

Discrete Optimization Specialization: Workshop 0

First Steps

1 Introduction

This workshop is simply about getting used to MiniZinc, and trying out a few trivial models.

1.1 Hello World - `hello.mzn`

Build a MiniZinc model `hello.mzn` that outputs the result “Hello World”!

Test it using

```
minizinc hello.mzn
```

1.2 Input and Output - `io.mzn`

Build a MiniZinc model `io.mzn` that takes a data file defining integer parameter n and outputs its value.

Test it using

```
minizinc io.mzn -D"n = d;"
```

for different values d , which it should print out.

1.3 Simple Decision - `x110.mzn`

Build a MiniZinc model `x110.mzn` with a single decision variable x taking values from one to ten, which outputs its value.

Test it using

```
minizinc x110.mzn -a
```

which should print out all the solutions for x .

1.4 Simple Optimization - `xopt.mzn`

Build a MiniZinc model `xopt.mzn` with a decision variable x taking values from 0 to 10, with constraints to ensure that x is divisible by 4, which outputs the value of x that gives the minimum value of $(x - 7)^2$.

Test it using

```
minizinc xopt.mzn
```

which should print out the optimal solution. Also test it using

```
minizinc xopt.mzn -a
```

which should print out all the solutions it found on the way to the optimal. Notice how the `-a` flag means different things for satisfaction problems (all solutions to the problem) and optimization problems (all solutions found on the way to the optimal).

Question Suppose you cannot use the `mod` function, how would you alternatively model that x is divisible by 4?

1.5 Arrays - `array.mzn`

Define a MiniZinc model `array.mzn` which takes an integer parameter n defining the length of an array of numbers x taking values from 0 to 9. Constrain the array so the sum of the numbers in the array is equal to the product of the numbers in the array. Output the resulting array.

Test your model using

```
minizinc array.mzn -a
```

which should print out all the solutions for x .

Add a constraint to ensure that the numbers in the array are non-decreasing, i.e. $x[1] \leq x[2] \leq \dots \leq x[n]$. This should reduce the number of similar solutions. This is an example of *symmetry breaking* which will be very useful.

How big a number can you solve with your model? Why do you think this happens?

1.6 A Sequence - `seq.mzn`

Define a MiniZinc model `seq.mzn` which takes a integer parameter n defining the length of an array of numbers x taking values from 0 to 3. Constrain the numbers so that the first number is zero, the last number is 3, and the sum of any two adjacent numbers in the array x is at most 3. Also constrain the value of x at positions divisible by 3 to be greater than or equal to 2. Maximize the sum of the numbers in the array x . The output should be of the form *sum = array of values*, e.g.

```
6 = [0,1,2,0,3]
```

given a solution $x = [0, 1, 2, 0, 3]$ for $n = 5$.

Test your model for various values d of n using

```
minizinc seq.mzn -D"n = d;"
```

Test out at least the values of d from 3 to 9.