University of
St Andrews

Towards Understanding Differences Between Modelling Pipelines: a Modelers Perspective

Csobán Balogh, **Ruth Hoffmann** and Joan Espasa

ModRef 2024, University of Girona

What and why?

From a (single) users perspective:

- Investigate the capabilities of constraint programming pipelines.
- Investigate the difference in languages.
- Comparing MiniZinc and Savile Row.

How?

- Create as close to equivalent models as possible.
- Using MiniZinc and Essence' (not Essence)
- Compare over the same solver (Chuffed).
- Compare over equivalent(ish) optimisation levels.
- 6 Models from different problem classes
  1. Quasigroup Completion, (no exciting differences)
  2. Wordpress Problem, (no exciting differences)
  3. Rotating Rostering Problem, (no exciting differences)
  4. Travelling Tournament Problem with Predefined Values,
  5. Multi-Skilled Project Scheduling Problem,
  6. Capacitated Vehicle Routing Problem with Time Windows.

`regular` (MZn) vs `forAll` (SR)

Traveling Tournament Problem with Predefined Venues at most three consecutive away or home games

- at most three consecutive away or home games

MZn `regular` asserts that a sequence of variables take a value from a finite automaton.

E' `forAll` checking that there are not four consecutive assignments.

`circuit` (MZn)

Capacitated Vehicle Routing problem with Time Windows, Service Times and Pickup and Deliveries.

- `circuit` is used to ensure the vehicle delivery routes do not take sub-tours in their route and visits each location uniquely for optimisation

MnZ A `circuit` is such that the cell value of an array points to the index of the next number, and this forms a circuit that continues around.

E' `https://github.com/MiniZinc/libminizinc/blob/master/share/minizinc/std/fzn_circuit.mzn`

Set Variables

Multi-Skilled Project Scheduling Problem

- Sets of skills, workers etc. (each assigned an integer)

MnZ Variables which are a set.

E' Occurrence representation of the integers/elements.

`letting` (MZn)

Multi-Skilled Project Scheduling Problem

- `letting` creates variables within constraints

```
let { set of int: WTasks =
        { i | i in Tasks where exists(k in has_skills[j])(rr[k, i] > 0) }
} in ...
let { set of int: TWorkers =
        { j | j in Workers where exists(k in has_skills[j])(rr[k, i] > 0) }
} in ...
```

```
forAll i : Tasks . forAll j : Workers .
    TWorkers[j, i] = 1 <->
        exists k : Skills . has_skills[j, k] = 1 /\ rr[k,i] > 0,
```

`cumulative` (MnZ)

Multi-Skilled Project Scheduling Problem

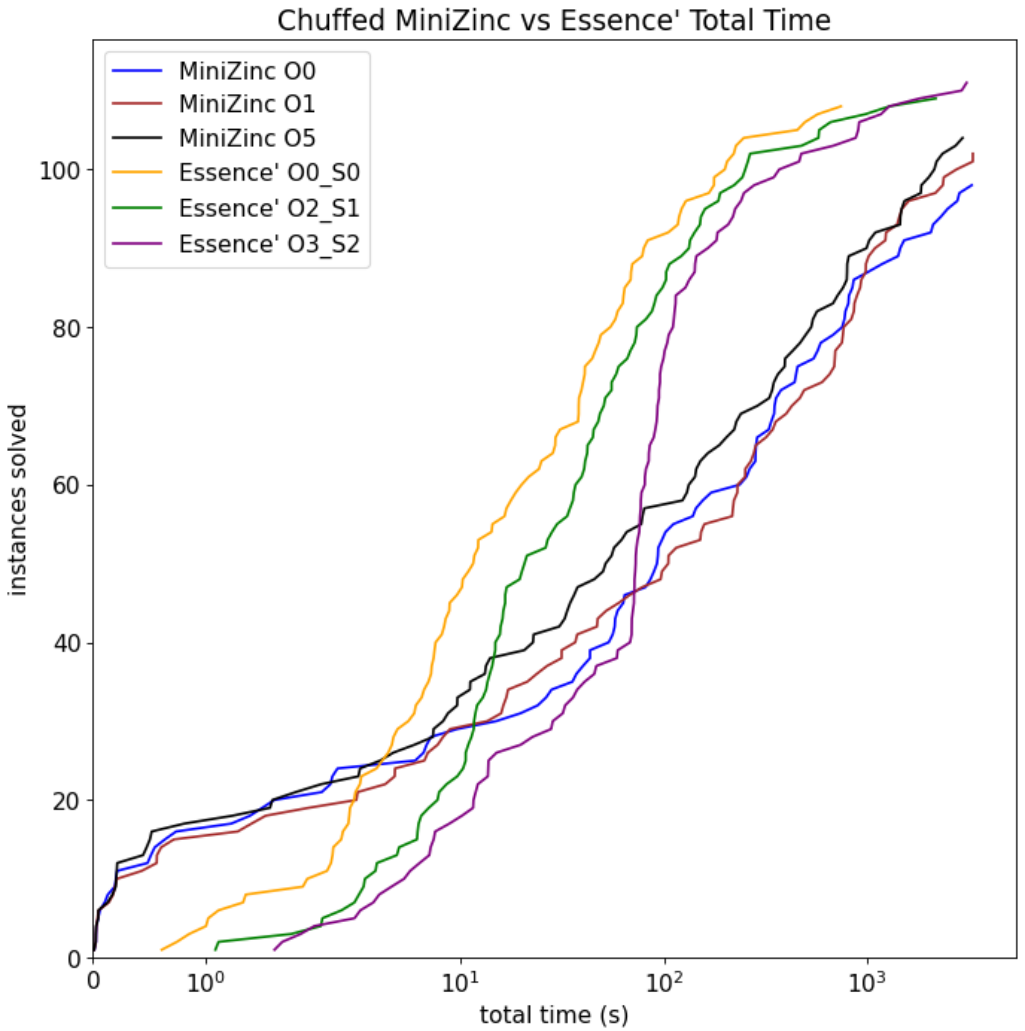
- Determines whether set of tasks with start times, durations, and resource requirements, never exceed the global resource bound at any time.

MnZ Determines if a cumulative resource usage is within bounds.

E' `https://github.com/MiniZinc/libminizinc/blob/master/share/minizinc/std/fzn_cumulative.mzn`

Results

Problem	#	E'			MnZ		
		O0S0	O2S1	O3S2	O0	O1	O5
Quasigroup	43	41	42	41	40	39	40
Quasigroup Occ.	43	41	41	42	32	37	38
Wordpress	9	6	6	6	6	6	6
Wordpress Symm.	9	4	4	6	4	4	4
TTPPV	20	3	3	3	3	3	3
MSPSP	6	6	6	6	6	6	6
CVRPTW	5	0	0	0	0	0	0
Rostering	7	7	7	7	7	7	7


Chuffed MiniZinc vs Essence' Total Time
MiniZinc O0
MiniZinc O1
MiniZinc O5
Essence' O0_S0
Essence' O2_S1
Essence' O3_S2
instances solved
total time (s)
0
20
40
60
80
100
10^0
10^1
10^2
10^3

Take Away

MnZ allows better (expert) modeler control

MnZ provides a slightly more expressive language due to the facilities for code organization and reusability

SR provides a solid set of default settings

SR has a more consistent performance profile

University of
St Andrews

Thank you!

ruthhoffmann

www.st-andrews.ac.uk/computer-science/people/rh347/

rh347@st-andrews.ac.uk