NOPT042 Constraint programming: Tutorial

Jakub Bulín

KTIML MFF UK

Fall 2021

About this tutorial

Useful links

- Our tutorial's webpage
- Our tutorial on Moodle
- Webpage of the lecture
- Course info in the SIS

Credit requirements

Choose an appropriately complex combinatorial problem (to be approved by me, see here for a few examples). Design an efficient MiniZinc model for your problem. The documentation should include a description of the problem and your model, example data including some hard instances (both positive and negative), discussion of your chosen approach and its comparison to other modeling strategies, results of numerical experiments. Give a brief presentation of your work.

Resources

- MiniZinc homepage
- The MiniZinc Handbook (Chapter 2 contains a tutorial)
- Coursera courses Basic and Advanced Modelling for Discrete Optimization by the creator of MiniZinc
- The MiniZinc Examples Archive
- Hakan Kjellerstrand's Library of MiniZinc Models
- Past tutorials based on SICStus Prolog

Constraint programming

Discrete (‘combinatorial’, as opposed to ‘continuous’) optimization, constraint satisfaction

- a form of decision making,
- many everyday problems:
- solve Sudoku
- schedule classes
- schedule trains
- coordinate multi-facility production
- logistics of product transportation
- ...

Assign values to variables subject to *constraints*, satisfy/optimize.

You will learn to...

- solve complex problems "without even knowing how"
- state the problem in a high-level constraint modeling language: MiniZinc
- use a constraint solver to "automagically"[1] solve it
- techniques and tricks to build efficient constraint models
- best practices, testing and debugging
- define your own constraints, search heuristics
- integrate constraint modeling within the Python toolchain (bind native Python objects with MiniZinc data structures, manipulate the solution stream, concurrent solving, ...)

[1] Magic explained in the lectures.

Why constraint programming?

- the ‘holy grail’ of programming: tell the computer what you want, not how to do it
- an order of magnitude easier than programming algorithms
- huge engineering investment in constraint solvers, highly optimized, often faster than your own algorithm would be (especially in “mixed” NP-complete problems), heuristic approach
- easier for molecular biologists to learn to *specify* their problems in a formal language, than for programmers to *learn* molecular biology

History and (folk) etymology

- prográphō ("I set forth as a public notice"), from pró ("towards") + gráphō ("I write")
- program of a political movement
- program of a concert, broadcast programming, tv program
- computer program (1940s)

Independently:

- U.S. Army operational programs (1940s)
- "linear programming" (1946) Maximize $\mathbf{c}^T\mathbf{x}$ (objective function) subject to $A\mathbf{x} \leq \mathbf{b}$, $\mathbf{x} \geq 0$ (constraints).
- integer programming (1964), logic programming (late 1960s), constraint logic programming (1987), constraint programming (early 1990s)

Modeling (USA) vs. Modelling (everywhere else)[2]

[2] MiniZinc comes from Australia, Gecode from Australia and Scandinavia

Why MiniZinc?

MiniZinc: Towards A Standard CP Modelling Language (2007)

- "solvers use different, incompatible modelling languages that express problems at varying levels of abstraction"
- ECLiPSe, SICStus (Prolog); Gecode, ILOG (C/C++); Choco (Java), Minion, OPL ...
- MiniZinc: a standard Constraint modelling language, easy implementation (FlatZinc), benchmarking (The MiniZinc Challenge)
- high-level, modern[3]

---

[3] Python 1990, C# 2001, Scala 2003, Clojure 2007, Go 2009

Hello, World!

Example (hello-world)

Install the MiniZinc bundle. Create and run "hello-world.mzn" with the following code:

```
output ["Hello, world!"];
```

Try both the IDE and command-line interface:

```
minizinc hello-world.mzn
```

Output statement

```
output <list-of-strings>;
```

- optional, at most one
- concatenation: ++
- \n, \t, "\(x)", show(x), show_int(k,x), show_float(k,d,x)

Pythagorean triples

Example (pythagorean-triples)

Generate all Pythagorean triples, i.e. natural numbers such that $a^2 + b^2 = c^2$, up to a fixed parameter.

Symmetry breaking[4]:

```
constraint a < b;
```

Generate all solutions:

```
minizinc -a pythagorean-triples.mzn
```

Parameters vs. Decision Variables

Solve statement

```
solve satisfy;
solve maximize ⟨arithmetic expression⟩;
solve minimize ⟨arithmetic expression⟩;
```

[4] See https://en.wikipedia.org/wiki/Symmetry-breaking_constraints

Chinese remainder theorem

Example (chinese-remainder)

After an indecisive battle, general Han Xin wanted to know how many soldiers of his 42000-strong army remained. In order to prevent enemy spies hidden among his soldiers to learn the number, he decided to use modular algebra: He ordered his soldiers to form rows of 5 and 3 soldiers remained. Then rows of 7; 2 remained. Then rows of 9; 4 remained. Then rows of 11; 10 remained. Finally, rows of 13; 1 remained.[5]

- What are the *parameters* of our problem?
- Identify the *decision variables* (type, domains[6])
- What are the constraints? (implicit?)
- Is it satisfaction or optimization?

[5] Was this necessary?

[6] as small as possible

Australia

Map coloring

Example (color-map)

Create a model to color the map of Australian states and territories [7] with 4 colors (cf. The 4-color Theorem).

Q: What is wrong with this example?[8]

Exercise for later

1. Create a model to decide if a given graph is 4-colorable.
2. Find the chromatic number of a given graph.

Data representation?

---

[7] excluding the Australian Capital Territory, the Jervis Bay Territory, and the external territories

[8] Separate model from data! (model vs. instance)

Overview of the IDE and CLI

Useful command-line options:

- `-h, --help, --help ⟨solver-id⟩`
- `--solvers`
- `-a, --all-solutions`
- `-v, --verbose`
- `-s, --statistics`
- `-c, --compile`
- `-o, --output-to-file`
- `-d ⟨filename⟩, --data ⟨filename⟩`
- `-D ⟨data⟩, --cmdline-data ⟨data⟩`

Solvers and tools

Gecode a good default choice, open-source, fast, supports MiniZinc natively, and more: `include "gecode.mzn";`

Chuffed lazy clause generation, combines finite domain propagation with ideas from SAT: explain and record failure and perform conflict directed backjumping; can find a solution fast

COIN-BC Computational Infrastructure for Operations Research: Branch and Cut; mixed-integer programming (MIP) solver

findMUS finds “minimal unsatisfiable sets” of constraints in the model

Gist visualize the search tree, guide the search manually

See The MiniZinc Handbook::Solvers for more information.

More about the language

- Identifiers: alphabetic chars, digits, _[9]
- Relational operators:
  `= or ==, !=, <, >, <=, >=` [10]
- Logical operators:
  `/\, \/, ->, <-, <->, xor, not`
- Integer arithmetic:
  `+,-,*,div,mod,abs(x),pow(x,y)` [11]
- Floating point: `+,-,*,/,int2float,abs,sqrt,ln, log2,log10,exp,pow,sin,cos,tan,...`
  literals: `1.23, 4.5e-6, 7.8+E9`
- primitive types: `int, float, bool, string`
  string for output&readability, range: 1..5: n; 0.0..1.0: p;

[9] starts with an alphabetic character (not a reserved word)

[10] Both = and == mean 'equals' (assignment not needed). Made by computer scientists for computer scientists.

[11] +,- are also unary

Basic structure of a model (order does not matter)

- Parameter declaration (optional keyword `par`)
  ```
  par int: n [= 1];
  int: n [=1];
  ```
- Decision variable declaration (keyword `var` is not optional!)
  ```
  var int: x [= n];
  ```
- Assignment item
  ```
  n = 5;
  ```
  `x = 3;` (equivalent to `constraint x = 3;`)
- Constraint item
  ```
  constraint ⟨Boolean expression⟩;
  ```
- Solve statement
- Output statement
- Include item
  ```
  include ⟨filename⟩;
  ```
- Predicate item, test item, solve annotation (later)

Crypt-arithmetic

Example (send-more-money)

Solve the crypt-arithmetic puzzle (each letter represents a different base-10 digit):

$$SEND + MORE = MONEY$$

Example (donald-gerard-robert)

Write a better constraint model[12] based on carry bits for the puzzle DONALD + GERARD = ROBERT.

[12] "Some letters can be computed from other letters and invalidity of the constraint can be checked before all letters are known" (from R. Barták's tutorial in Prolog, see the code), "If we don't study the mistakes of the future, we're bound to repeat them for the first time." (Ken M)

Crypt-arithmetic cont'd

Compare w/ `send-more-money.cpp` (for pre-MiniZinc Gecode).

Exercise for later

Design a general model for crypt-arithmetic puzzles of the form word1 + word2 = word3.

Try your model on the following hard instance: [13]

```
  B A I J J A J I I A H F C F E B B J E A
+ D H F G A B C D I D B I F F A G F E J E
-----------------------------------------
= G J E G A C D D H F A F J B F I H E E F
```

[13] From Hakan Kjellerstrand's library (orig. source: Prolog benchmark problem GNU Prolog, P. Van Hentenryck, adapted by D. Diaz)

Global constraints (more on this later!)

Constraints that represent high-level modelling abstractions, for which many solvers (e.g. Gecode) implement special, efficient inference algorithms.

- Include all global constraints: `include "globals.mzn";`
- or include individual constraints, e.g.: `include "alldifferent.mzn;"`

Example (pigeonhole)

Can n pigeons fit in m holes so that every pigeon has its own hole?

- `predicate all_different(array [$X] of var int: x)`
- Compare models with and without the global constraint (compiled code, performance). Look at documentation and implementation.
- Idea: dual model (switch variables&values), chanelling constraint

Optimization

Example (baking-cakes)

How many chocolate&banana cakes should we bake to get rich?

- banana cake: 250g flour, 2 bananas, 75g sugar, 100g butter (sells for $4.50)
- chocolate cake: 200g flour, 75g cocoa, 150g sugar, 150g butter (sells for $4)

Supplies: 4kg flour, 6 bananas, 500g cocoa, 2kg sugar, 500g butter

- Separate data from model (.dzn file or cmd-line argument):
  `minizinc baking-cakes2.mzn pantry.dzn`
- Verify consistency of data:
  `constraint assert(⟨boolean-expr⟩,⟨error-msg⟩);`
- Better: a general production planning model (later)

Structured data types

Example (laplace)

Rectangular metal plate with different temperatures on each side: In a stable state, temperature at each internal point is the average of its neighbours (finite element method).

- declare a 2-dimensional *array* of decision variables:
  ```
  array[0..w, 0..h] of var float:  t;
  ```
- w and h are parameters (index sets have to be fixed)
  ```
  int w; int h;
  ```
- named range (a *set*):
  ```
  set of int:  ROW = 0..h;
  ```
- aggregation function example: `forall` (Not a for-loop!)
  ```
  constraint forall(i in ROW)(t[i,0] = left);
  ```
  other aggregation functions: exists, sum, max, min, xorall, ...

Production Planning

Example (production-planning)

We can produce different types of products. Each type of product consumes certain amount of each resource and produces a certain amount of profit. Goal: maximize profit subject to resource capacity constraints.

Make your model fit the data declared here:
`baking-cakes-data.dzn`

More on MiniZinc IDE and CLI

I will try to use the mouse, but...

Minizinc IDE keyboard shortcuts

CTRL+R	MiniZinc→Run	Compile and execute
CTRL+E	MiniZinc→Stop	Abort execution (very useful)
CTRL+B	Minizinc→Compile	Compile only (to FlatZinc)

How to add MiniZinc to PATH on Windows (change the path to match your installation):

```
setx PATH "%PATH%;C:\Program Files\MiniZinc\"
```

More on installation here:

https://www.minizinc.org/doc-2.2.3/en/installation.html

The Knapsack Problem

Example (knapsack)

Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

A thief breaks into a home. They can carry 23kg. What should they take to maximize profit? There are the following items:

- a TV (weighs 15kg, costs $500),
- a desktop computer (weighs 11kg, costs $350)
- a laptop (weighs 5kg, costs $230),
- a tablet (weighs 1kg, costs $115),
- an antique vase (weighs 7kg, costs $180),
- a bottle of whisky (weighs 3kg, costs $75), and
- a leather jacket (weighs 4kg, costs $125).

More examples using arrays

Example (n-queens)

Place n queens on an $n \times n$ chess board so that no queen is attacked by another queen.

Example (sudoku)

Design a MiniZinc model which solves the Sudoku puzzle.

Example (magic-square)

A magic square is an $n \times n$ table filed with numbers from 1 to n^2 such that the sums of every row, column, and both main diagonals are the same. Can a given partially pre-filled table be filled to get a magic square?

Examples (minesweeper)

Find all mines starting from a given Minesweeper game configuration.

Arrays

- Declare an array:
  ```
  array [1..8,1..8] of bool:  a;
  array [<index-set-1>,...,<index-set-n>] of <type>
  ```
- one-dim array literal: `b = [1,2,3,4,5,6];`
- two-dim array literal: `c = [|1,2|3,4|5,6|];`
  `c = array2d(1..3, 1..2, [1, 2, 3, 4, 5, 6]);` [14]
- indexing: `b[1]; c[3,2];`
- concatenation: `b ++ [7,8];`
- length of a one-dim array: `length(b);`
- array literals indexed starting from 1:
  ```
  array[0..2] of int:  a = [4,5,6];
  correct:  a = array1d(0..2,[4,5,6])
  ```
- index sets cannot be decision variables :-(

---

[14] up to array6d()

Sets

- Declare a set variable:

```
set of <type>: <identifier>;
```

- type can be int, bool, float, or enum[15]
- set literals: `{1,3,5}` , `{JUN,SEP,DEC}`
- or a range (over int, float or enum):

```
2..5 , 1.5..2.5 , JAN..MAY
```

- set operations:
  - `in` (membership),
  - `superset, subset` (non-strict),
  - `union, intersect` (union and intersection),
  - `diff, symdiff` (difference and symmetric difference),
  - `card` (cardinality)
  - why not complement?
- only (bounded) sets of int or enum can be decision variables

```
var set of 1..10: a;
```

[15] basically a range 1..n with fancy names, more later

List and set comprehension, generators

List comprehension:

```
[<expr> | <generator-expr>,...,<generator-expr>]
```

- `<generator-exp>` is of the form:
  `<generator>` or `<generator> where <bool-expr>`
- `<generator>` is of the form:
  `<identifier>,...<identifier> in <array-exp>`
- idenfitifiers are iterators over the array[16]
- generators and filter usually *fixed* (free of decision variables)[17]

Set comprehension:

```
{<expr> | <generator-expr>,...,<generator-expr>}
```

- generators, boolean expr., generated elements must be fixed

---

[16](the last one iterates most rapidly)

[17]if not, we get `array[] of var opt <type>` (more on option types later)

Aggregation functions and generator call expressions

Aggregation function: any function whose input is a single array

Built-in aggregation functions:

- arithmetic: `sum, product, min, max`
- boolean: `forall, exists, xorall, iffall`

Generator call expression:

```
<agg-func> ( <generator-exp> ) ( <exp> )
```

this is equivalent to:

```
<agg-func> ( [ <exp> | <generator-exp> ] )
```

For example:

```
forall( [a[i] != a[j] | i,j in 1..3 where i < j])
forall(i,j in 1..3 where i < j)(a[i] != a[j])
```

Enumerated types

- Declare an enum:
  `enum: days;`
- Enum literal:
  `days = {Mon, Tue, Wed, Thu, Fri, Sat, Sun};`
- a "set of constants"
- used for type safety
- converted to an integer range starting from 1, i.e.
  `Mon + Tue = 3;`
- `enum_next(days, Fri); enum_prev(days, Sun);`
- `to_enum(days, 4) = Thu;`
- `card(days)=7; min(days)=Mon; max(days)=Sun;`
- `enum my_enum = enum_anon(n);`

Even more about MiniZinc

- conditional statement:
  `if <bool-exp> then <exp-1> else <exp-2> endif`
- type coercion: bool $\to$ int, int $\to$ float, and bool $\to$ float (two-step), coercion is automatic, for both parameters and decision variables (functions `bool2int, int2float`)
- the built-in function `fix` : coerce a decision variable to a parameter, in output statements
- have a look at the MiniZinc Cheatsheet.

How to implement a CP solver

- Gecode: (one of) the best constraint solvers
- A recent talk about Gecode:
  - https://chschulte.github.io/talks/Gecode%202018.pdf
- Source code (300k lines):
  - https://github.com/Gecode/gecode
- Maybe have a look at a more lightweight solver, like Java-based MiniCP:
  - https://github.com/Damoy/MiniCP
- ...or code your own CP solver using ideas from the lecture!

A simple planning problem

Example (swimmers)

[18] In medley swimming relay, a team of four swimmers must swim 4x100m, each swimmer using a different style: breaststroke, backstroke, butterfly, or freestyle. The table below gives their average times for 100m in each style. Which swimmer should swim which stroke to minimize total time?

Swimmer	Free	Breast	Fly	Back
A	54	54	51	53
B	51	57	52	52
C	50	53	54	56
D	56	54	55	53

Things to try: Write a general model, generate larger instances, and try to make your model as efficient as possible.

[18] From: W. Winston, Operations Research: Applications & Algorithms

Modelling functions

The swimmers problem is an example of the Assignment Problem:
https://en.wikipedia.org/wiki/Assignment_problem

In general, how to model a function (mapping) $f : A \to B$?

- as an array:
  ```
  array[A] of var B: f;
  ```
- injective:
  ```
  constraint alldifferent(f);
  ```
- surjective: a partition of A into classes labelled by B
  ```
  array[B] of var set of A: classes;
  constraint partition_set(classes, A);
  ```
- bijective:
  ```
  array[B] of var A: invf;
  constraint inverse(f, invf);
  ```

More on modelling functions

- partial function: a dummy value for undefined inputs
- dual model: switch the role of variables and values (not a function unless f injective, see above)
- channelling: combine the primal and dual models
  `constraint int_set_channel(f, classes);`
- assignment is a very common part of practical modeling
- (technically, every CSP is an assignment problem)
- there are global constraints for everything
  https://www.minizinc.org/doc-2.5.0/en/lib-globals.html

A simple scheduling problem

Example (moving[19])

Four friends are moving. The table shows how much time and how many people are necessary to move each item. Schedule the moving to minimize total time. (When to start moving each item?)

Item	Time (min)	People
piano	45	4
chair	10	1
bed	25	3
table	15	2
couch	30	3
cat	15	1

Things to try: Write a general model, generate larger instances, and try to make your model as efficient as possible.

[19] Adapted from R. Barták's practical; check the SICStus Prolog model.

The Cummulative global constraint

Requires that a set of tasks given by start times s, durations d, and resource requirements r, never require more than a global resource bound b at any one time.

```
predicate cumulative(
        array [int] of var int:  s,
        array [int] of var int:  d,
        array [int] of var int:  r,
        var int:  b)
```

Assuming that for all i, $d[i] \geq 0$ and $r[i] \geq 0$.

Example (moving-trolleys)

Schedule moving furniture so that each piece of furniture has enough people and enough trolleys available during the move.

Modelling the Boolean Satisfiability Problem (SAT)

Example (sat)

Is a given CNF formula satisfiable?

- a literal is either a variable or the negation of a variable,
- a clause is a disjunction of literals,
- a formula is in conjunctive normal form (CNF) if it is a conjunction of clauses.

DIY SAT solver

Do-it-yourself SAT solver which understands the DIMACS CNF format, using Python and MiniZinc (the minizinc python package).

Python minizinc package

- Install:
  `pip install minizinc`
- Homepage:
  https://pypi.org/project/minizinc/
- Documentation:
  https://minizinc-python.readthedocs.io/en/latest/
- Useful for processing input and output, executing many instances, statistics[20], constraint solving as a blackbox inside a more complex system, ...
- See also: IPython/Jupyter notebook iminizinc magic
  https://github.com/MiniZinc/iminizinc
- But in this practical, we mostly care about modelling one (more complex) problem well.

[20] Phase transitions in discrete optimization; scheduling: predict running time

Moving with trolleys

Example (moving-trolleys)

Schedule moving furniture so that each piece of furniture has enough people and enough trolleys available during the move.

A possible instance:

```
OBJECTS = {piano, fridge, doublebed, singlebed,
wardrobe, chair1, chair2, table};
available_handlers = 4;
available_trolleys = 3;
duration = [60, 45, 30, 30, 20, 15, 15, 15];
handlers = [3, 2, 2, 1, 2, 1, 1, 2];
trolleys = [2, 1, 2, 2, 2, 0, 0, 1];
```

Stable relationships

Example (stable-marriage)

Given n men and n women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. (When there are no such pairs, the matching is stable.)

Try the instance A:YXZ, B:ZYX, C:XZY, X:BAC, Y:CBA, Z:ACB

Example (stable-roommate)

Given $2n$ people, where each person has ranked all others in order of preference, pair the people together such that there are no two people who would both rather have each other than their current partners.

Try the instance A:BCD, B:CAD, C:ABD, D:ABC

How to model general constraints

The table constraint enforces that a tuple of variables takes a value from an array of tuples. Since there are no tuples in MiniZinc this is encoded using arrays:

```
table(array[int] of var int:x, array[int,int] of int:t)
```

(Alternatively, there is a version with boolean arrays.)

Example (Graph homomorphism)

Given a pair of graphs G, H, find all homomorphisms from G to H.

Graph homomorphism is a function $f : V(G) \to V(H)$ such that

$$\{u, v\} \in E(G) \implies \{f(u), f(v)\} \in E(H)$$

- Generalizes graph k-coloring ($c : G \to K_k$)
- Easier version: oriented graphs
- How would you model the Graph Isomorphism Problem?

Defining predicates

Predicates are defined by a statement of the form

```
predicate <name> (<arg-def>,...,<arg-def>) = <bool-exp>
```

where each `<arg-def>` is a type declaration, except index sets don't have to be fixed, i.e. we can write: `array[int] of var int`

```
predicate no_overlap(var int:s1, int:d1, var int:s2,
int:d2) = s1 + d1 <= s2 \/ s2 + d2 <= s1;
```

We can declare a predicate but define it in a seperate file(s):

```
predicate edge(var VERTEX: u, var VERTEX: v);
% include "edge-from-list.mzn";
% include "edge-from-incidence-matrix.mzn";
```

Test constraints, assert

Using the keyword `test` we can define new constraints that only involve parameters and unlike predicates can be used inside the test of a conditional expression:

```
test even(int:x) = x mod 2 = 0;
```

A new form of the assert command for use in predicates:

```
assert (<bool-exp>, <string-exp>, <exp>)
```

If the first argument is true, it returns the third argument, otherwise it prints the second argument.

```
predicate lookup(array[int] of var int:x,int:i,var int:y)
= assert(i in index_set(x),"index out of range",y = x[i]);
```

Defining functions

Similar to predicates, but return type doesn't have to be bool:

```
function <type>:<name>(<arg-def>,...,<arg-def>)=<exp>
```

Example: the Manhattan metric

```
function var int: manhattan(var int:x1, var int:y1, var
int:x2, var int:y2) = abs(x2 - x1) + abs(y2 - y1);
```

Local variables and constraints:

```
let {...} in <exp>
```

Example: square root `function var int: mysqrt(var int:x) =
let { var 0..infinity: y; constraint x = y * y; } in y;`

Reflection functions

Array index sets:

- `index_set(<1-D array>)`
- `index_set_1of2(<2-D array>)`
- `index_set_2of2(<2-D array>)`

Domain reflection:

- `dom(<exp>)` returns a safe approximation to the possible values of the expression
- `lb(<exp>)` a safe lower bound
- `ub(<exp>)` a safe upper bound

and versions for arrays:

```
dom_array(<array-exp>)
lb_array(<array-exp>)
ub_array(<array-exp>)
```

Balanced diet

Example (balanced-diet)

Plan a meal consisting of a main dish, a side dish, and a dessert. Each item has a kilojoule count, protein in grams, salt in milligrams, and fat in grams, as well as cost in cents.

The goal is to find the cheapest possible meal which has:

- at least the given minimum kilojoule count,
- at least the given minimum amount of protein,
- at most the given maximum amount of salt, and
- at most the given maximum amount of fat.

See balanced-diet.dzn for a sample instance.

Rostering

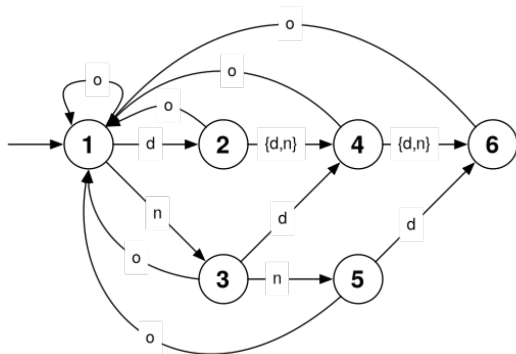
Example (nurse-roster)

Schedule the shifts of `num_nurses` nurses over `num_days` days.

Each nurse is scheduled for each day as either: (d) on day shift, (n) on night shift, or (o) off. In each four day period a nurse must have at least one day off, and no nurse can be scheduled for 3 night shifts in a row.

We require `req_day` nurses on day shift each day, and `req_night` nurses on night shift, and that each nurse takes at least `min_night` night shifts.

Nurse rostering condition as a DFA



	d	n	o
1	2	3	1
2	4	4	1
3	4	5	1
4	6	6	1
5	6	0	1
6	0	0	1

The regular constraint

Is a sequence of symbols accepted by a DFA?

```
regular(array[int] of var int: x, int: Q, int: S,
array[int,int] of int: d, int: q0, set of int: F)
```

Constrains that the sequence of values in array x (which must all be in $\{1, \ldots, S\}$ is accepted by the DFA of Q states with input alphabet $\{1, \ldots, S\}$ and transition function

$$d : \{1, \ldots, Q\} \times \{1, \ldots, S\} \rightarrow \{0, \ldots, Q\}$$

and initial state $q0 \in \{1, \ldots, Q\}$ and accepting states F. State 0 is reserved to be a fail state.

See also `regular_nfa` .

The seesaw problem[21]

Example (seesaw)

Adam (36 kg), Boris (32 kg) and Cecil (16 kg) want to sit on a 10-foot long seesaw such that they are at least 2 feet apart and the seesaw is balanced.

Write a general model for any number of people.
Possible decision variables?

1. Position on the seesaw for each person.
2. Distances between persons, position of the first person, and order of persons.
3. Person or empty for each position on the seesaw.

Multiple modeling?

How to improve performance of our model?

---

[21] From R. Barták's practical

Symmetry breaking[22]

Add constraints to choose only one of symmetric variants of a (partial) assignment; many useful global constraints

- **Bin packing:** when trying to pack items into bins, any two bins that have the same capacity are symmetric.
- **Graph colouring:** When trying to assign colours to nodes in a graph such that adjacent nodes must have different colours, we typically model colours as integer numbers. However, any permutation of colours is again a valid graph colouring.
- **Vehicle routing:** if the task is to assign customers to certain vehicles, any two vehicles with the same capacity may be symmetric (this is similar to the bin packing example).
- **Rostering/time tabling:** two staff members with the same skill set may be interchangeable, just like two rooms with the same capacity or technical equipment.

[22] From The MiniZinc Handbook

Search annotations

Specify how to search: `solve::<annotation>`

```
int_search(<variables>,<varchoice>,<constrainchoice>)
```

- `<variables>` is a 1-dim array of `var int` ,
- `<varchoice>` is a variable choice annotation, and
- `<constrainchoice>` is a choice of how to constrain a variable.

Example: *n*-queens

```
solve::int_search(q, first_fail, indomain_min)
satisfy;
```

Similarly we have `bool_search,set_search` .

Search annotations: variable choice

- `input_order` choose in order from the array
- `first_fail` choose the variable with the smallest domain size
- `smallest` choose the variable with the smallest value in its domain
- `dom_w_deg` choose the variable with the smallest value of domain size divided by weighted degree, which is the number of times it has been in a constraint that caused failure earlier in the search.

See the documentation for more.

Search annotations: constrain choice

- `indomain_min` assign the variable its smallest domain value
- `indomain_median` assign the variable its median domain value
- `indomain_random` assign the variable a random value from its domain
- `indomain_split` bisect the variables domain excluding the upper half.

See the documentation for more.

Restart (and warm start)

Return to the top of the search tree (for nonedeterministic search strategies).

- `restart_constant(n)` restart after n nodes searched
- `restart_linear(n)` k-th restart after kn nodes
- `restart_geometric(b,n)` k-th restart after $n \cdot b^k$ nodes

Example:

```
solve::int_search(q, first_fail, indomain_random)
::restart_linear(1000) satisfy;
```

Warm start: supply a partial or suboptimal solution, or ranges for variables to start with (currently not supported in Gecode)

Choosing between models[23]

The better model is likely to have some of the following features:

- smaller number of variables, or at least those that are not functionally defined by other variables
- smaller domain sizes of variables
- more succinct, or direct, definition of the constraints of the model
- uses global constraints as much as possible

In reality all this has to be tempered by how effective the search is for the model. Usually the effectiveness of search is hard to judge except by experimentation.

[23] From The MiniZinc Handbook

Globalizer

The Holy Grail: anyone with domain knowledge can write (efficient!) models. Analyze the model and suggest global constraints.[24]

- https://www.minizinc.org/doc-2.5.0/en/globalizer.html
- Under development
- Only supports a subset of the language, no set or enum types, no command line data.
- Example: queens.mzn
  ```
  gcc(queens,[1,1,1,1,1,1,1,1]);) %no longer supported
  ```
  Instead:
  ```
  global_cardinality(queens,[i|i in 1..n],[1|i in 1..n]);
  ```
- Global cardinality constraints

[24] K. Leo et al, "Globalizing Constraint Models", CP'2013.

Modelling with sets

- A subset $X \subseteq \{1, \ldots, n\}$:

```
var set of 1..n: x;
array[1..n] of var bool: ch;
constraint link_set_to_booleans(x,ch);
% i in x <-> ch[i]
```

- fixed cardinality subset:

```
var set of 1..n: x; constraint card(x) = k;
array[1..k] of var 1..n: x;
constraint all_different(x);
```

- bounded cardinality subset:

```
var set of 1..n: x; card(x)>=l; card(x) <= u;
array[1..u] of var 0..n: x;
constraint alldifferent_except_0(x);
```

- Many global constraints have variants for sets, e.g.:

```
all_different(array [$X] of var set of int: S);
all_disjoint(array [$X] of var set of int: S);
```

MiniZinc and IPython

- iminizinc extension: https://github.com/MiniZinc/iminizinc

See the Jupyter notebook.

Golomb's ruler[25]

Example (golomb)

A Golomb ruler is an imaginary ruler with n marks such that the distance between every two marks is different. Find the shortest possible ruler for a given n.

Try symmetry breaking, adding implicit constraints, and better search strategies.

[25] See R. Barták's practical

The law of leaky abstractions

https://en.wikipedia.org/wiki/Leaky_abstraction

the following constraint

```
dist[i,j] >= (j-i)*(j-i+1) div 2
```

is roughly twice faster than

```
dist[i,j] >= (j-i)*(j-i+1) / 2
```

Optimization levels

- -O0: Disable optimize (–no-optimize)
- -O1: Single pass (default)
- -O2: Two pass
- -O3: Two pass with Gecode
- -O4: 03 and shaving
- -O5: 03 and singleton arc consistency

Higher levels are expensive: better for small yet hard problems (unsatisfiable or solutions rare)

Ramsay's partition

Example (ramsay-partition)

Partition the integers 1 to n into three parts, such that for no part are there three different numbers with two adding to the third. For which n is it possible?

See also Folkman's Theorem.

Minimum common string partition

Example (mcsp)

Find a common partition of two finite strings x, y (into non-overlapping substrings) with the minimum number of blocks.

Example: $x = AGACTG$, $y = ACTAGG$, valid partition: $\{A, A, C, T, G, G\}$, best partition: $\{ACT, AG, G\}$