# MiniZinc Basics

Centre for
Data Analytics

Insight

Helmut Simonis

CRT-AI CP Week 2025

## Licence

# Acknowledgments

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 12/RC/2289-P2 at Insight the SFI Research Centre for Data Analytics at UCC, which is co-funded under the European Regional Development Fund.
A version of 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.



Part I

# MiniZinc Basics

# Objectives

- Understand MiniZinc IDE
- Bundled Solvers
- Basic Modelling in MiniZinc
- Some More Examples



# Outline

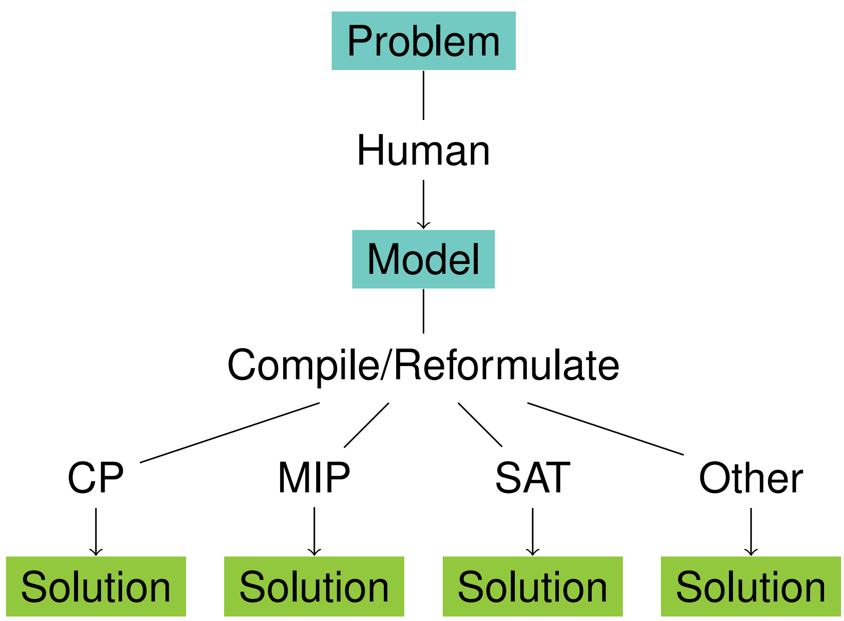
- MiniZinc Background
- IDE
- Elements of MiniZinc Programs
- Running Programs

# MiniZinc

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



# Framework Process



# Bundled Solvers

- Chuffed
- Coin-BC
- Gecode
- Gecode Gist
- (Cplex)
- (Gurobi)



# Chuffed

- Developed at Melbourne University/Monash
- Clause Learning FD Solver including SAT Reasoning
- Learns from failures
- Very successful in competitions

# Coin-BC

- Open Source MIP Solver
- Initially Developed at IBM
- Completely different from techniques described here
- Moderate performance (non commercial)



# Gecode

- Developed at KTH Stockholm
- Powerful C++ based solver
- Copying based solver design

## Gecode Gist

- Extension of Gecode to interactive use
- Explore search tree interactively
- Visualization of search tree
- Useful to understand behaviour



## Cplex/Gurobi

- Commercial MIP solvers
- Only interface bundled, needs installation on machine
- Two most successful MIP solvers at this time

## Others

- Many solvers can be used as back-end to Minizinc
- Need manual installation
- Not all specific functionality may be available



## Which to Choose?

- Difficult to state in general terms

# Demo



# Elements of MiniZinc

- Comments
- Parameters
- Variables
- Constraints
- Comprehensions
- Solve

# Comments

```
% comments rest of line
/* comment here */
```



# Parameter

```
int: n = 8;
int: n; % set somewhere else
int: nrDays = 4;
set of int: days = 1..nrDays;
set of int: games = {1,3,5,7};
array[days] of int: mat;
array[days] of int: mat = [1,2,3,4];
```

# Variables

```
var 1..8:x;
array[days] of var games:y;
```



# Constraints

```
constraint x != y;
constraint 4*y[1]+5*y[2] = z;
% operators =,!=,<,>,<=,>=
constraint alldifferent(y);
% annotations ::bounds ::domain

constraint forall(game in games)
  (pDay[game] = mapDay[x[game]]);
```

# Defining Constraints

```
predicate exactly(array[int] of var int:x,
    int:count,int:value) =
  count = sum(i in index_set(x))(x[i] = value);
```



# Comprehensions

```
constraint alldifferent([pDay[i]|i in team1Games]);

forall(i in days)(x[i] != v);
```

# Solve

```
solve satisfy;
solve minimize(x);
solve maximize(x);
```



# Solve annotations

- ::int_search(vars,var_selection,value_selection)
- input_order,first_fail,smallest,dom_w_deg
- indomain_min,indomain_median,
- indomain_random,indomain_split

# Seq_search Example

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



# Priority_search Example

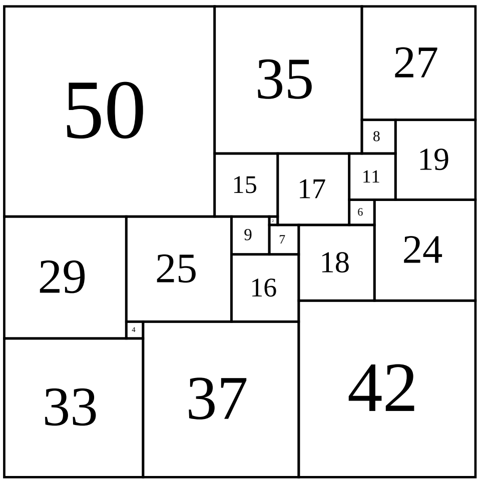
```
include “chuffed.mzn”;
solve ::priority_search(x,
       [int_search([x[i],y[i]],
        input_order,indomain_min)| i in T],
        smallest,complete)
    minimize objective;
```

# Square Placement

- Consider a set of square rectangles of different sizes
- Pack them into an enclosing square
- Total surface of squares to pack is equal to the available area
- Perfect problem: no subset forms rectangle
- Famous combinatorial problem, difficult to solve by hand
- `http://www.squaring.net/sq/ss/spss/spss.html`
- Link to William Tutte (Breaking the Lorenz machine code in WW II)
- Solved in 1978 by A.J.W. Duijvestijn



# Original Solution




21 : 112A AJD 1978

# Data

```
S = 1..21;
size =  [2,4,6,7,8,9,11,15,16,17,
         18,19,24,25,27,29,33,35,37,42,50];
box = 112;
```



# Square Packing Program (I)

```
include "globals.mzn";
set of int: S;
array[S] of int:size;
int: box;
include "squares.dzn";

array[S] of var 0..box:x;
array[S] of var 0..box:y;
```

# Square Packing Program (II)

```
constraint forall (i in S)
    (x[i]+size[i]<=box);
constraint forall (i in S)
    (y[i]+size[i]<=box);
constraint diffn(x,y,size,size);
constraint cumulative(x,size,size,box);
constraint cumulative(y,size,size,box);

solve satisfy;
```



# Solved with Chuffed

```
Running squares.mzn
x = array1d(1..21, [60, 33, 60, 53, 77, 53, 66, 62,
  37, 60, 42, 66, 42, 37, 85, 33, 0, 77, 0, 0, 62]);
y = array1d(1..21, [47, 79, 24, 42, 19, 49, 19, 47,
  42, 30, 24, 0, 0, 58, 0, 83, 79, 27, 42, 0, 62]);
-------
Finished in 4s 364msec
```

# Job Shop Scheduling

- Schedule a number of jobs
- Each job consists of a number of tasks
- Each task has a duration and must run on one specific machine
- Tasks of a job must be executed in sequence
- A machine can only work on one task as a time



# History

- 10x10 instance proposed by Fisher& Thompson
- Also known as 10x10 Muth & Thompson instance (1963)
- Stayed as open problem for 25 years
- Solved by Carlier and Pinson in 1989

# Job Shop Data (I)

```
nrJobs= 6;
nrRes= 6;
taskUse= [|
   2, 0, 1, 3, 5, 4|
   1, 2, 4, 5, 0, 3|
   2, 3, 5, 0, 1, 4|
   1, 0, 2, 3, 4, 5|
   2, 1, 4, 5, 0, 3|
   1, 3, 5, 0, 4, 2 |];
taskDuration= [|
   1, 3, 6, 7, 3, 6|
   8, 5, 10, 10, 10, 4|
   5, 4, 8, 9, 1, 7|
   5, 5, 5, 3, 8, 9|
   9, 3, 5, 4,3, 1|
   3, 3, 9, 10, 4, 1 |];
```



# Job Shop Data (II)

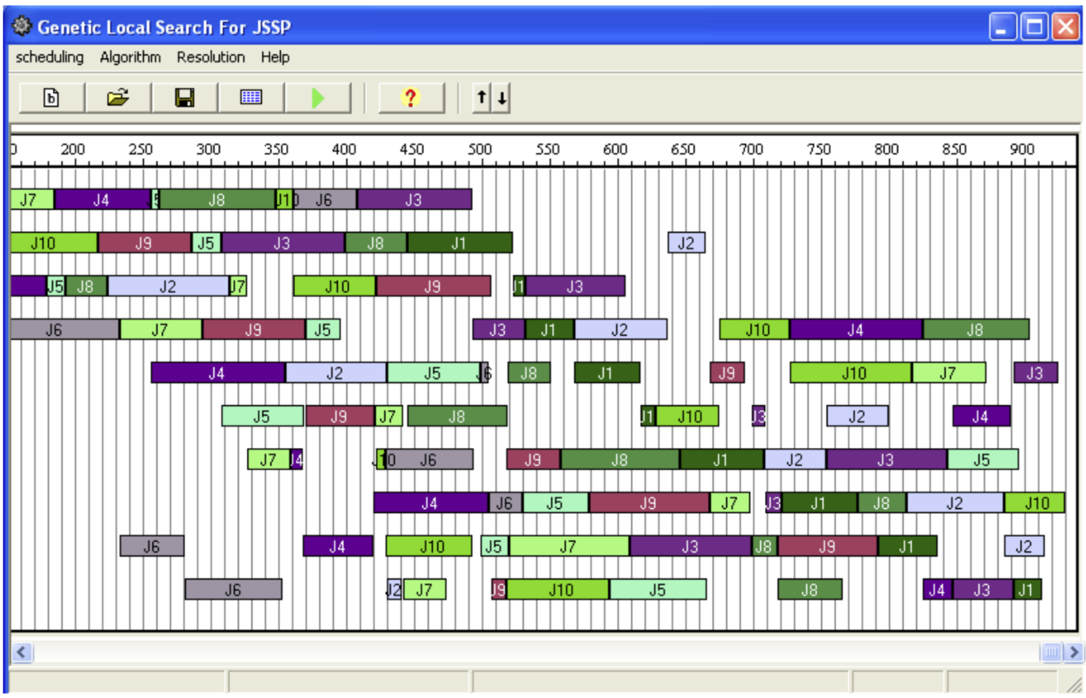
```
nrJobs= 10;
nrRes= 10;

taskUse= [|
0, 1, 2, 3, 4, 5, 6, 7, 8, 9|
0, 2, 4, 9, 3, 1, 6, 5, 7, 8|
1, 0, 3, 2, 8, 5, 7, 6, 9, 4|
1, 2, 0, 4, 6, 8, 7, 3, 9, 5|
2, 0, 1, 5, 3, 4, 8, 7, 9, 6|
2, 1, 5, 3, 8, 9, 0, 6, 4, 7|
1, 0, 3, 2, 6, 5, 9, 8, 7, 4|
2, 0, 1, 5, 4, 6, 8, 9, 7, 3|
0, 1, 3, 5, 2, 9, 6, 7, 4, 8|
1, 0, 2, 6, 8, 9, 5, 3, 4, 7 |];
```

# Job Shop Data (III)

```
taskDuration= [|
29, 78,  9, 36, 49, 11, 62, 56, 44, 21|
43, 90, 75, 11, 69, 28, 46, 46, 72, 30|
91, 85, 39, 74, 90, 10, 12, 89, 45, 33|
81, 95, 71, 99,  9, 52, 85, 98, 22, 43|
14,  6, 22, 61, 26, 69, 21, 49, 72, 53|
84,  2, 52, 95, 48, 72, 47, 65,  6, 25|
46, 37, 61, 13, 32, 21, 32, 89, 30, 55|
31, 86, 46, 74, 32, 88, 19, 48, 36, 79|
76, 69, 76, 51, 85, 11, 40, 89, 26, 74|
85, 13, 61,  7, 64, 76, 47, 52, 90, 45 |];
```



# Example Solution




screenshot from: A LOCAL SEARCH GENETIC ALGORITHM FOR THE JOB SHOP SCHEDULING PROBLEM Kebabla Mebarek, Mouss Leila Hayat and Mouss Nadia

# Job-Shop Program (I)

```
include "globals.mzn";
int:nrJobs;
int:nrRes;

set of int: J=1..nrJobs;
set of int: R=1..nrRes;

array[J,R] of int:taskUse;
array[J,R] of int:taskDuration;
include "mt06.dzn";
int:ub =sum(j in J,r in R)(taskDuration[j,r]);

array[J,R] of var 0..ub:start;
var 0..ub:objective;
```



# Job-Shop Program (II)

```
constraint forall(j in J)
    (objective >= start[j,nrRes]+
                  taskDuration[j,nrRes]);
constraint forall(j in J, r in 1..nrRes-1)
    (start[j,r+1] >= start[j,r]+
                     taskDuration[j,r]);
constraint forall (r in R)
(cumulative(
  [start[j,k]|j in J, k in R
    where taskUse[j,k]+1=r],
  [taskDuration[j,k]|j in J, k in R
    where taskUse[j,k]+1=r],
  [1|j in J, k in R
    where taskUse[j,k]+1=r],
  1)
);
```

# Job-Shop Program (III)

```
solve minimize objective;
```