Centre for
Data Analytics

Insight

Introduction to Constraint Programming

Helmut Simonis

CRT-AI Training Program, March 29th, 2021

DCU NUI Galway UCC UCD sfi

Licence

Acknowledgments

The author is partially supported by Science Foundation Ireland (Grant Number 05/IN/I886). This material was developed as part of the ECLiPSe ELearning course:
`https://eclipseclp.org/ELearning/index.html`
Support from Cisco Systems and the Silicon Valley Community Foundation is gratefully acknowledged.

Objectives

- Overview of Core Constraint Programming
- Three Main Concepts
  - Constraint Propagation
  - Global Constraints
  - Customizing Search
- Get Some Experience with MiniZinc
- Based on Examples, not Formal Description

Outline

- Why Constraint Programming?
- Constraint Propagation
- Global Constraints
- Customizing Search

Using MiniZinc IDE

- Developed in the Australian NICTA project
- Maintained by Monash University
- Modelling tool with multiple back-end solvers
- Available from `https://www.minizinc.org/`

Examples in ECLiPSe

- Open sourced constraint programming language
- Development goes back to 1985
- ECRC, ICL, IC-Parc, PTL, Cisco
- `https://eclipseclp.org/`
- Specialities
  - Develop new solvers for specific domains
  - Integration with MIP
- Not included in bundled MiniZinc IDE
- Specialized visualization tools used here
  - CP-Viz, Simonis et al. 2010

Course Based on ECLiPSe ELearning Course

- Self-study course in constraint programming
- Supported by Cisco Systems and Silicon Valley Community Foundation
- Multi-media format, video lectures, slides, handout etc
- `https://eclipseclp.org/ELearning/index.html`

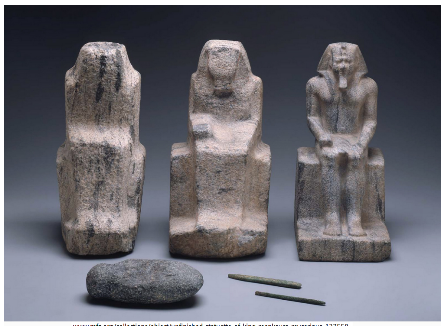
Constraint Programming - in a nutshell

- Declarative description of problems with
  - *Variables* which range over (finite) sets of values
  - *Constraints* over subsets of variables which restrict possible value combinations
  - A *solution* is a value assignment which satisfies all constraints
- Constraint propagation/reasoning
  - Removing inconsistent values for variables
  - Detect failure if constraint can not be satisfied
  - Interaction of constraints via shared variables
  - Incomplete
- Search
  - User controlled assignment of values to variables
  - Each step triggers constraint propagation
- Different domains require/allow different methods

Constraint Programming is Different

- Declarative Programming
  - Concentrate on what you want
  - Not how to get there
  - Program != Algorithm
  - Program = Model
- Applied to Combinatorial Problems
  - No complete polynomial algorithms known (exist?)
  - CP less ad-hoc than heuristics
  - Models can evolve

A Subtractive Process


www.mfa.org/collections/object/unfinished-statuette-of-king-menkaura-mycerinus-137558

“Oh, bosh, as Mr. Ruskin says. Sculpture, per se, is the simplest thing in the world. All you have to do is to take a big chunk of marble and a hammer and chisel, make up your mind what you are about to create and chip off all the marble you don’t want.”-Paris Gaulois.

Source: `https://quoteinvestigator.com/2014/06/22/chip-away/`

Basic Process

Problem

Human

↓

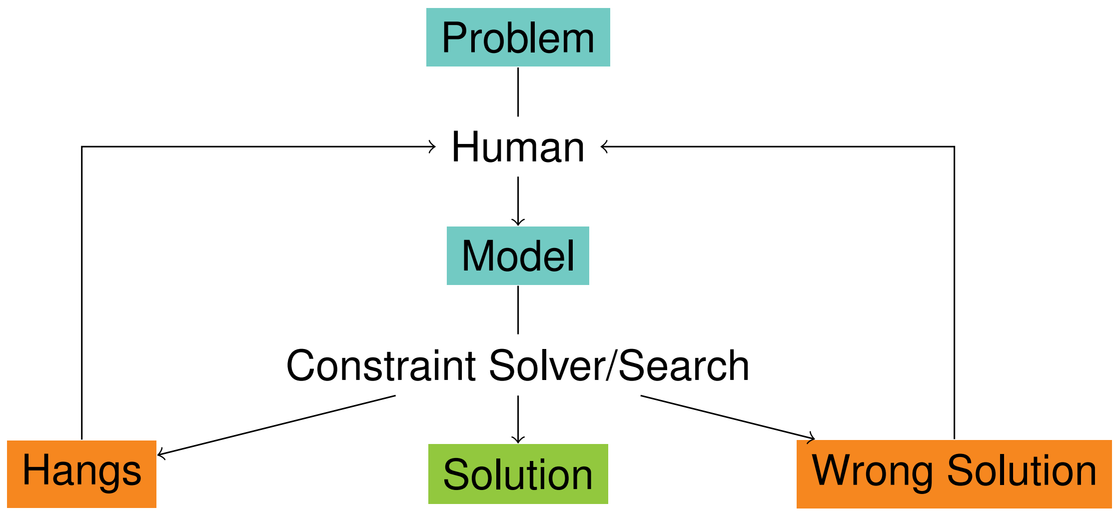
Model

Constraint Solver/Search

↓

Solution

More Realistic



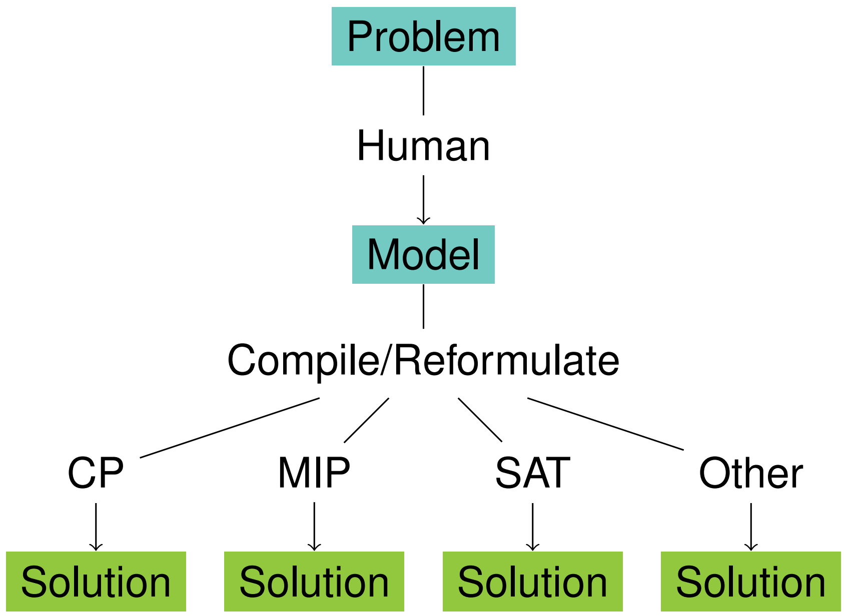
Dual Role of Model

- Allows Human to Express Problem
  - Close to Problem Domain
  - Constraints as Abstractions
- Allows Solver to Execute
  - Variables as Communication Mechanism
  - Constraints as Algorithms

Modelling Frameworks

- MiniZinc (NICTA, Monash University, Australia)
- NumberJack (Insight, Ireland)
- Essence (UK)
- Allow use of multiple back-end solvers
- Compile model into variants for each solver
- A priori solver independent model(CP, MIP, SAT)

Framework Process



Do It Now!

- Download and install Minizinc
- `https://www.minizinc.org/`

Part I

Basic Constraint Propagation

Example 1: SEND+MORE=MONEY

- Example of Finite Domain Constraint Problem
- Models and Programs
- Constraint Propagation and Search
- Some Basic Constraints: linear arithmetic, alldifferent, disequality
- A Built-in search
- Visualizers for variables, constraints and search

Problem Definition

A Crypt-Arithmetic Puzzle

We begin with the definition of the SEND+MORE=MONEY puzzle. It is often shown in the form of a hand-written addition:

$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

Rules

- Each character stands for a digit from 0 to 9.
- Numbers are built from digits in the usual, positional notation.
- Repeated occurrence of the same character denote the same digit.
- Different characters denote different digits.
- Numbers do not start with a zero.
- The equation must hold.

$$
\begin{array}{cccccc}
 & & S & E & N & D \\
+ & & M & O & R & E \\
\hline
 & M & O & N & E & Y
\end{array}
$$

Model

- Each character is a variable, which ranges over the values 0 to 9.
- An *alldifferent* constraint between all variables, which states that two different variables must have different values. This is a very common constraint, which we will encounter in many other problems later on.
- Two *disequality constraints* (variable X must be different from value V) stating that the variables at the beginning of a number can not take the value 0.
- An arithmetic *equality constraint* linking all variables with the proper coefficients and stating that the equation must hold.

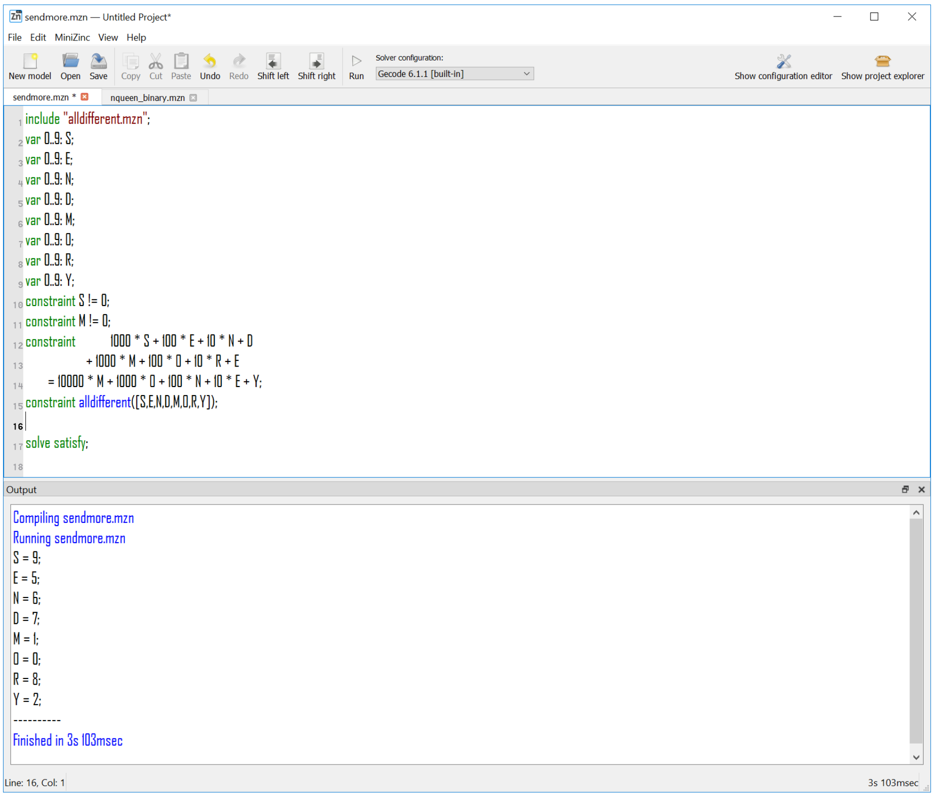
MiniZinc

```
include "alldifferent.mzn";
var 0..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 0..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;
constraint S != 0;
constraint M != 0;
constraint           1000 * S + 100 * E + 10 * N + D
                   + 1000 * M + 100 * O + 10 * R + E
       = 10000 * M + 1000 * O + 100 * N + 10 * E + Y;
constraint alldifferent([S,E,N,D,M,O,R,Y]);

solve satisfy;
```


Choice of Model

- This is *one* model, not *the* model of the problem
- Many possible alternatives
- Choice often depends on your constraint system
  - Constraints available
  - Reasoning attached to constraints
- Not always clear which is the *best* model
- Often: Not clear what is the *problem*

Running the Program (MiniZinc IDE)



Question

- But how did the program come up with this solution?
- We show solution with ECLiPse, other solvers vary slightly

Domain Definition

```
var 0..9: S;
var 0..9: E;
var 0..9: N;
var 0..9: D;
var 0..9: M;
var 0..9: O;
var 0..9: R;
var 0..9: Y;
```


Domain Visualization

Columns = Values

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M			Cells= State							
O										
R										
Y										

Rows = Variables

Alldifferent Constraint

```
include "alldifferent.mzn";

constraint alldifferent([S,E,N,D,M,O,R,Y]);
```

- Built-in alldifferent predicate included
- No initial propagation possible
- *Suspends*, waits until variables are changed
- When variable is fixed, remove value from domain of other variables
- *Forward checking*

Alldifferent Visualization

Uses the same representation as the domain visualizer

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

Disequality Constraints

```
constraint S != 0;
constraint M != 0;
```

Remove value from domain

$$S \in \{1..9\}, M \in \{1..9\}$$

Constraints solved, can be removed

Domains after Disequality

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

Equality Constraint

- Normalization of linear terms
  - Single occurence of variable
  - Positive coefficients
- Propagation

Normalization

	1000*S+	100*E+	10*N+	D
	+1000*M+	100*O+	10*R+	E
10000*M+	1000*O+	100*N+	10*E+	Y

is transformed into

	1000*S+	91*E+		D
		+	10*R	
9000*M+	900*O+	90*N+		Y

Simplified Equation

$$1000 * S + 91 * E + 10 * R + D = 9000 * M + 900 * O + 90 * N + Y$$

Propagation

$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{1000..9918} =$$

$$\underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..89919}$$

Deduction:

$$M = 1, S = 9, O \in \{0..1\}$$

Why? Skip

Consider lower bound for S

$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{9000..9918} = \underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..9918}$$

- Lower bound of equation is 9000
- Rest of lhs (left hand side) $(91 * E^{0..9} + 10 * R^{0..9} + D^{0..9})$ is atmost 918
- S must be greater or equal to $\frac{9000-918}{1000} = 8.082$
  - otherwise lower bound of equation not reached by lhs
- S is integer, therefore $S \geq \lceil\frac{9000-918}{1000}\rceil = 9$
- S has upper bound of 9, so $S = 9$

Consider upper bound of M

$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{9000..9918} = \underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..9918}$$

- Upper bound of equation is 9918
- Rest of rhs (right hand side) $900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}$ is at least 0
- M must be smaller or equal to $\frac{9918-0}{9000} = 1.102$
- M must be integer, therefore $M \leq \lfloor\frac{9918-0}{9000}\rfloor = 1$
- M has lower bound of 1, so $M = 1$

Consider upper bound of O

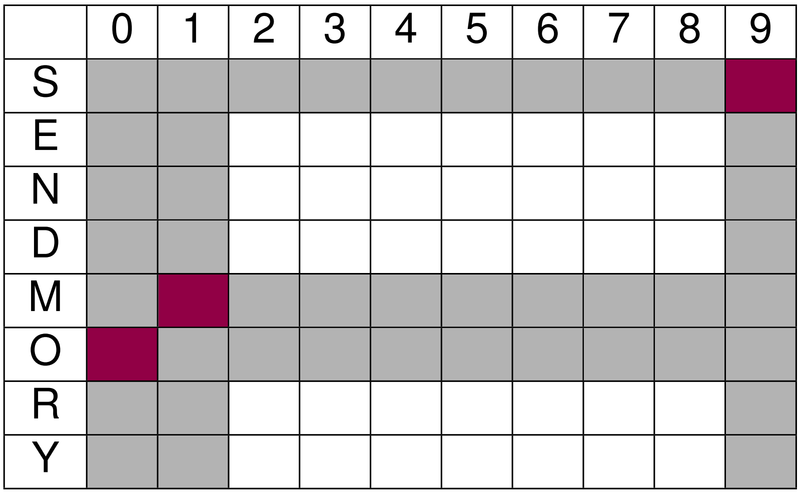
$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{9000..9918} = \underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..9918}$$

- Upper bound of equation is 9918
- Rest of rhs (right hand side) $9000 * 1 + 90 * N^{0..9} + Y^{0..9}$ is at least 9000
- O must be smaller or equal to $\frac{9918-9000}{900} = 1.02$
- O must be integer, therefore $O \leq \lfloor \frac{9918-9000}{900} \rfloor = 1$
- O has lower bound of 0, so $O \in \{0..1\}$

Propagation of equality: Result

	0	1	2	3	4	5	6	7	8	9
S		-	-	-	-	-	-	-	-	✹
E										
N										
D										
M		✹	-	-	-	-	-	-	-	-
O			✖	✖	✖	✖	✖	✖	✖	✖
R										
Y										

Propagation of alldifferent



$$O = 0, [E, R, D, N, Y] \in \{2..8\}$$

Waking the equality constraint

- Triggered by assignment of variables
- *or* update of lower or upper bound

Removal of constants

$$1000 * 9 + 91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = \\ 9000 * 1 + 900 * 0 + 90 * N^{2..8} + Y^{2..8}$$

$$\mathbf{1000} * \mathbf{9} + 91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = \\ \mathbf{9000} * \mathbf{1} + \mathbf{900} * \mathbf{0} + 90 * N^{2..8} + Y^{2..8}$$

$$91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = 90 * N^{2..8} + Y^{2..8}$$

Propagation of equality (Iteration 1)

$$\underbrace{91 * E^{2..8} + 10 * R^{2..8} + D^{2..8}}_{204..816} = \underbrace{90 * N^{2..8} + Y^{2..8}}_{182..728}$$

$$\underbrace{91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = 90 * N^{2..8} + Y^{2..8}}_{204..728}$$

$$N \geq 3 = \lceil \frac{204 - 8}{90} \rceil, E \leq 7 = \lfloor \frac{728 - 22}{91} \rfloor$$

Propagation of equality (Iteration 2)

$$91 * E^{2..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}$$

$$\underbrace{91 * E^{2..7} + 10 * R^{2..8} + D^{2..8}}_{204..725} = \underbrace{90 * N^{3..8} + Y^{2..8}}_{272..728}$$

$$\underbrace{91 * E^{2..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}}_{272..725}$$

$$E \geq 3 = \lceil \frac{272 - 88}{91} \rceil$$

Propagation of equality (Iteration 3)

$$91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8}}_{295..725} = \underbrace{90 * N^{3..8} + Y^{2..8}}_{272..728}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}}_{295..725}$$

$$N \geq 4 = \lceil \frac{295 - 8}{90} \rceil$$

Propagation of equality (Iteration 4)

$$91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8}}_{295..725} = \underbrace{90 * N^{4..8} + Y^{2..8}}_{362..728}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}}_{362..725}$$

$$E \geq 4 = \lceil \frac{362 - 88}{91} \rceil$$

Propagation of equality (Iteration 5)

$$91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8}}_{386..725} = \underbrace{90 * N^{4..8} + Y^{2..8}}_{362..728}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}}_{386..725}$$

$$N \geq 5 = \lceil \frac{386 - 8}{90} \rceil$$

Propagation of equality (Iteration 6)

$$91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8}}_{386..725} = \underbrace{90 * N^{5..8} + Y^{2..8}}_{452..728}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}}_{452..725}$$

$$N \geq 5 = \lceil \frac{452-8}{90} \rceil, E \geq 4 = \lceil \frac{452-88}{91} \rceil$$

No further propagation at this point

Domains after setup

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

Search

solve satisfy;

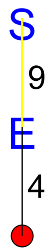
- Try to find a feasible solution, choice left to solver
- Naive search strategy shown here
  - Try variable in order given
  - Try values starting from smallest value in domain
  - When failing, backtrack to last open choice
  - *Chronological Backtracking*
  - *Depth First search*

Search Tree Step 1

Variable *S* already fixed

Step 2, Alternative $E = 4$

Variable $E \in \{4..7\}$, first value tested is 4



Assignment $E = 4$

	0	1	2	3	4	5	6	7	8	9
S										
E					✹	-	-	-		
N										
D										
M										
O										
R										
Y										

Propagation of $E = 4$, equality constraint

$$91 * 4 + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}$$

$$\underbrace{91 * 4 + 10 * R^{2..8} + D^{2..8}}_{386..452} = \underbrace{90 * N^{5..8} + Y^{2..8}}_{452..728}$$

$$\underbrace{91 * 4 + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}}_{452}$$

$$N = 5, Y = 2, R = 8, D = 8$$

Result of equality propagation

	0	1	2	3	4	5	6	7	8	9
S										
E										
N						✸	-	-	-	
D			-	-	-	-	-	-	✸	
M										
O										
R			-	-	-	-	-	-	✸	
Y			✸	-	-	-	-	-	-	

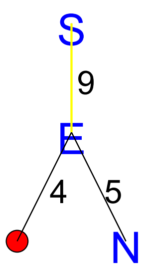
Propagation of alldifferent

	0	1	2	3	4	5	6	7	8	9
S										
E										
N						✹	-	-		
D			-	-	-	-	-	-	✹	
M										
O										
R			-	-	-	-	-	-	✹	
Y			✹	-	-	-	-	-		

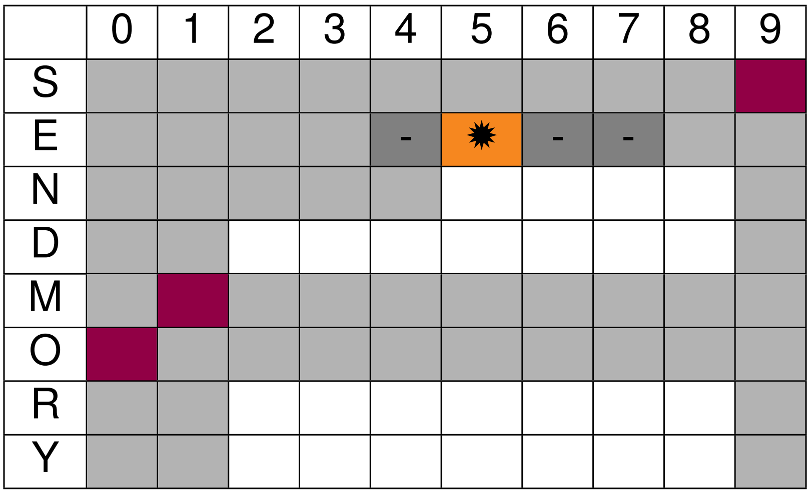
Alldifferent fails!

Step 2, Alternative $E = 5$

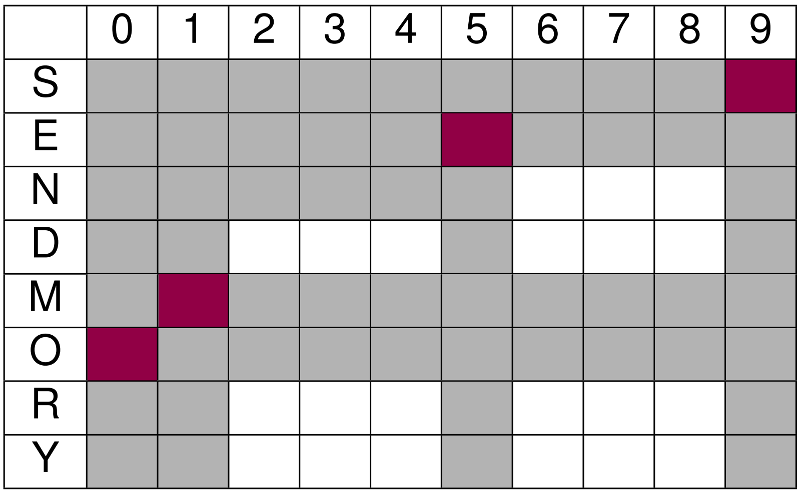
Return to last open choice, E, and test next value



Assignment $E = 5$



Propagation of alldifferent



$$N \neq 5, N \geq 6$$

Propagation of equality

$$91 * 5 + 10 * R^{2..8} + D^{2..8} = 90 * N^{6..8} + Y^{2..8}$$

$$\underbrace{91 * 5 + 10 * R^{2..8} + D^{2..8}}_{477..543} = \underbrace{90 * N^{6..8} + Y^{2..8}}_{542..728}$$

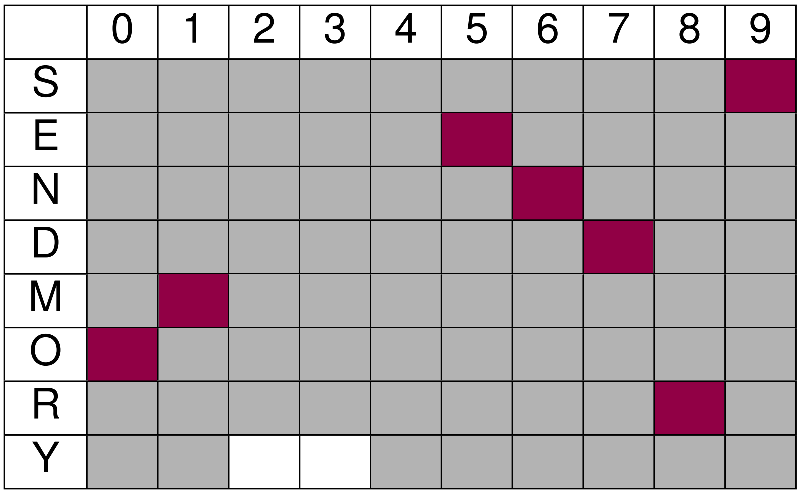
$$\underbrace{91 * 5 + 10 * R^{2..8} + D^{2..8} = 90 * N^{6..8} + Y^{2..8}}_{542..543}$$

$$N = 6, Y \in \{2, 3\}, R = 8, D \in \{7..8\}$$

Result of equality propagation

	0	1	2	3	4	5	6	7	8	9
S										
E										
N							✹	-	-	
D			✖	✖	✖		✖			
M										
O										
R			-	-	-		-	-	✹	
Y					✖		✖	✖	✖	

Propagation of `alldifferent`



$$D = 7$$

Propagation of equality

$$91 * 5 + 10 * 8 + 7 = 90 * 6 + Y^{2..3}$$

$$\underbrace{91 * 5 + 10 * 8 + 7}_{542} = \underbrace{90 * 6 + Y^{2..3}}_{542..543}$$

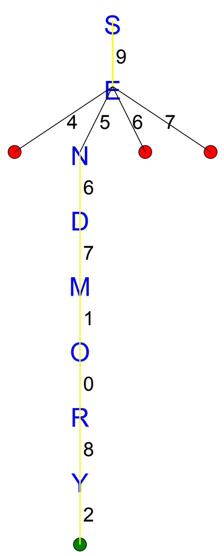
$$\underbrace{91 * 5 + 10 * 8 + 7 = 90 * 6 + Y^{2..3}}_{542}$$

$$Y = 2$$

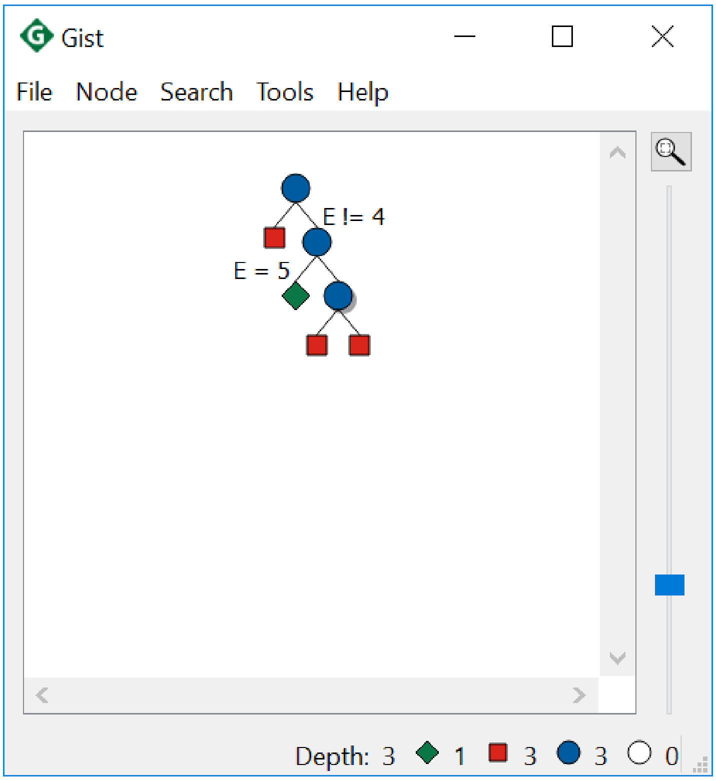
Last propagation step

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y			✹	-						

Complete Search Tree



Search Tree with Gecode/GIST



Some Differences

- Uses Binary branching
- Solutions in green, failure leafs in red, internal nodes in blue
- By default, shows all failed sub trees collapsed
- By default, uses different search strategy

Solution

	9	5	6	7
+	1	0	8	5
1	0	6	5	2

Points to Remember

- Constraint models are expressed by variables and constraints.
- Problems can have many different models, which can behave quite differently. Choosing the best model is an art.
- Constraints can take many different forms.
- Propagation deals with the interaction of variables and constraints.
- It removes some values that are inconsistent with a constraint from the domain of a variable.
- Constraints only communicate via shared variables.

Points to Remember

- Propagation usually is not sufficient, search may be required to find a solution.
- Propagation is data driven, and can be quite complex even for small examples.
- The default search uses chronological depth-first backtracking, systematically exploring the complete search space.
- The search choices and propagation are interleaved, after every choice some more propagation may further reduce the problem.

Part II

Global Constraints

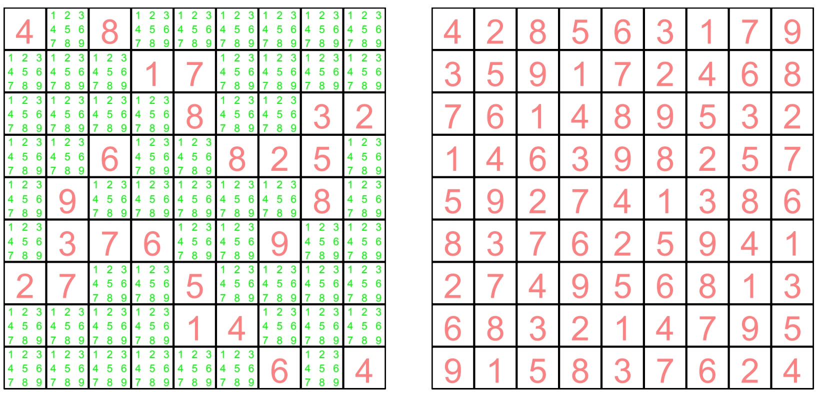
Example 2: Sudoku

- Global Constraints
  - Powerful modelling abstractions
  - Non-trivial propagation
  - Different consistency levels
- Example: Sudoku puzzle

Problem Definition

Sudoku

Fill in numbers from 1 to 9 so that each row, column and block contain each number exactly once



Model

- A variable for each cell, ranging from 1 to 9
- A 9x9 matrix of variables describing the problem
- Preassigned integers for the given hints
- `alldifferent` constraints for each row, column and 3x3 block

Reminder: `alldifferent`

- Argument: list of variables
- Meaning: variables are pairwise different
- Reasoning: Forward Checking (FC)
  - When variable is assigned to value, remove the value from all other variables
  - If a variable has only one possible value, then it is assigned
  - If a variable has no possible values, then the constraint fails
  - Constraint is checked whenever one of its variables is assigned
  - Equivalent to decomposition into binary disequality constraints

Main Program

```
int: s;
int: n=s*s;
array[1..n,1..n] of var 1..n: puzzle;
include "sudoku.dzn";
predicate alldifferent(array[int] of var int: x) =
  forall(i,j in index_set(x) where i < j)
    (x[i] != x[j]);
constraint forall(i in 1..n)
    (alldifferent([puzzle[i,j]| j in 1..n]));
constraint forall(j in 1..n)
    (alldifferent([ puzzle[i,j]| i in 1..n]));
constraint forall(i,j in 1..s)
    (alldifferent([puzzle[s*(i-1)+p, s*(j-1)+q]|
                  p,q in 1..s]));
solve satisfy;
```


Main Program Output

```
output [ "sudoku:\n" ] ++
  [ show(puzzle[i,j]) ++
    if j = N then
       if i mod S = 0 /\ i < N then "\n\n"
       else "\n"
       endif
     else
       if j mod S = 0 then "  "
       else " "
       endif
     endif
    | i,j in 1..N ];
```

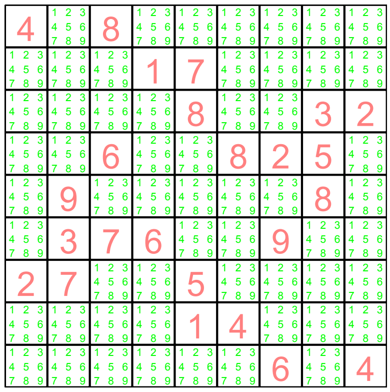
Main Program Data

```
s=3;
puzzle=[|
4, _, 8, _, _, _, _, _, _|
_, _, _, 1, 7, _, _, _, _|
_, _, _, _, 8, _, _, 3, 2|
_, _, 6, _, _, 8, 2, 5, _|
_, 9, _, _, _, _, _, 8, _|
_, 3, 7, 6, _, _, 9, _, _|
2, 7, _, _, 5, _, _, _, _|
_, _, _, _, 1, 4, _, _, _|
_, _, _, _, _, _, 6, _, 4|
|];
```

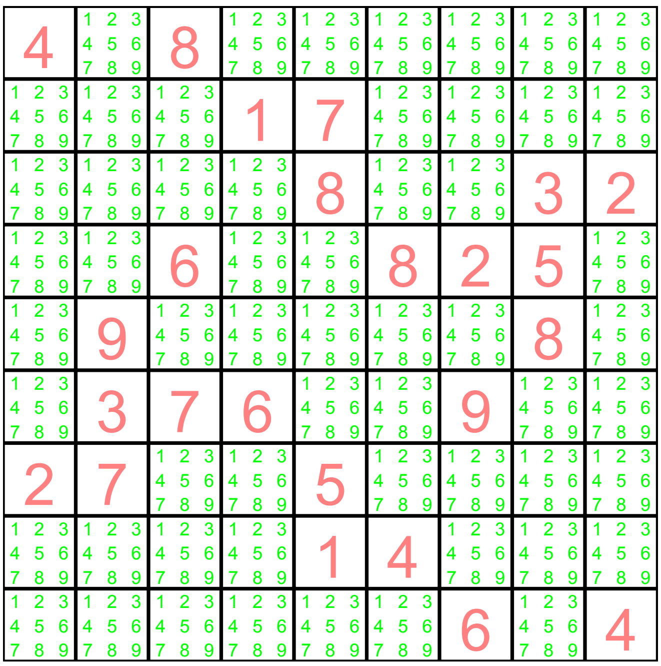

Running sudoku_decompose.mzn

Domain Visualizer

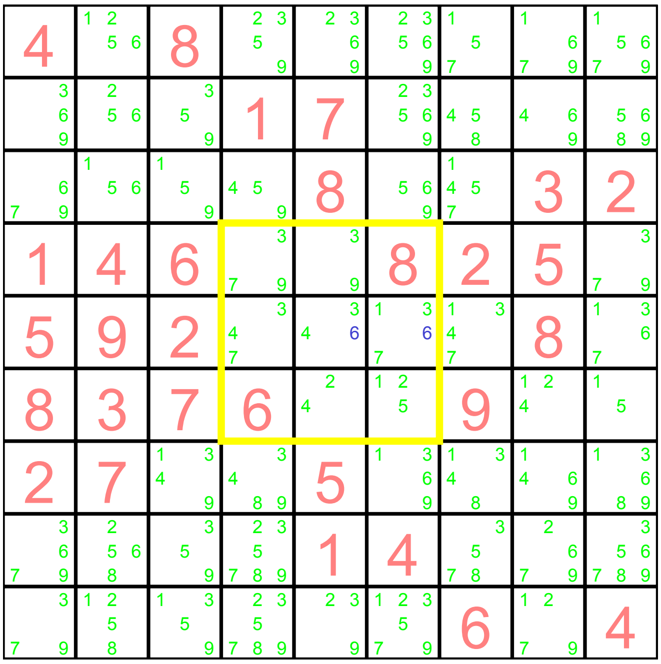
- Problem shown as matrix
- Each cell corresponds to a variable
- Instantiated: Shows integer value (large)
- Uninstantiated: Shows values in domain



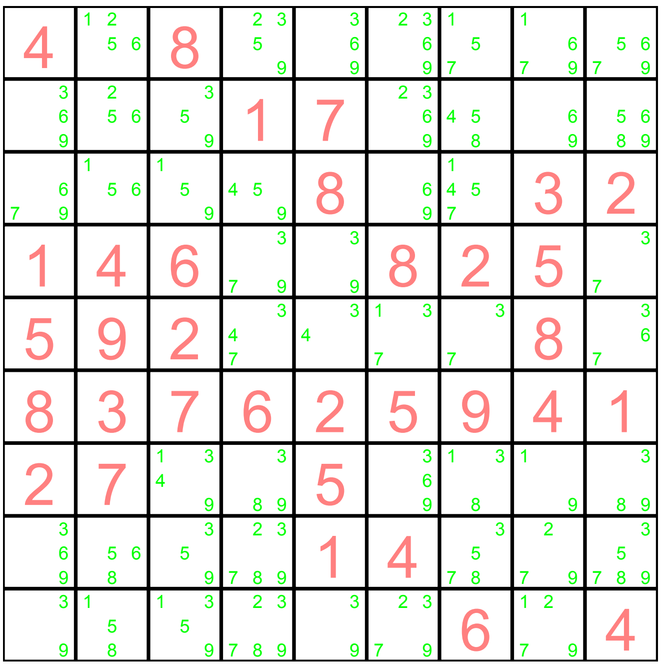
Initial State (Forward Checking)



Propagation Steps (Forward Checking)



After Setup (Forward Checking)



Can we do better?

- The alldifferent constraint is missing propagation
  - How can we do more propagation?
  - Do we know when we derive all possible information from the constraint?
- Constraints only interact by changing domains of variables

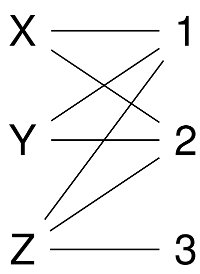
A Simpler Example

```
include "alldifferent.mzn";

var 1..2:X;
var 1..2:Y;
var 1..3:Z;

constraint alldifferent([X,Y,Z]);

solve satisfy;
```

Using Forward Checking

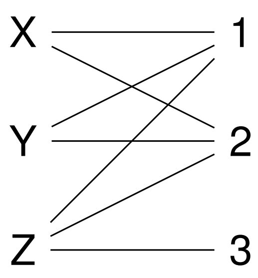
- No variable is assigned
- No reduction of domains
- But, values 1 and 2 can be removed from Z
- This means that Z is assigned to 3

Visualization of alldifferent as Graph



- Show problem as graph with two types of nodes
  - Variables on the left
  - Values on the right
- If value is in domain of variable, show link between them
- This is called a *bipartite* graph

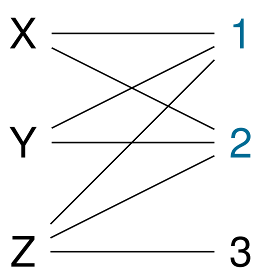
A Simpler Example



Value Graph for

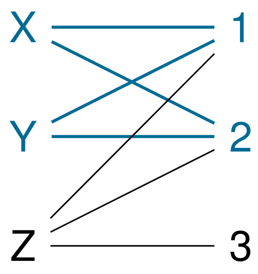
```
var 1..2:X;
var 1..2:Y;
var 1..3:Z;
```


A Simpler Example



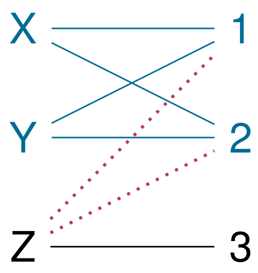
Check interval [1,2]

A Simpler Example



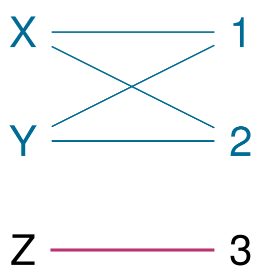
- Find variables completely contained in interval
- There are two: X and Y
- This uses up the capacity of the interval

A Simpler Example



No other variable can use that interval

A Simpler Example



Only one value left in domain of Z, this can be assigned

Idea (Hall Intervals)

- Take each interval of possible values, say size N
- Find all K variables whose domain is completely contained in interval
- If $K > N$ then the constraint is infeasible
- If $K = N$ then no other variable can use that interval
- Remove values from such variables if their bounds change
- If $K < N$ do nothing
- Re-check whenever domain bounds change

Implementation

- Problem: Too many intervals ($O(n^2)$) to consider
- Solution:
  - Check only those intervals which update bounds
  - Enumerate intervals incrementally
  - Starting from lowest(highest) value
  - Using sorted list of variables
- Complexity: $O(n \log(n))$ in standard implementations
- Important: Only looks at min/max bounds of variables

Bounds Consistency

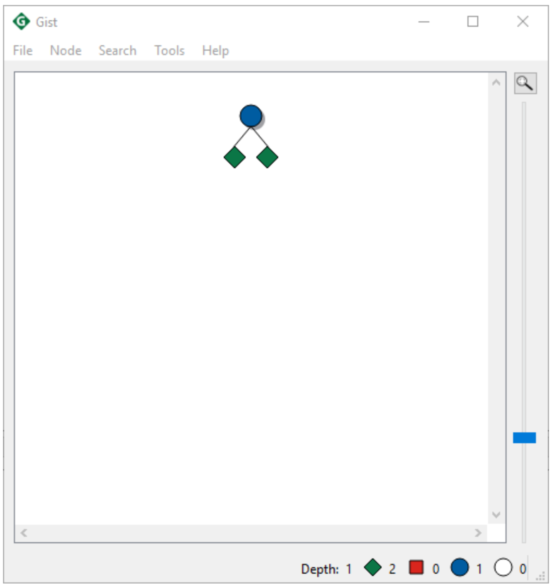
Definition

A constraint achieves *bounds consistency*, if for the lower and upper bound of every variable, it is possible to find values for all other variables between their lower and upper bounds which satisfy the constraint.

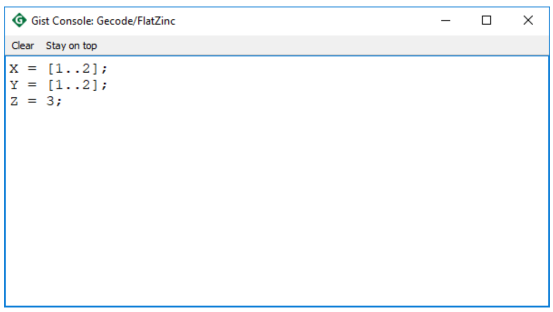
Annotation: :: bounds

```
include "alldifferent.mzn";

var 1..2:X;
var 1..2:Y;
var 1..3:Z;

constraint alldifferent([X,Y,Z]) :: bounds;
solve satisfy;
```


Running with Gecode Gist



All Solutions



Node Inspector (Root)

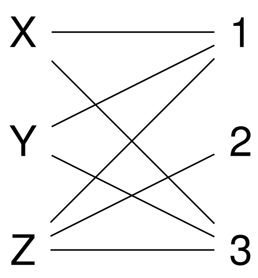
Can we do even better?

- Bounds consistency only considers min/max bounds
- Ignores “holes” in domain
- Sometimes we can improve propagation looking at those holes

Another Simple Example

```
include "alldifferent.mzn";

var {1,3}:X; % note enumerated domain
var {1,3}:Y;
var 1..3:Z; % note domain as interval

% annotated constraint
constraint alldifferent([X,Y,Z]) :: bounds;
solve satisfy;
```

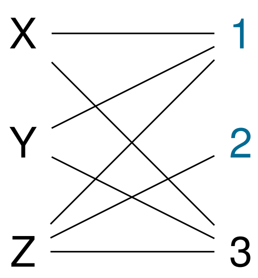
Another Simple Example



Value Graph for

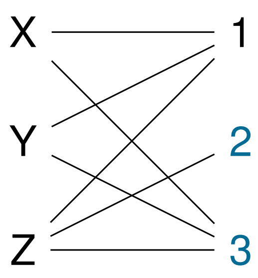
```
var {1,3}:X;
var {1,3}:Y;
var 1..3:Z;
```


Another Simple Example



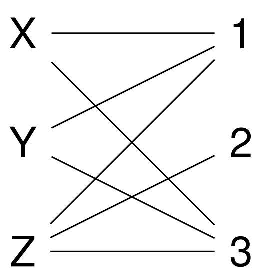
- Check interval [1,2]
- No domain of a variable completely contained in interval
- No propagation

Another Simple Example



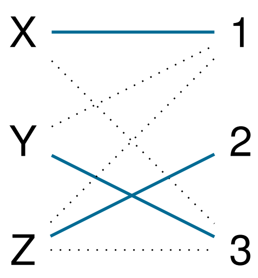
- Check interval [2,3]
- No domain of a variable completely contained in interval
- No propagation

Another Simple Example



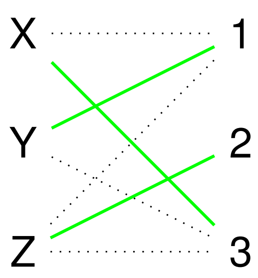
But, more propagation is possible, there are only two solutions

Another Simple Example



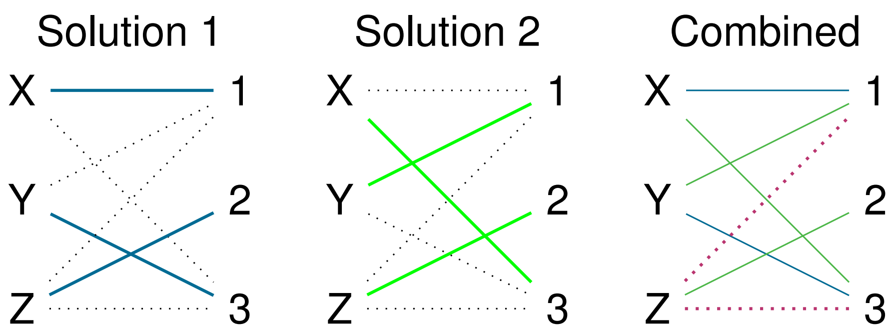
Solution 1: assignment in blue

Another Simple Example



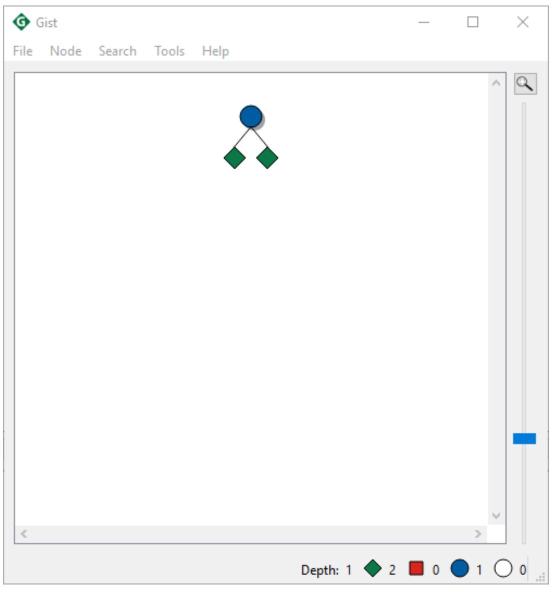
Solution 2: assignment in green

Another Simple Example

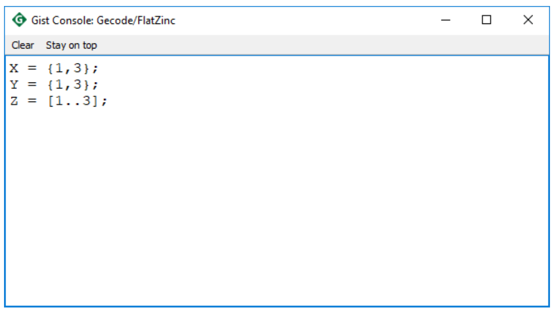


Combining solutions shows that Z=1 and Z=3 are not possible. Can we deduce this without enumerating solutions?

Bounds Consistency with Gecode Gist: No Propagation



All Solutions



Node Inspector (Root)

Solutions and Maximal Matchings

- A *Matching* is subset of edges which do not coincide in any node
- No matching can have more edges than number of variables
- Every solution corresponds to a *maximal matching* and vice versa
- If a link does not belong to some maximal matching, then it can be removed

Implementation

- Possible to compute all links which belong to some matching
  - Without enumerating all of them!
- Enough to compute **one** maximal matching
- Requires algorithm for *strongly connected components*
- Extra work required if more values than variables
- All links (values in domains) which are not supported can be removed
- Complexity: $O(n^{1.5}d)$

Domain Consistency

Definition

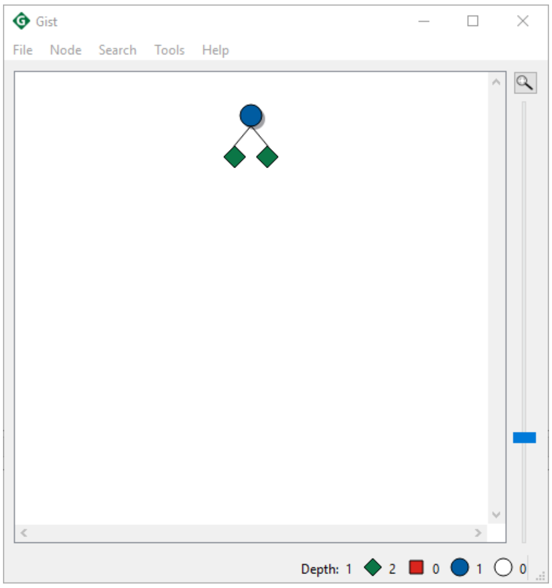
A constraint achieves *domain consistency*, if for every variable and for every value in its domain, it is possible to find values in the domains of all other variables which satisfy the constraint.

- Also called *generalized arc consistency (GAC)*
- or *hyper arc consistency*

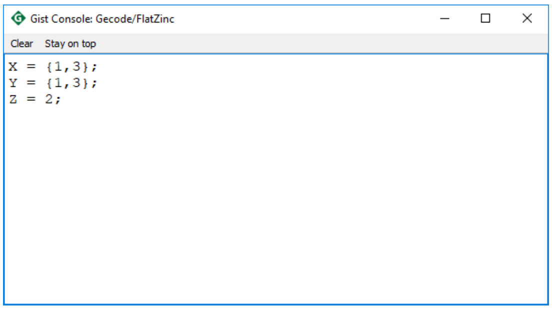
Simple Example Revisited

```
include "alldifferent.mzn";

var {1,3}:X; % note enumerated domain
var {1,3}:Y;
var 1..3:Z; % note domain as interval

% note different annotation
constraint alldifferent([X,Y,Z]) :: domain;
solve satisfy;
```

Domain Consistency with Gecode Gist: Propagation



All Solutions



Node Inspector (Root)

Can we still do better?

- NO! This extracts all information from this one constraint
- We could perhaps improve speed, but not propagation
- But possible to use different model
- Or model interaction of multiple constraints

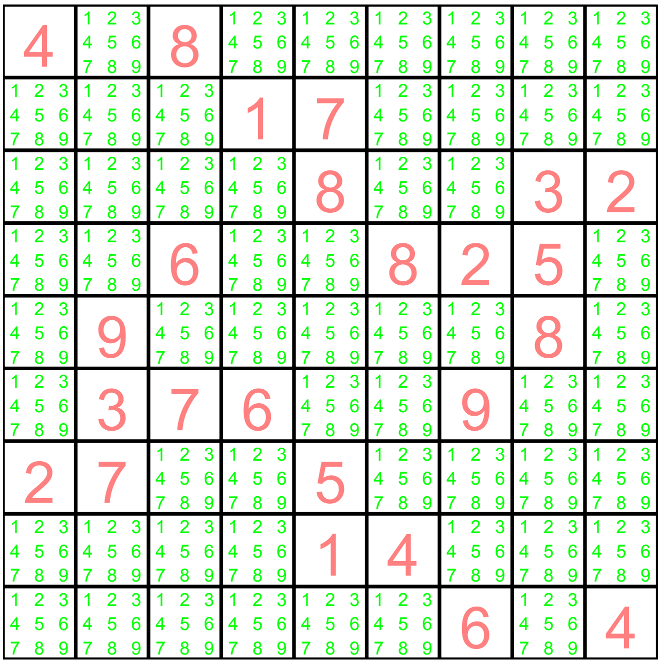
Should all constraints achieve domain consistency?

- Domain consistency is usually more expensive than bounds consistency
  - Overkill for simple problems
  - Nice to have choices
- For some constraints achieving domain consistency is NP-hard
  - We have to live with more restricted propagation

Main Program

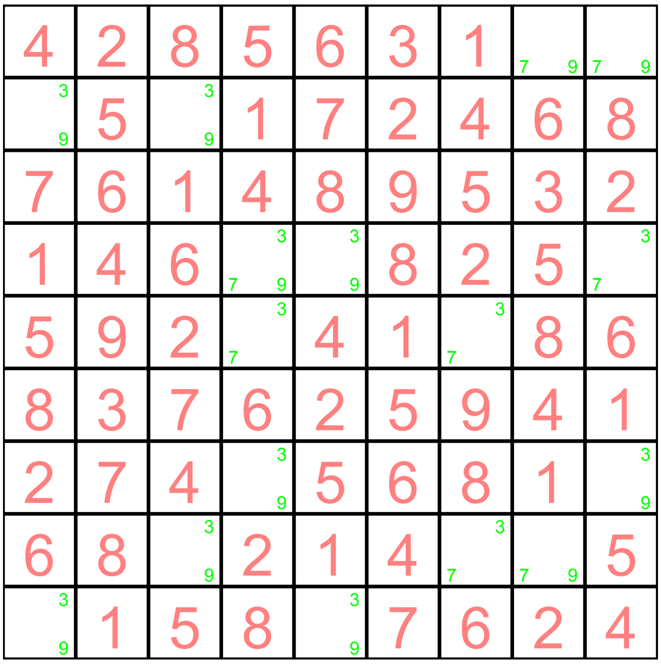
```
int: s;
int: n=s*s;
array[1..n,1..n] of var 1..n: puzzle;
include "sudoku.dzn";

include "alldifferent.mzn";
constraint forall(i in 1..n)
    (alldifferent([puzzle[i,j]| j in 1..n]));
constraint forall(j in 1..n)
    (alldifferent([ puzzle[i,j]| i in 1..n]));
constraint forall(i,j in 1..s)
    (alldifferent([puzzle[s*(i-1)+p, s*(j-1)+q]|
                   p,q in 1..s]));

solve satisfy;
```

Initial State (Domain Consistency)



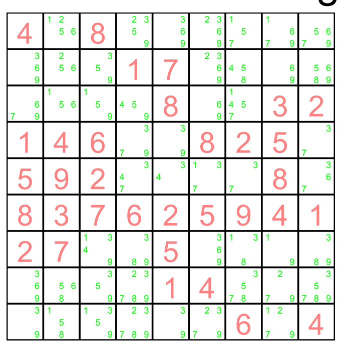
Propagation Steps (Domain Consistency)

After Setup (Domain Consistency)

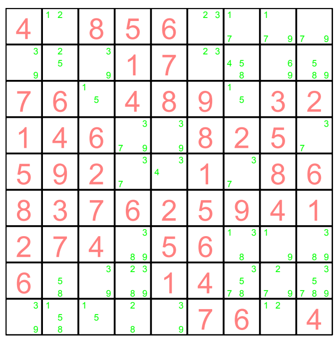


Comparison

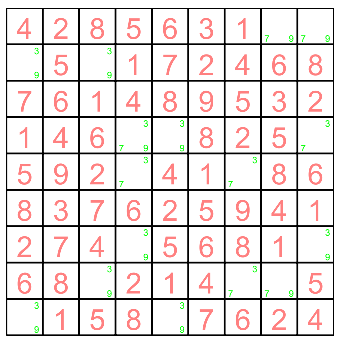
Forward Checking



Bounds Consistency



Domain Consistency



Typical?

- This does not always happen
- Sometimes, two methods produce same amount of propagation
- Possible to predict in certain special cases
- In general, tradeoff between speed and propagation
- Not always fastest to remove inconsistent values early
- But often required to find a solution at all

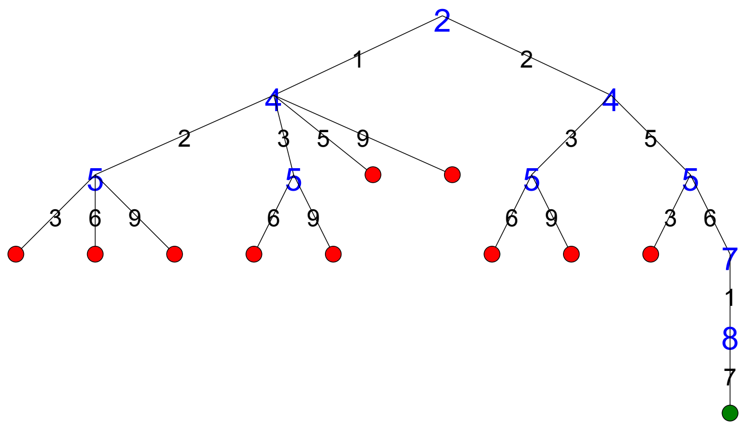
Simple search routine

- Enumerate variables in given order
- Try values starting from smallest one in domain
- Complete, chronological backtracking
- Advantage: Results can be compared with each other
- Disadvantage: Usually not a very good strategy

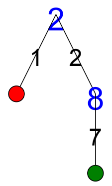
Forcing Naive Search

```
solve :: int_search(
  puzzle,
  input_order,
  indomain_min)
satisfy;
```


Search Tree (Forward Checking)



Search Tree (Bounds Consistency)



Search Tree (Domain Consistency)

Trading Propagation Against Search

- If we perform more propagation, search is more constrained
- Fewer values left, fewer alternatives to explore in search
- Best compromise is not obvious
- But can be learned from examples or during search
- Annotations are optional

Are there other Global Constraints?

- alldifferent is the most commonly used constraint
- Propagation methods can be explained
- But there are many more

Global Constraint Catalog

- `https://sofdem.github.io/gccat/`
- Description of 354 global constraints, 2800 pages
- Not all of them are widely used
- Detailed, meta-data description of constraints in Prolog

Families of Global Constraints

- Value Counting
  - alldifferent, global cardinality
- Scheduling
  - cumulative
- Properties of Sequences
  - sequence, no_valley
- Graph Properties
  - circuit, tree

Common Algorithmic Techniques

- Flow Based Algorithms (see talk on Tuesday)
- Automata
- Task Intervals
- Reduced Cost Filtering
- Decomposition

Part III

Customizing Search

What we want to introduce

- Importance of search strategy, constraints alone are not enough
- Two schools of thought
  - Black-box solver, solver decides by itself
  - Human control over process
- Dynamic variable ordering exploits information from propagation
- Variable and value choice
- Hard to find strategy which works all the time
- `int_search` annotation, simple search abstraction
- `seq_search` and `priority_search`, add flexibility
- Different way of improving stability of search routine

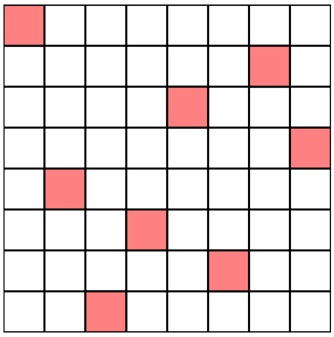
Example Problem

- N-Queens puzzle
- Rather weak constraint propagation
- Many solutions, limited number of symmetries
- Easy to scale problem size

Problem Definition

8-Queens

Place 8 queens on an 8×8 chessboard so that no queen attacks another. A queen attacks all cells in horizontal, vertical and diagonal direction. Generalizes to boards of size $N \times N$.



Solution for board size 8×8

Basic Model

- Cell based Model
  - A 0/1 variable for each cell to say if it is occupied or not
  - Constraints on rows, columns and diagonals to enforce no-attack
  - N^2 variables, $6N - 2$ constraints
- Column (Row) based Model
  - A 1..N variable for each column, stating position of queen in the column
  - Based on observation that each column must contain exactly one queen
  - N variables, $N^2/2$ binary constraints

Model

$$\text{assign} \quad [X_1, X_2, ...X_N]$$

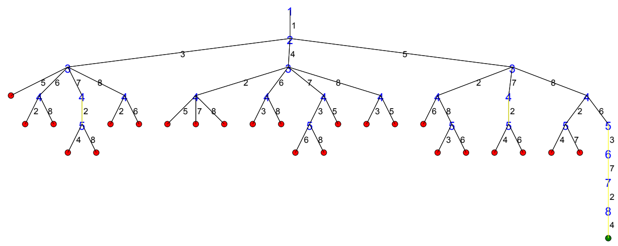
s.t.

$$\begin{aligned} \forall 1 \le i \le N: &\quad X_i \in 1..N \\ \forall 1 \le i < j \le N: &\quad X_i \neq X_j \\ \forall 1 \le i < j \le N: &\quad X_i + j \neq X_j + i \\ \forall 1 \le i < j \le N: &\quad X_i + i \neq X_j + j \end{aligned}$$

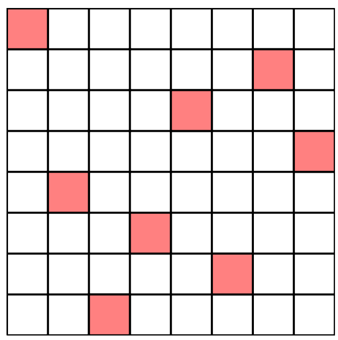
MiniZinc Program

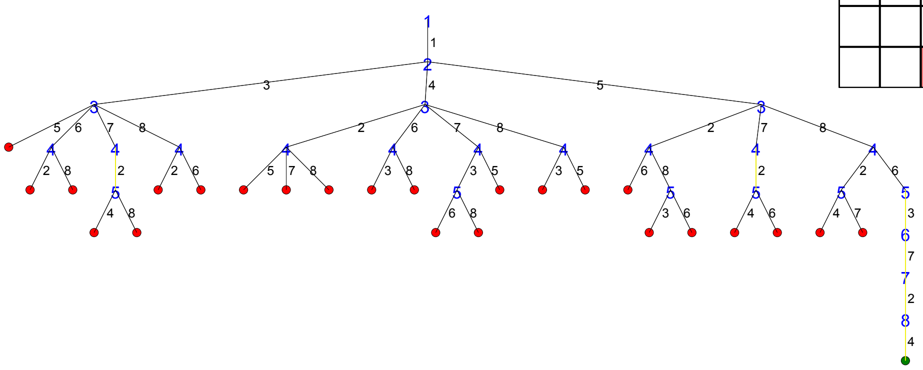
```
int: n=8;
array[1..n] of var 1..n: queens;
constraint
    forall(i, j in 1..n where i < j) (
         queens[i] != queens[j] /\
         queens[i] + i != queens[j] + j /\
         queens[i] - i != queens[j] - j
    )
 ;
solve :: int_search(
        queens,
        input_order,
        indomain_min)
        satisfy;
```

Default Strategy



First Solution

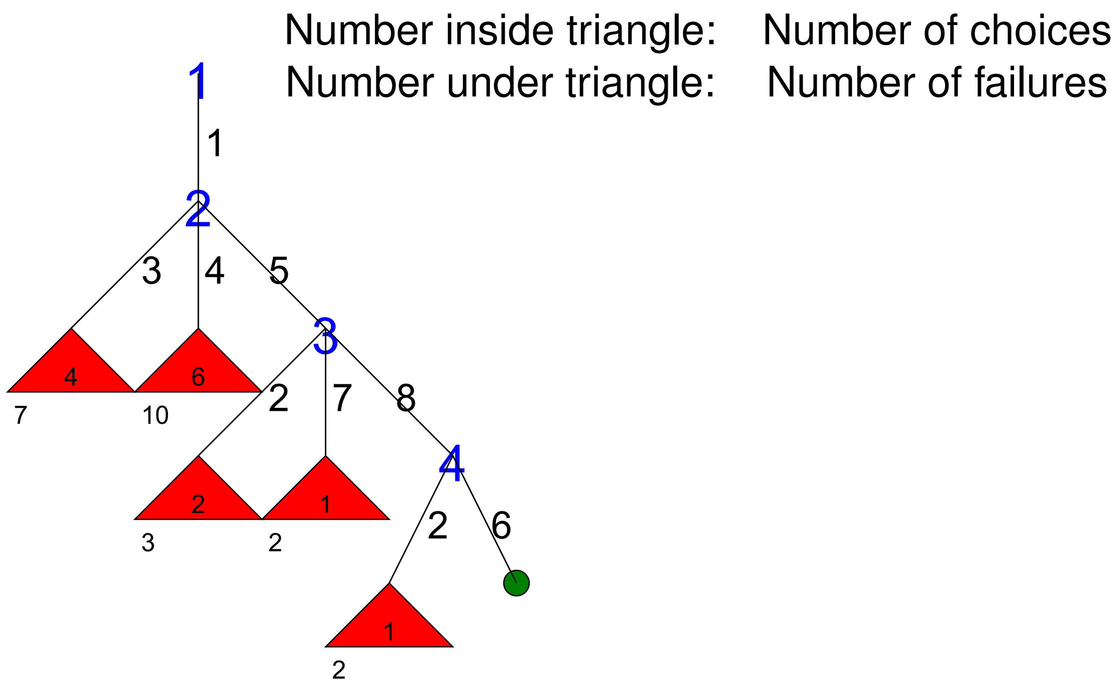




Observations

- Even for small problem size, tree can become large
- Not interested in all details
- Ignore all automatically fixed variables
- For more compact representation abstract failed sub-trees

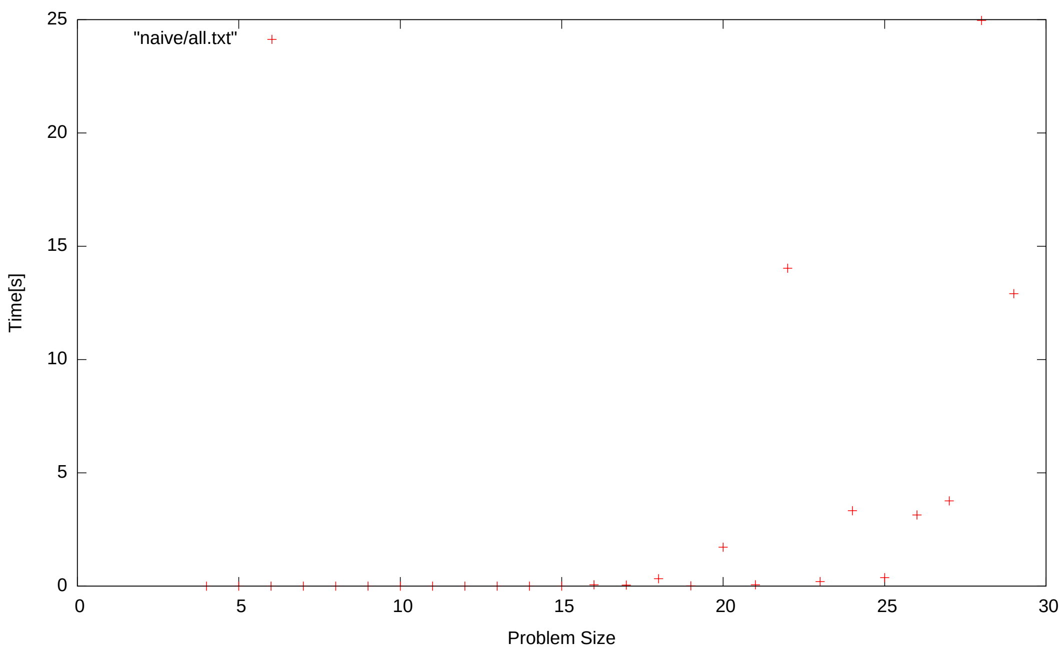
Compact Representation



Exploring other board sizes

- How stable is the model?
- Try all sizes from 4 to 100
- Timeout of 100 seconds

Naive Stategy, Problem Sizes 4-100



Observations

- Time very reasonable up to size 20
- Sizes 20-30 times very variable
- Not just linked to problem size
- No size greater than 30 solved within timeout

Possible Improvements

- Better constraint reasoning
  - Remodelling problem with 3 `alldifferent` constraints
  - Global reasoning as described before
  - Not explored here
- Better control of search
  - Static vs. dynamic variable ordering
  - Better value choice
  - Not using complete depth-first chronological backtracking

Static vs. Dynamic Variable Ordering

- Heuristic Static Ordering
  - Sort variables before search based on heuristic
  - Most important decisions
  - Smallest initial domain
- Dynamic variable ordering
  - Use information from constraint propagation
  - Different orders in different parts of search tree
  - Use all information available

First Fail strategy

- Dynamic variable ordering
- At each step, select variable with smallest domain
- Idea: If there is a solution, better chance of finding it
- Idea: If there is no solution, smaller number of alternatives
- Needs tie-breaking method

Modified MiniZinc Program

```
int: n=8;
array[1..n] of var 1..n: queens;
constraint
    forall(i, j in 1..n where i < j) (
         queens[i] != queens[j] /\
         queens[i] + i != queens[j] + j /\
         queens[i] - i != queens[j] - j
    )
 ;
solve :: int_search(
        queens,
        first_fail,
        indomain_min)
        satisfy;
```


Variable Choice

- Determines the order in which variables are assigned
- `input_order` assign variables in static order given
- `smallest` assign variable with smallest value in domain first
- `first_fail` select variable with smallest domain first
- `dom_w_deg` consider ratio of domain size and failure count
- Others, including programmed selection for specific solvers

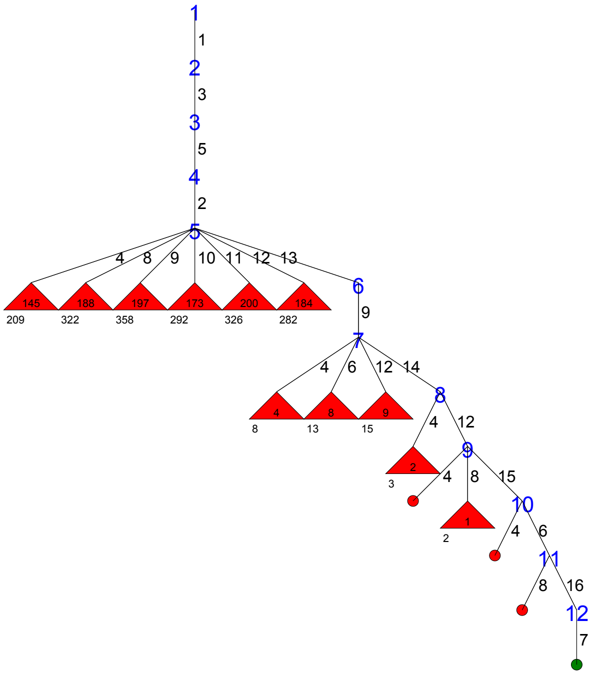
Value Choice

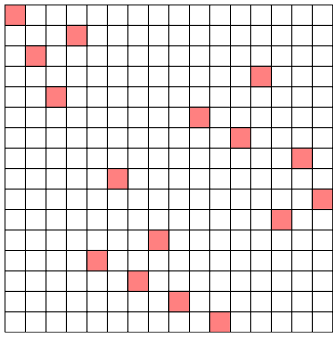
- Determines the order in which values are tested for selected variables
- `indomain_min` Start with smallest value, on backtracking try next larger value
- `indomain_median` Start with value closest to middle of domain
- `indomain_random` Choose values in random order
- `indomain_split` Split domain into two intervals

Comparison

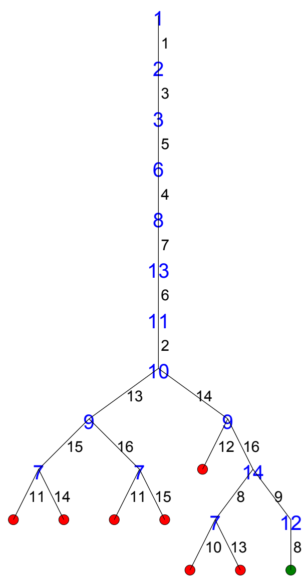
- Board size 16x16
- Naive (Input Order) Strategy
- First Fail variable selection

Naive (Input Order) Strategy (Size 16)

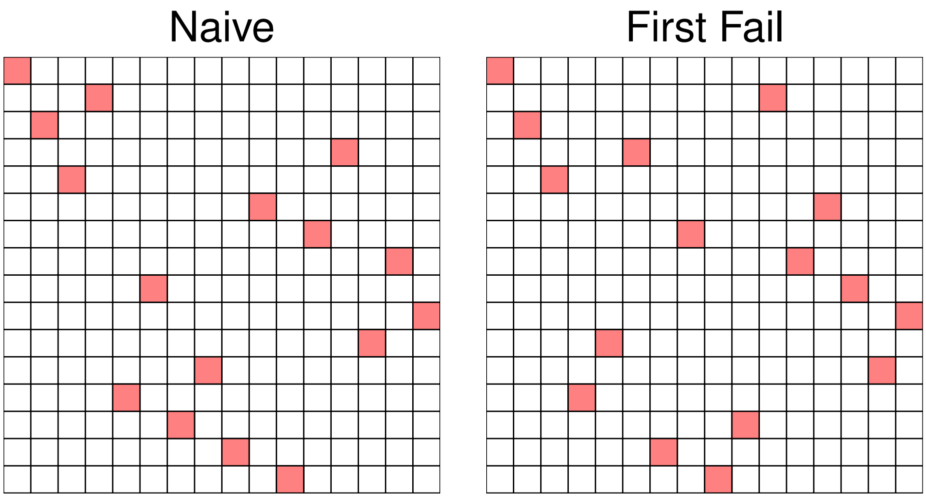




FirstFail Strategy (Size 16)

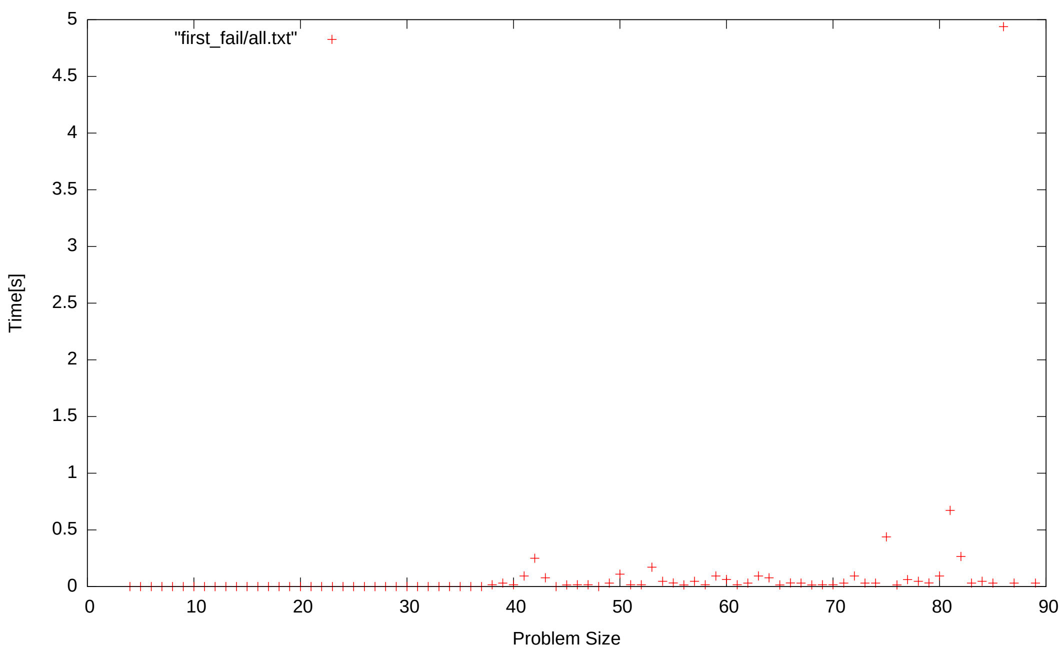


Comparing Solutions



Solutions are different!

FirstFail, Problem Sizes 4-100



Observations

- This is much better
- But some sizes are much harder
- Timeout for sizes 88, 91, 93, 97, 98, 99

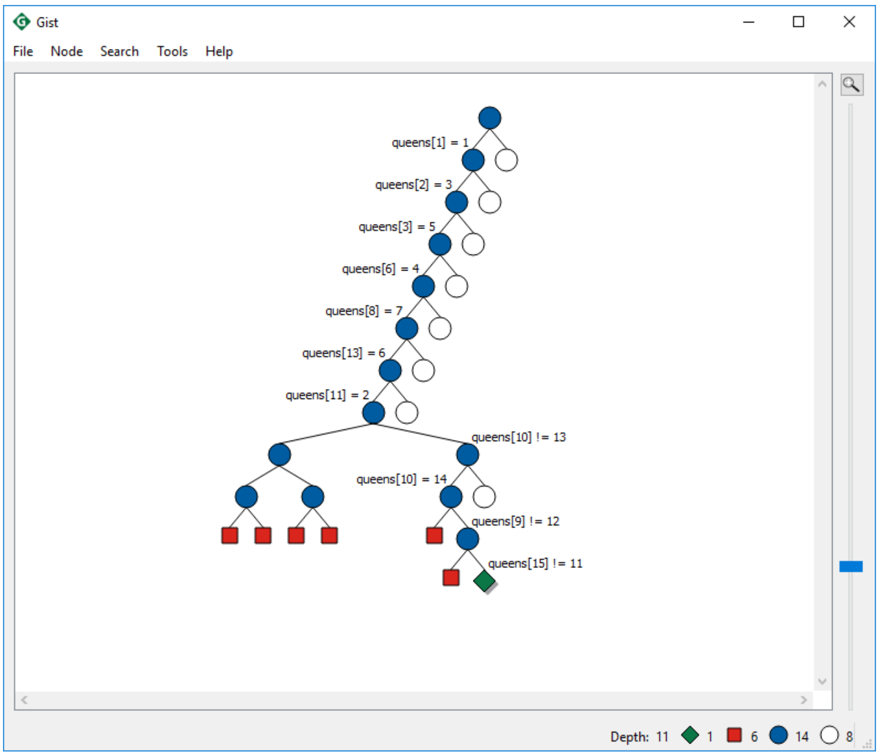
More Reactive Variable Selection

- Domain size is important, but other information is useful as well
- Dom/Weighted Degree: better results in many situations
- Weight Degree: count how often variable has been involved in failure
- Focus on more complicated part of problem
- Changes during search, learns from past performance
- Option **dom_w_deg**

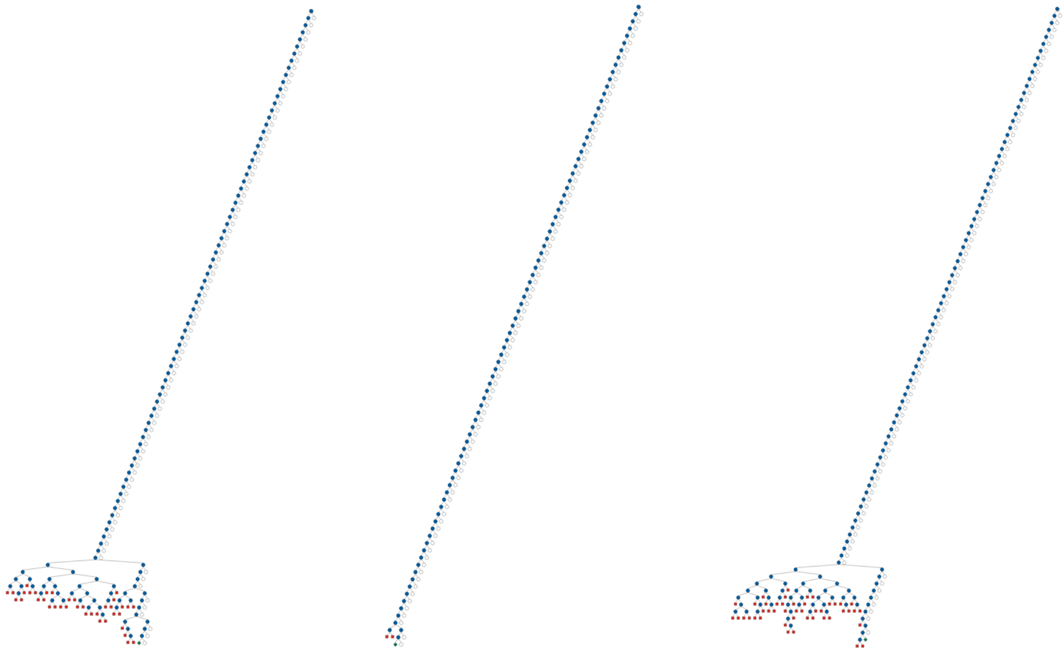
Weighted Degree Variable Selection

```
int: n=8;
array[1..n] of var 1..n: queens;
constraint
    forall(i, j in 1..n where i < j) (
         queens[i] != queens[j] /\
         queens[i] + i != queens[j] + j /\
         queens[i] - i != queens[j] - j
    )
 ;
solve :: int_search(
        queens,
        dom_w_deg,
        indomain_random)
        satisfy;
```


Result for size 16 with Gecode-Gist



Sample Results for Larger Sizes



Size 93 Size 94 Size 95

Approach 1: Heuristic Portfolios

- Try multiple strategies for the same problem
- With multi-core CPUs, run them in parallel
- Only one needs to be successful for each problem

Approach 2: Restart with Randomization

- Only spend limited number of backtracks for a search attempt
- When this limit is exceeded, restart at beginning
- Requires randomization to explore new search branch
- Randomize variable choice by random tie break
- Randomize value choice by shuffling values
- Needs strategy when to restart

Random Variable Choice and Restarts

```
int: n=8;
array[1..n] of var 1..n: queens;
constraint
    forall(i, j in 1..n where i < j) (
         queens[i] != queens[j] /\
         queens[i] + i != queens[j] + j /\
         queens[i] - i != queens[j] - j
    )
 ;
solve :: int_search(
        queens,
        dom_w_deg,
        indomain_random)
        :: random_linear(100)
        satisfy;
```

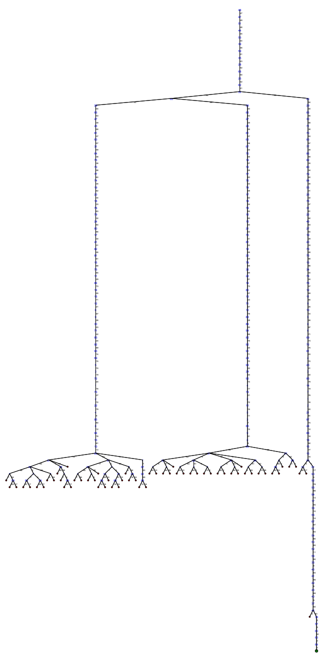
Approach 3: Partial Search

- Abandon depth-first, chronological backtracking
- Don't get locked into a failed sub-tree
- A wrong decision at a level is not detected, and we have to explore the complete subtree below to undo that wrong choice
- Explore more of the search tree
- Spend time in promising parts of tree

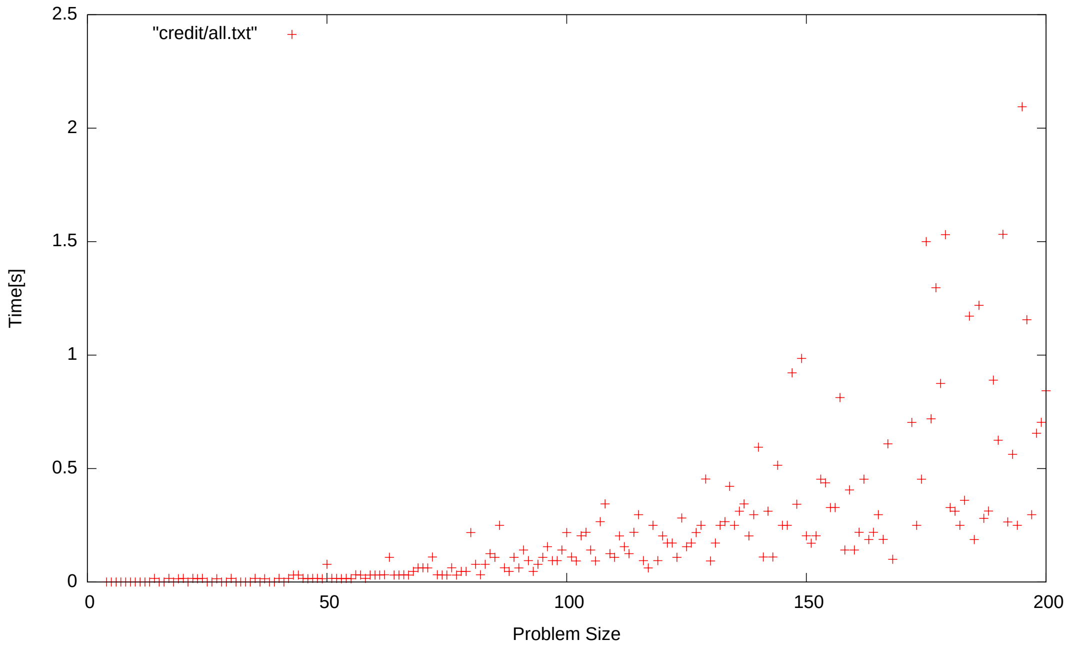
Example: Credit Search

- Not available in all solvers
- Explore top of tree completely, based on credit
- Start with fixed amount of credit
- Each node consumes one credit unit
- Split remaining credit amongst children
- When credit runs out, start bounded backtrack search
- Each branch can use only K backtracks
- If this limit is exceeded, jump to unexplored top of tree

Credit, Search Tree Problem Size 94



Credit, Problem Sizes 4-200



Dealing with Heterogeneous Variables

- `int_search` works when all variables represent the same concept
- e.g. the start of an activity
- It struggles if different variable sets denote different concepts
- eg. x and y dimension in a placement problem
- Two alternative search methods
  - `seq_search` do two searches, one after the other
  - `priority_search` interleave the assignment of the different variables

Seq_search

- Find first solution for one set of variables, then for another
- Simple form of problem decomposition
- Especially useful to control assignment of cost variables
- Often a risky, high pay-off strategy
  - If it works, it works very well
  - But if it does not work, it leads to very deep backtracking

Seq_search Example

```
solve ::seq_search([
    int_search(x,smallest,indomain_split),
    int_search(y,first_fail,indomain_split)])
    minimize objective;
```


Priority_search

- Often two sets of variables are linked with each other
- X and y coordinate of rectangle to place
- Time and location in time tabling
- Want to interleave assignment, e.g. fix x and y coordinate of one item before assigning the next
- Still want to use dynamic variables selection, based on properties of one of the variables
- Only available in Chuffed

Priority_search Example

```
include "chuffed.mzn";
solve ::priority_search(x,
       [int_search([x[i],y[i]],
        input_order,indomain_min)| i in T],
        smallest,complete)
    minimize objective;
```


Points to Remember

- Choice of search can have huge impact on performance
- Dynamic variable selection can lead to large reduction of search space
- Packaged search can do a lot, but programming search adds even more
- Depth-first chronologicial backtracking not always best choice
- How to control this explosion of search alternatives?

Part IV

What is missing?

Many Specialized Topics

- How to design efficient core engine
- Hybrids with LP/MIP tools
- Hybrids with SAT
- Symmetry breaking
- Use of MDD/BDD to encode sets of solutions
- High level modelling tools
- Debugging/visualization

Reformulation

- Just because the user has modelled it this way, it doesn't mean we have to solve it that way
  - Replace some constraint(s) by other, equivalent constraints
  - Because we don't have that constraint in our system
  - For performance

Learning

- While solving the problem we can learn how to strengthen the model/search
  - Understand which constraints/method contribute to propagation and change schedule
  - Learn no-good constraints by explaining failure
  - Adapt search strategy based on search experience