NOPT042 Constraint programming: Tutorial 2

Jakub Bulín

KTIML MFF UK

October 7, 2020

Chinese remainder theorem

Example (chinese-remainder)

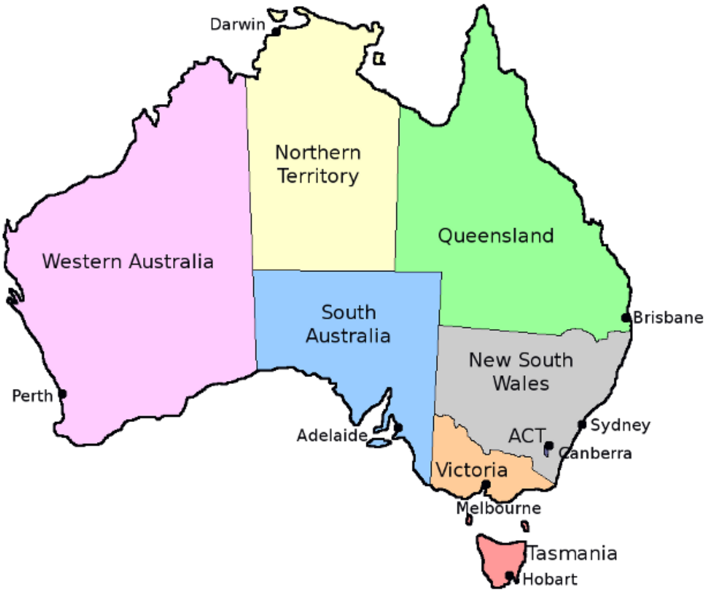
After an indecisive battle, general Han Xin wanted to know how many soldiers of his 42000-strong army remained. In order to prevent enemy spies hidden among his soldiers to learn the number, he decided to use modular algebra: He ordered his soldiers to form rows of 5 and 3 soldiers remained. Then rows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

- What are the *parameters* of our problem?
- Identify the *decision variables* (type, domains[2])
- What are the constraints? (implicit?)
- Is it satisfaction or optimization?

[1] Was this necessary?

[2] as small as possible

Australia



Darwin
Northern Territory
Queensland
Western Australia
South Australia
Brisbane
New South Wales
Perth
Sydney
Adelaide
ACT
Canberra
Victoria
Melbourne
Tasmania
Hobart

Map coloring

Example (color-map)

Create a model to color the map of Australian states and territories [3] with 4 colors (cf. The 4-color Theorem).

Q: What is wrong with this example?[4]

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?

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

[4] 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, _[5]
- Relational operators:
  `= or ==, !=, <, >, <=, >=` [6]
- Logical operators:
  `/\, \/, ->, <-, <->, xor, not`
- Integer arithmetic:
  `+,-,*,div,mod,abs(x),pow(x,y)` [7]
- 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;

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

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

[7] +,- 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[8] based on carry bits for the puzzle DONALD + GERARD = ROBERT.

[8] "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: [9]

```
  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
```

[9]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 $n + 1$ pigeons fit in n holes so that every pigeon has its own hole? [Answer: No.]

```
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.

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)