Oxiflex - A Constraint Programming Solver for MiniZinc written in Rust

Gianluca Klimmer

University of Basel

15.07.2024

Constraint Programming Solver for MiniZinc written in Rust

Constraint Programming Solver for MiniZinc
written in Rust

Rust

Why Rust?

- Performance

Why Rust?

- Performance
  - $\rightarrow$ no abstractions, no garbage collector, no JIT

Possible Languages

Why Rust?

- Performance
  - $\rightarrow$ no abstractions, no garbage collector, no JIT
- Correctness
  - Prevent bugs

Why Rust?

- Performance
  - $\rightarrow$ no abstractions, no garbage collector, no JIT
- Correctness
  - Prevent bugs
- Ease of use
  - Library manager - Cargo

Why Rust?

- Performance
  - $\rightarrow$ no abstractions, no garbage collector, no JIT
- Correctness
  - Prevent bugs
- Ease of use
  - Library manager - Cargo
  - Functional - Haskell similarities

Why Rust?

- Performance
  - $\rightarrow$ no abstractions, no garbage collector, no JIT
- Correctness
  - Prevent bugs
- Ease of use
  - Library manager - Cargo
  - Functional - Haskell similarities
  - Enums

Why Rust?

- Performance
  - $\rightarrow$ no abstractions, no garbage collector, no JIT
- Correctness
  - Prevent bugs
- Ease of use
  - Library manager - Cargo
  - Functional - Haskell similarities
  - Enums
- Learn Rust

Constraint Programming Solver for MiniZinc
written in Rust

Constraint Programming Solver for MiniZinc
written in Rust

Constraint Network

Constraint Network

- Variables
  - Values to choose from
- Constraints
  - Rules for choosing values

Simple Example

Variables:

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

Constraints:

$$w = 2 \cdot x$$
$$w < z$$
$$y > z$$

Constraint Programming

var 1..4: w;
var 1..4: y;
var 1..3: x;
var 1..3: z;

constraint $w = 2 \cdot x$;
constraint $w < z$;
constraint $y > z$;

Constraint Programming

var 1..4: w;
var 1..4: y;
var 1..3: x;
var 1..3: z;

constraint $w = 2 \cdot x$;
constraint $w < z$;
constraint $y > z$;

solve satisfy;

Constraint Programming

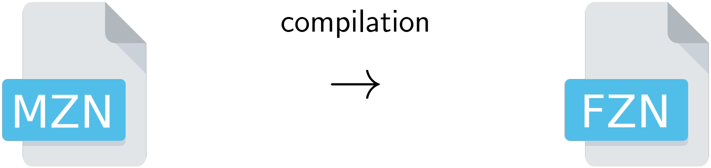
var 1..4: w;
var 1..4: y;
var 1..3: x;
var 1..3: z;

constraint w $= 2 \cdot$ x;
constraint w $<$ z;
constraint y $>$ z;

solve satisfy;

$\rightarrow$ MiniZinc!

MiniZinc


MZN
compilation
FZN

FlatZinc

```
array [1..2] of int: x_introduced_2_ = [1,-2];
array [1..2] of int: x_introduced_3_ = [1,-1];
array [1..2] of int: x_introduced_4_ = [-1,1];

var 2..4: w:: output_var;
var 1..4: y:: output_var;
var 1..3: x:: output_var;
var 1..3: z:: output_var;

constraint int_lin_eq(x_introduced_2_,[w,x],0);
constraint int_lin_le(x_introduced_3_,[w,z],-1);
constraint int_lin_le(x_introduced_4_,[y,z],-1);

solve satisfy;
```

FlatZinc

```
array [1..2] of int: x_introduced_2_ = [1,-2];
array [1..2] of int: x_introduced_3_ = [1,-1];
array [1..2] of int: x_introduced_4_ = [-1,1];

var 2..4: w:: output_var;
var 1..4: y:: output_var;
var 1..3: x:: output_var;
var 1..3: z:: output_var;

constraint int_lin_eq(x_introduced_2_,[w,x],0);
constraint int_lin_le(x_introduced_3_,[w,z],-1);
constraint int_lin_le(x_introduced_4_,[y,z],-1);

solve satisfy;
```

FlatZinc constraint example

```
predicate int_lin_eq(array [int] of int: as,
                     array [int] of var int: bs,
                     int: c)
```

FlatZinc constraint example

predicate int_lin_eq(array [int] of int: as,
array [int] of var int: bs,
int: c)

$$c = \sum_i \mathsf{as}[i] \cdot \mathsf{bs}[i]$$

FlatZinc constraint example

```
predicate int_lin_eq(array [int] of int: as,
                     array [int] of var int: bs,
                     int: c)
```

$$c = \sum_{i} \mathsf{as}[i] \cdot \mathsf{bs}[i]$$

$$w = 2 \cdot x$$

```
array [1..2] of int: x_introduced_2_ = [1,-2];
...
constraint int_lin_eq(x_introduced_2_,[w,x],0);
```

```
array [1..2] of int: x_introduced_2_ = [1,-2];
...
constraint int_lin_eq(x_introduced_2_,[w,x],0);
```

$$c = \sum_i \mathsf{as}[i] \cdot \mathsf{bs}[i]$$

array [1..2] of int: x_introduced_2_ = [1,-2];

...

constraint int_lin_eq(x_introduced_2_,[w,x],0);

$$c = \sum_i \text{as}[i] \cdot \text{bs}[i]$$

$$0 = \sum_i \text{x\_introduced\_2\_}[i] \cdot [\text{w,x}][i]$$

array [1..2] of int: x_introduced_2_ = [1,-2];

...

constraint int_lin_eq(x_introduced_2_,[w,x],0);

$$c = \sum_i \text{as}[i] \cdot \text{bs}[i]$$

$$0 = \sum_i \text{x\_introduced\_2\_}[i] \cdot \text{[w,x]}[i]$$

$$0 = \sum_i \text{[1,-2]}[i] \cdot \text{[w,x]}[i]$$

array [1..2] of int: x_introduced_2_ = [1,-2];

...

constraint int_lin_eq(x_introduced_2_,[w,x],0);

$$c = \sum_i \text{as}[i] \cdot \text{bs}[i]$$

$$0 = \sum_i \text{x\_introduced\_2\_}[i] \cdot \text{[w,x]}[i]$$

$$0 = \sum_i \text{[1,-2]}[i] \cdot \text{[w,x]}[i]$$

$$0 = 1 \cdot w - 2 \cdot x$$

array [1..2] of int: x_introduced_2_ = [1,-2];

...

constraint int_lin_eq(x_introduced_2_,[w,x],0);

$$c = \sum_i \text{as}[i] \cdot \text{bs}[i]$$

$$0 = \sum_i \text{x\_introduced\_2\_}[i] \cdot [\text{w,x}][i]$$

$$0 = \sum_i [\text{1,-2}][i] \cdot [\text{w,x}][i]$$

$$0 = 1 \cdot w - 2 \cdot x$$

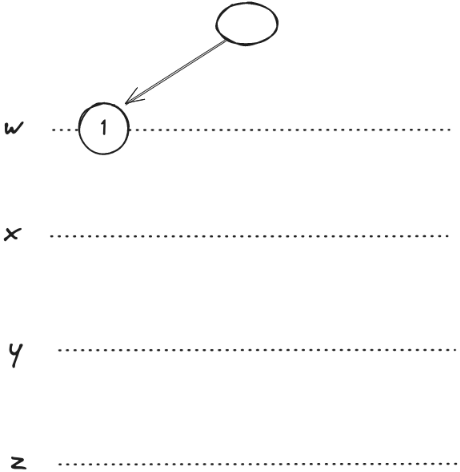
$$w = 2 \cdot x$$

Constraint Programming Solver for MiniZinc written in Rust

Constraint Programming Solver for MiniZinc written in Rust

Backtracking

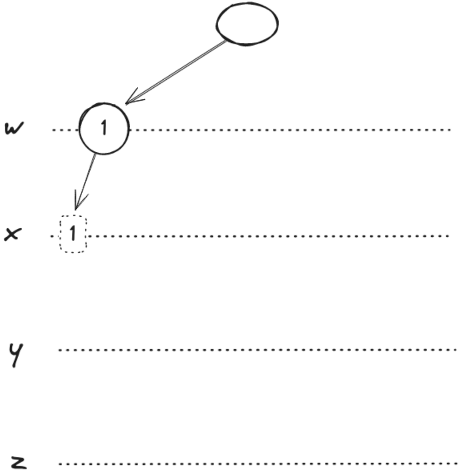
Backtracking Example



Constraints:

```
w = 2 * x
w < z
y > z
```

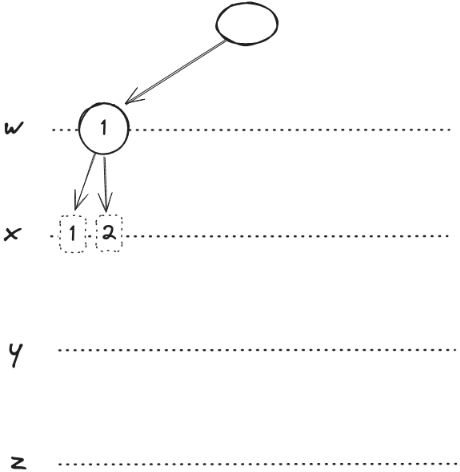
Backtracking Example



Constraints:

```
w = 2 * x
w < z
y > z
```

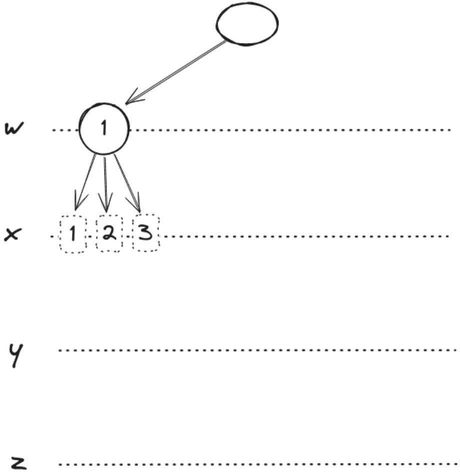
Backtracking Example



Constraints:

```
w = 2 * x
w < z
y > z
```

Backtracking Example



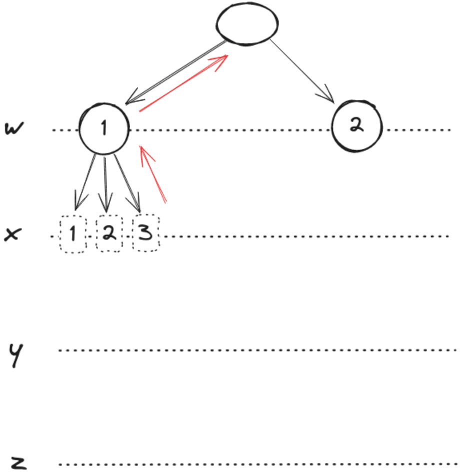
Constraints:

w = 2 * x

w < z

y > z

Backtracking Example



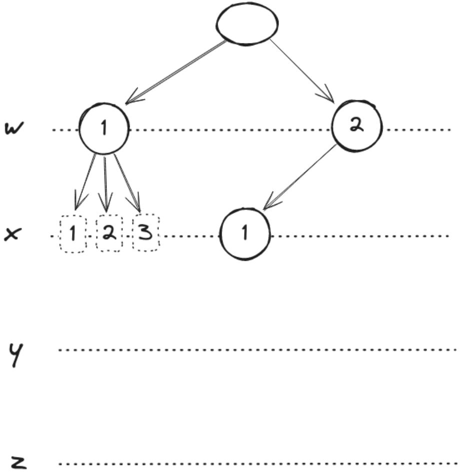
Constraints:

w = 2 * x

w < z

y > z

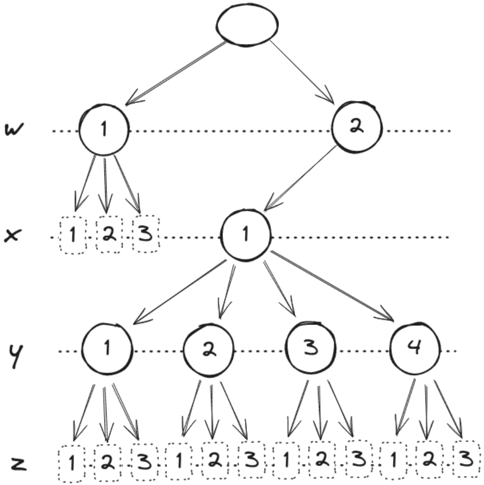
Backtracking Example



Constraints:

```
w = 2 * x
w < z
y > z
```

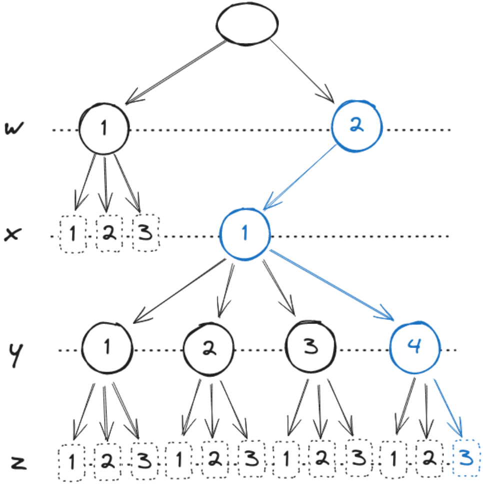
Backtracking Example



Constraints:

```
w = 2 * x
w < z
y > z
```

Backtracking Example



Constraints:

w = 2 * x

w < z

y > z

Solution:

w = 2

x = 1

y = 4

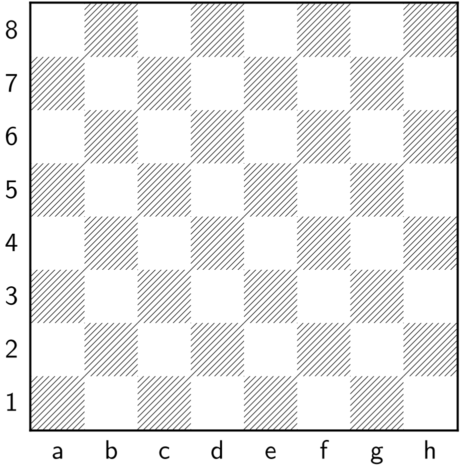
z = 3

Kinda like search...

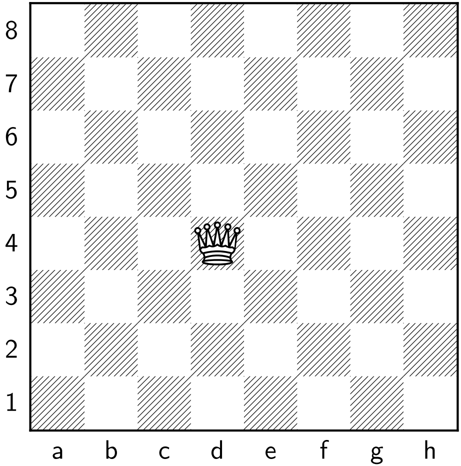
Can we do better?

8-Queens Problem

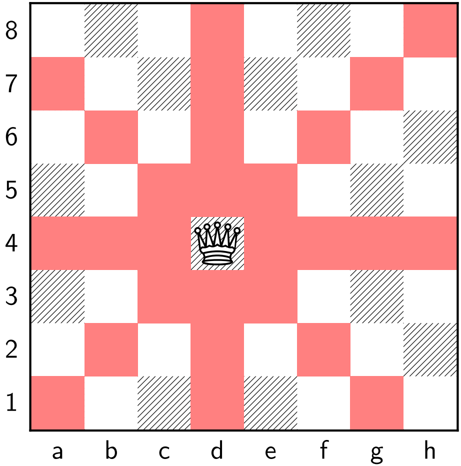
Chessboard



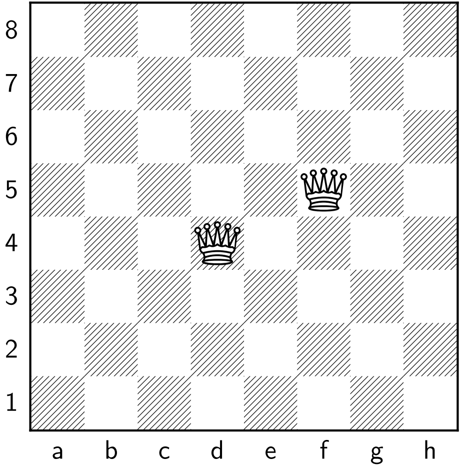
One Queen



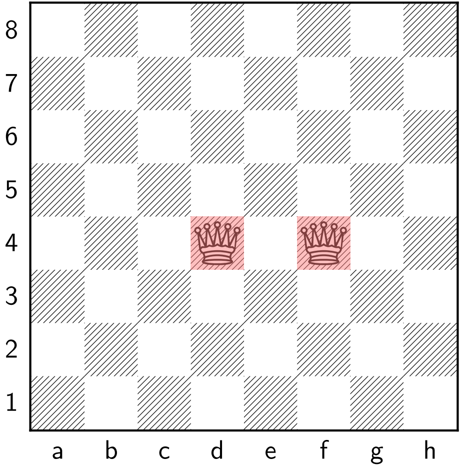
One Queen



Two Queens



Two Queens



N-Queens Problem

- Scalable

N-Queens Problem

- Scalable
  - 8-Queens $\rightarrow$ N-Queens

N-Queens Problem

- Scalable
  - 8-Queens $\rightarrow$ N-Queens
  - $n \times n$ chessboard

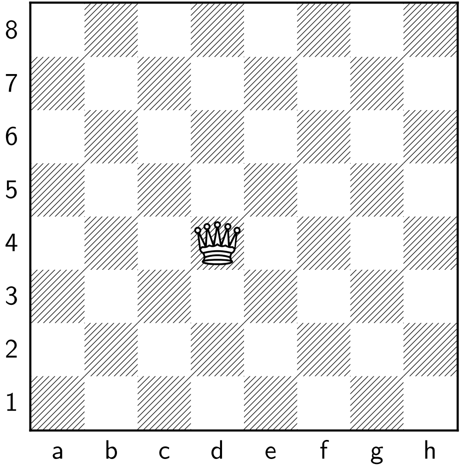
N-Queens Problem

- Scalable
  - 8-Queens $\rightarrow$ N-Queens
  - $n \times n$ chessboard
  - n Queens

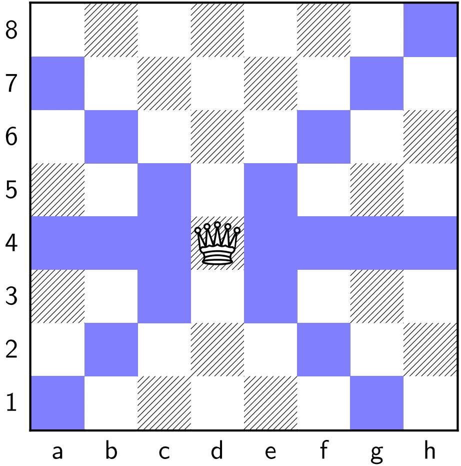
Inference

Forward Checking

Forward Checking Example

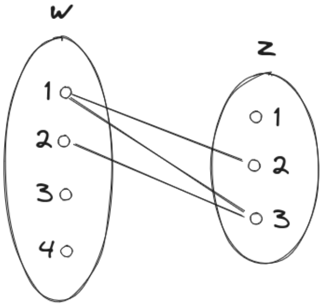


Forward Checking Example

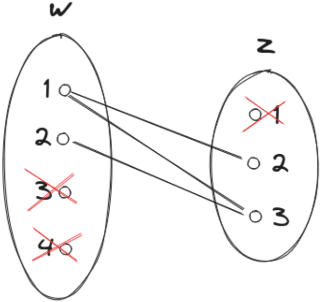


Arc Consistency

Arc Consistency Example



Arc Consistency Example



Improvements

- Inference

Improvements

- Inference
    - Forward Checking

Improvements

- Inference
  - Forward Checking
  - Arc consistency

Improvements

- Inference
  - Forward Checking
  - Arc consistency
    - AC-1

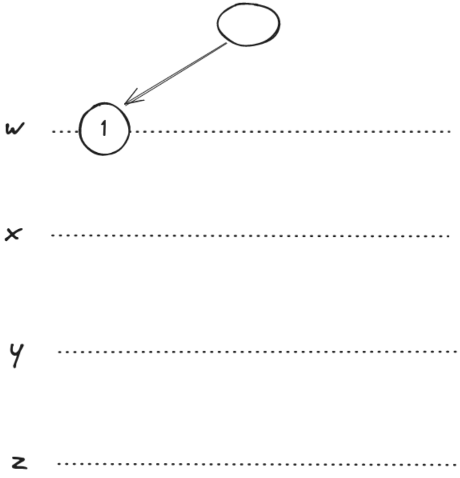
Improvements

- Inference
  - Forward Checking
  - Arc consistency
    - AC-1
    - AC-3

Improvements

- Inference
  - Forward Checking
  - Arc consistency
    - AC-1
    - AC-3
- Dynamic Variable Ordering

Dynamic Variable Ordering



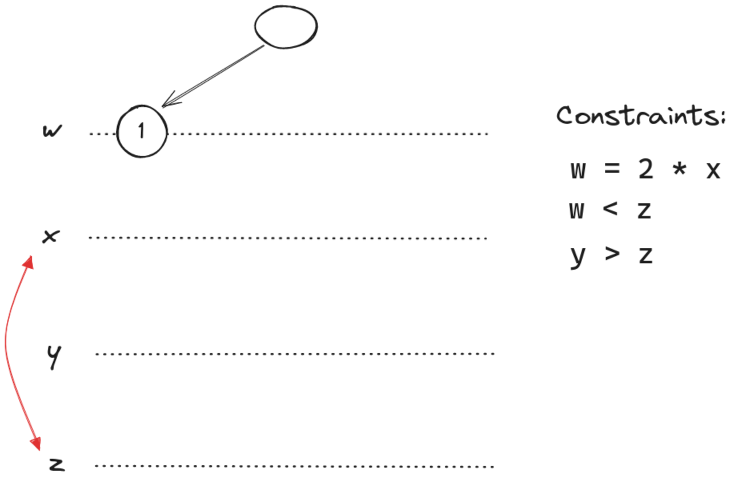
Constraints:

w = 2 * x

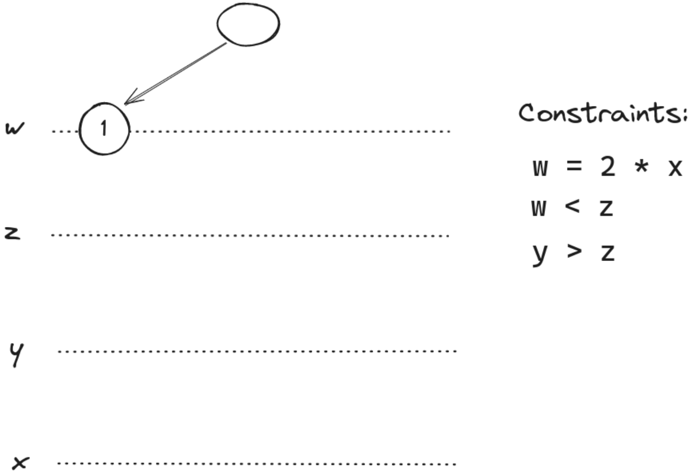
w < z

y > z

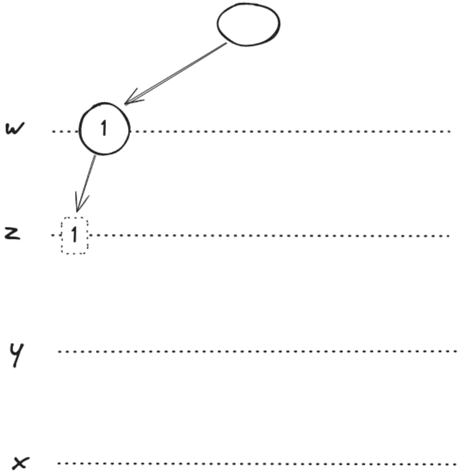
Dynamic Variable Ordering



Dynamic Variable Ordering



Dynamic Variable Ordering



Constraints:

w = 2 * x

w < z

y > z

Oxiflex

Demo

Limitations

- FlatZinc builtins

Limitations

- FlatZinc builtins
  - IntLinEq
  - IntLinLe
  - IntLinNe

Limitations

- FlatZinc builtins
  - IntLinEq
  - IntLinLe
  - IntLinNe
- No floating points

Limitations

- FlatZinc builtins
  - IntLinEq
  - IntLinLe
  - IntLinNe
- No floating points
- No minimize / maximize

Limitations

- FlatZinc builtins
  - IntLinEq
  - IntLinLe
  - IntLinNe
- No floating points
- No minimize / maximize
- Only one solution

Constraint Programming Solver for MiniZinc written in Rust

Benchmarks

N-Queens Problem

Queens Time

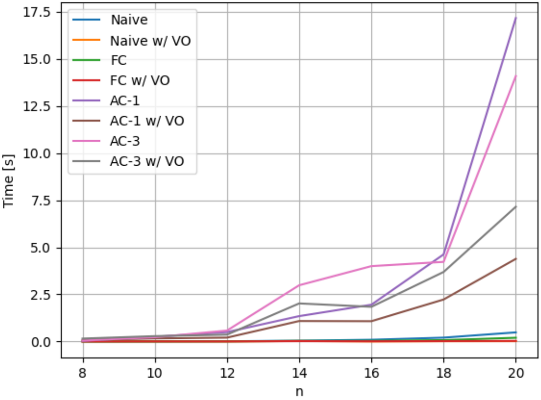


Figure: Averaged over > 10 runs

Queens Time

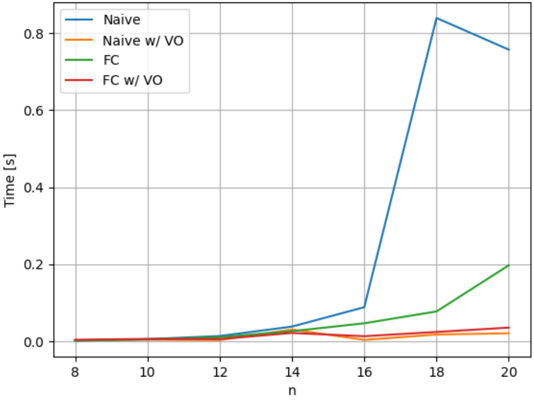


Figure: Averaged over > 10 runs

Queens Iterations

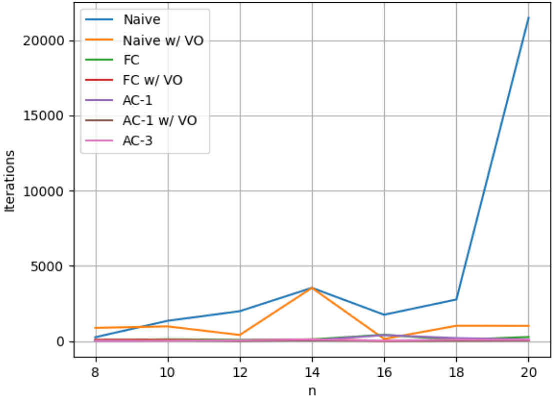


Figure: Averaged over 5 runs

Queens Iterations

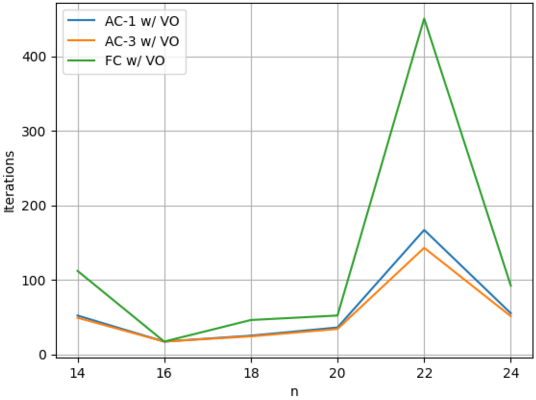


Figure: Averaged over 5 runs

Slow Convergence

Slow Convergence for small n

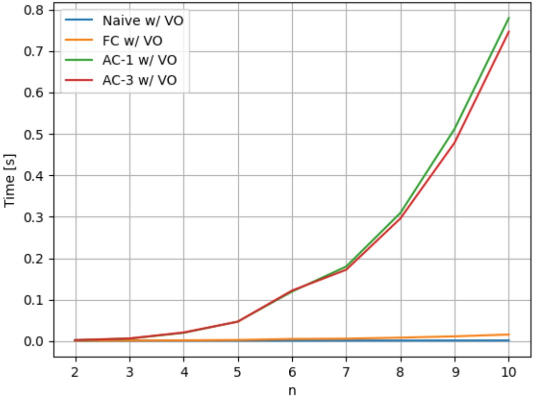


Figure: Averaged over > 10 runs

Slow Convergence for small n

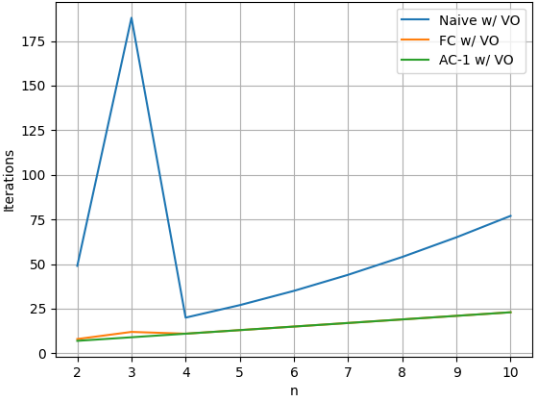


Figure: Averaged over 5 runs

Slow Convergence Comparison

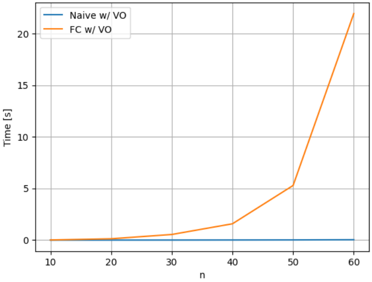


Figure: Averaged over > 10 runs

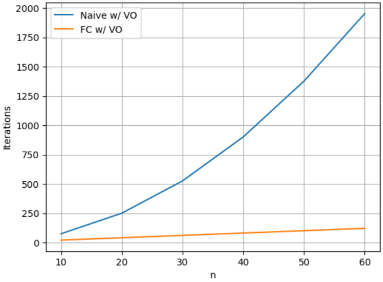


Figure: Averaged over 5 runs

Oxiflex vs Gecode

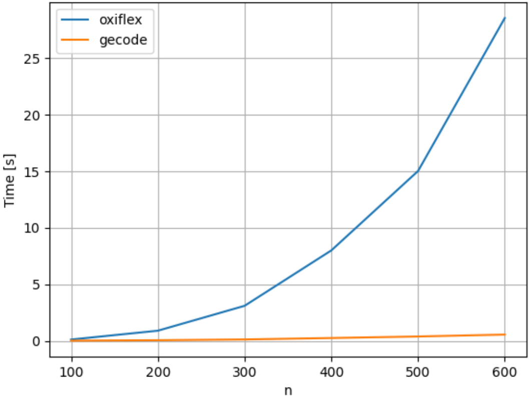


Figure: Averaged over > 10 runs

Conclusion

The end