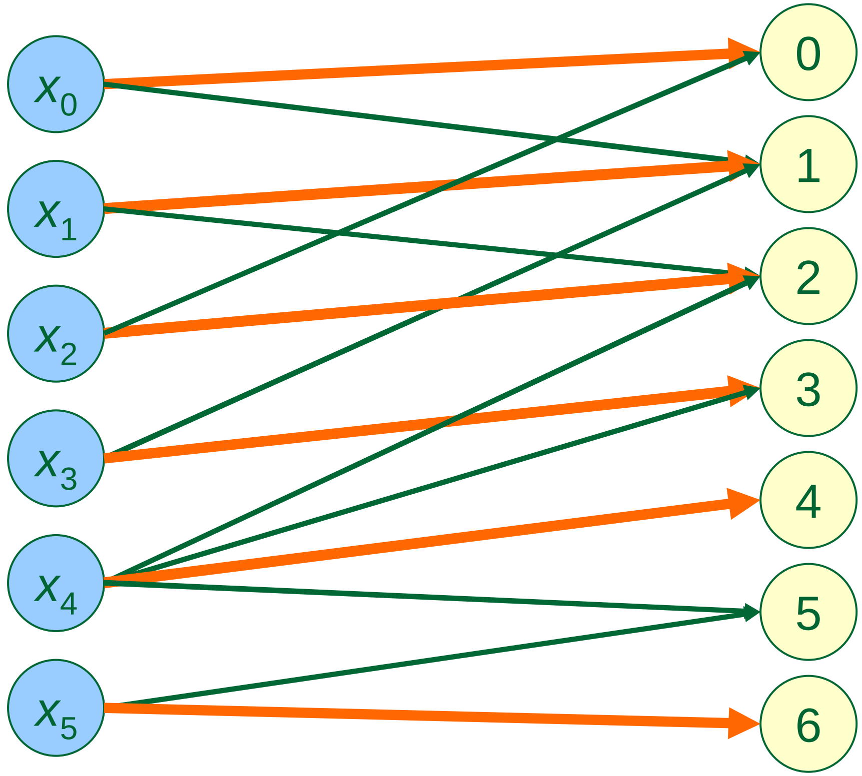
$a(x_1)=1$

$a(x_2)=2$

$a(x_3)=3$

$a(x_4)=4$

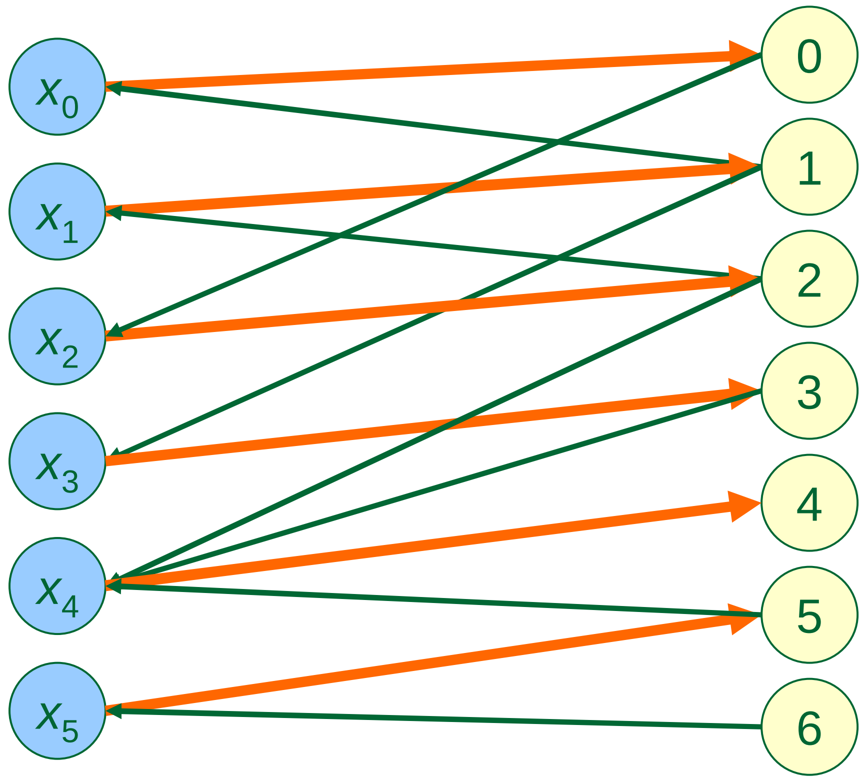
$a(x_5)=6$



Matching Theory

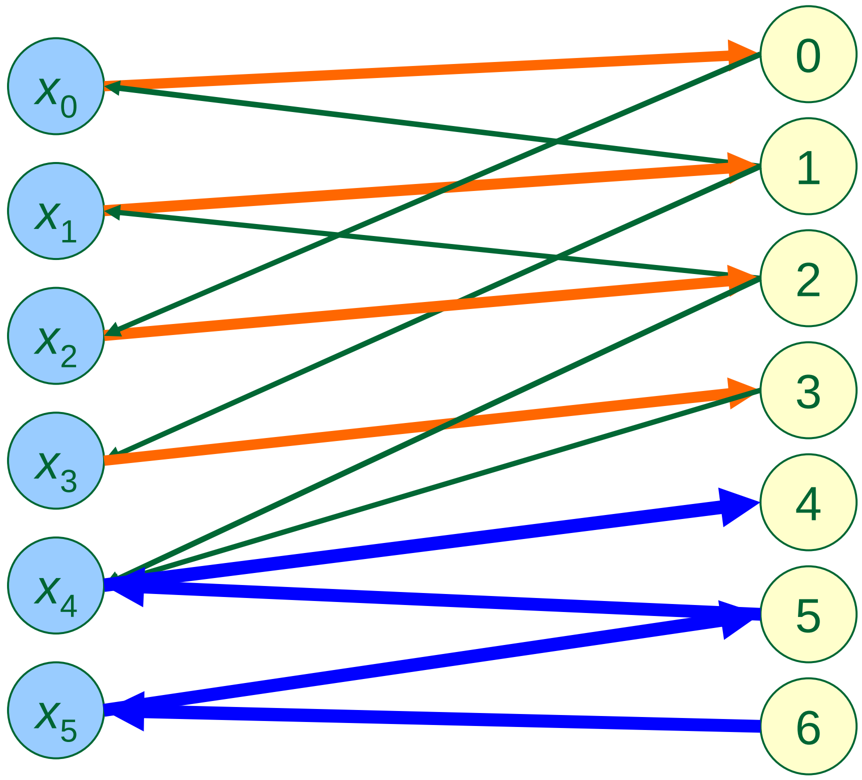
- Edge *e* belongs to some matching ⇔
  for some arbitrary matching *M*:
  either: *e* belongs to even alternating path starting at free node
  or: *e* belongs to even alternating cycle
  - [C. Berge, 1970] See Régin's paper

Oriented Graph: Alternation



Alternating Paths…

- Only free node: 6
  - mark 6 → x_5
  - mark x_5 → 5
  - mark 5 → x_4
  - mark x_4 → 4
- Intuition: edges can be permuted

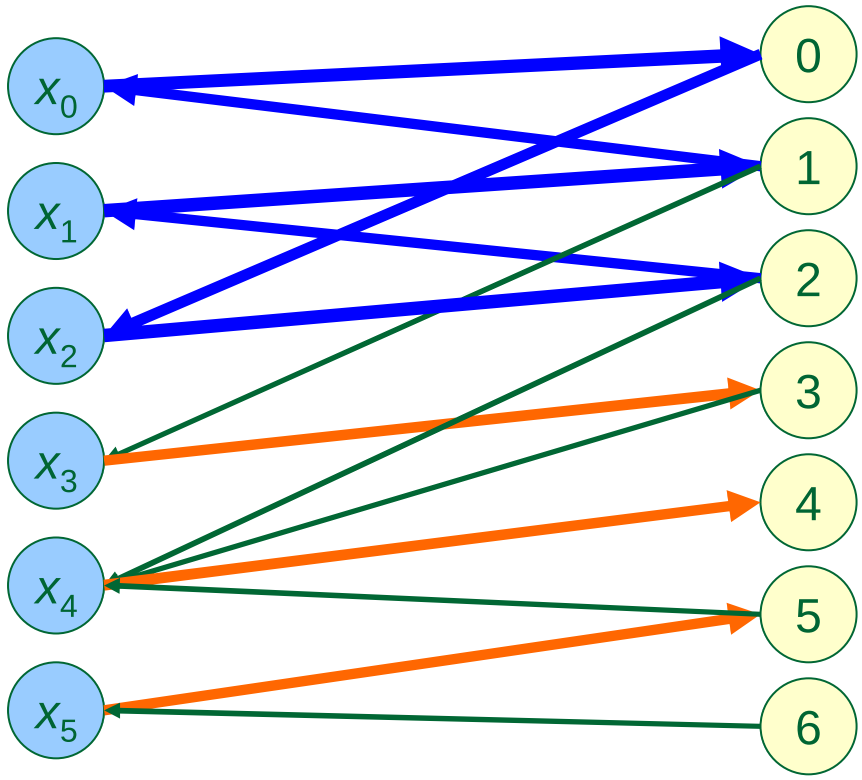


Alternating Cycles…

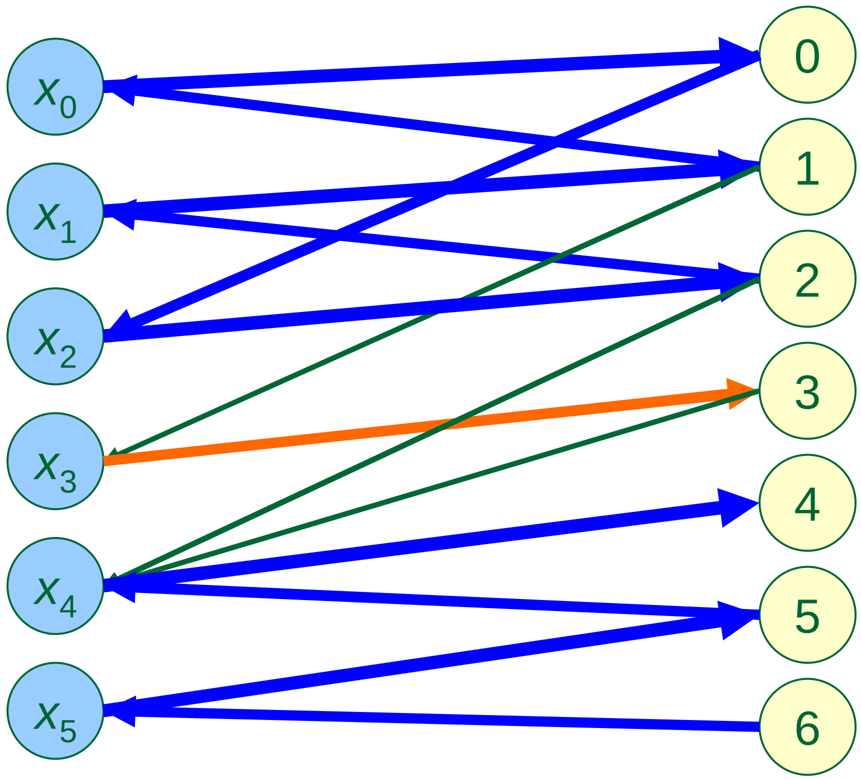
- Nodes in SCC

  $x_0, x_1, x_2,$

  0, 1, 2
- Mark joining edges
- Intuition: variables take all values from SCC

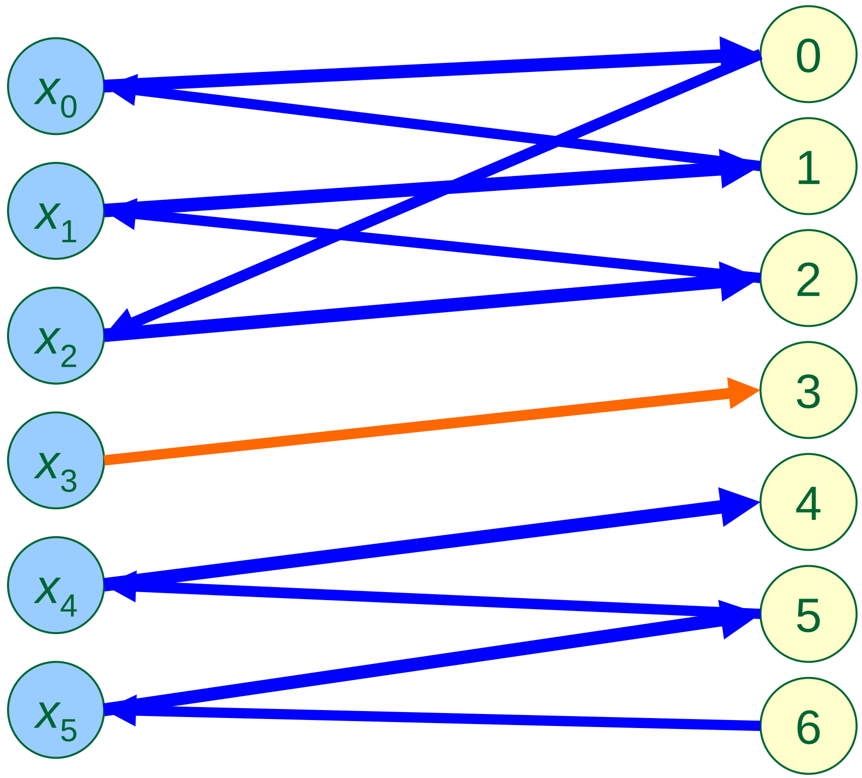


All Marked Edges



Edges Removed

- Remove
  - 1 → x_3
  - 2 → x_4
  - 3 → x_4
- Keep
  - x_3 → 3
  - matched!
- Edge removal
  - value removal



Characterising Strength: Consistency

- Domain-consistent propagator for constraint
  - every value appears in at least one solution of constraint
  - strongest possible propagation
  - Régin's method is domain-consistent
  - also known as: generalized arc consistency, ...
- Bounds-consistent propagator for constraint
  - extremal values appears in solution of convex relaxation
  - depends on relaxation: integer versus real
  - weaker but cheaper yet relevant
  - confusion about variants...

Global Constraints

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

- Plethora available
  - scheduling, sequencing, cardinality, sorting, circuits, ...
  - systematic catalogue with hundreds available...
  - difficult to pick the right one (consistency versus efficiency, etc)

Trends and State of the Art

Trends and State of The Art

- Focus here
    - constraints for combinatorial problems

ignoring
    - programming languages, graphics, databases, tractability, complexity, ...

- Up-to-date overview

Handbook of Constraint Programming

Rossi, van Beek, Walsh, eds., Elsevier, 2006.

Modelling

- Symmetry breaking
- Implied constraints
- Variable domains
- Soft constraints
- Modelling languages
- ...

Symmetry Breaking

- Absolutely essential
  - just search for single solution, ignore symmetric solutions
  - drastically prunes search space
  - without, most problems can not be solved
- Key questions
  - how to find symmetries automatically?
  - class of symmetries: value, variable symmetries?
  - how to break them (rule out symmetric solutions)?
  - how many to break (all typically to expensive)?
  - break them statically or dynamically?
  - break them during search?

Implied Constraints

- Absolutely essential
  - find constraints that are semantically implied
  - yet provide essential propagation

- Key questions
  - how to find them?
  - manual versus automatic?
  - how to propagate them?

Variable Domains

- Finite sets, multisets, intervals, ...
- Often help to avoid symmetries (sets)
- Typically require approximation
  - full set representation: exponential time and space
  - bounds approximation: describe by glb and lub
- Key questions
  - total ordering for symmetry breaking?
  - efficient representations?
  - efficient and strong propagators for global constraints?

Soft Constraints

- Important to capture inconsistent models
  - as they tend to be in practice
- Devise new framework
  - generalize propagation to cater for softness
- Remain in same framework
  - propagators that propagate according to degree of violation
- …

Modelling Languages

- Fundamental difference to LP and SAT
  - language has structure (global constraints)
  - different solvers support different constraints
- In its infancy
- Key questions:
  - what level of abstraction?
  - solving approach independent: LP, SAT, CP, ...?
  - how to map to different systems?

Solving

- Automatic solving ("black box" solvers)
- Constraint-based local search
- Hybrid approaches
- Constraint programming systems
- Global constraints
- ...

Automatic Solving

- Modelling is very difficult for CP
  - requires lots of knowledge and tinkering
  - very different from SAT
- How to automatize
  - restart search?
  - automatic symmetric breaking?
  - new idea, promising first ideas and approaches?
  - to which extent possible?

Constraint-based Local Search

- Local search
  - operate on assignments not necessarily solutions
  - find "good" assignments
- Use constraints as abstractions to model and solve with local search
- Derive implementations automatically from constraints
- Hybrid approaches?
- Very promising
  - check out Comet: www.comet-online.org

Hybrid Approaches

- Operations research methods
- Key issue: CP poor for optimization
- Key questions
  - relaxations to obtain bounds?
  - column generation?
  - Benders decomposition?
  - cuts?
- Extremely important for practical problems

Global Constraints

- Ever more! Ever more?
- Key questions
  - what are the essential primitive ones?
  - how to characterize them?
  - how to automatically get an implementation?

Constraint Programming Systems

- Essential for initial and continuing success
- Two approaches
  - library-based: ILOG Solver, Koalog, Choco, Gecode, ...
  - language-based: SICStus Prolog, Eclipse, Oz, ...
- Key questions
  - parallelism
  - efficiency
  - robustness
  - automatic
  - coverage

Summary

Constraint Programming

- Powerful approach for modelling and solving combinatorial problems
- Key aspect: middleware for
  - powerful algorithmic components
  - essential extra constraints
- Key issues: modelling, propagation, search
- Widely used but modelling is challenging