Discrete Optimization Modelling (with MiniZinc)

Combinatorial Problems are extremely difficult. In the industry, we are more interested in tasks like "Maximize Profit", "Minimize Costs" than questions like "in how many ways...". Some of this problems are proven to be so hard (NP-Complete) that a general solving time shoots up exponentially unless P = NP.

In the real world, we need an answer to these problems anyway. There are many sophisticated solvers such as gecode, flatzinc, Google or-tools and many more. (You might think of Prolog too.) They try to *exploit* substructures and *simplify* the problems with sophisticated techniques gathered from several PhD theses and research papers.

These solvers can be used to solve a variety of problems ranging from several logic puzzles, solving a minesweeper board to serious problems like linear programming, bin packing or scheduling problems.

Contents

Modelling

Modelling is in no way a form *procedural* programming. It is a form of *declarative* programming. To clarify, when modelling we make a very specific and more importantly, non-ambiguous description of the problem.

This description can be handed over to the solver, *who decides how to solve the problem*. That is not to say that all equivalent models of the same problem are equally as powerful or we have absolutely no control over the solving mechanism. Just like the human brain, the solver have varying difficulties solving a problem even though they are technically equivalent. Also, to some extent, it is possible to specify the search technique that the solver is going to use.

Here is a simple example of a MiniZinc model of a Linear Programming Problem

EXAMPLE

We know how to make two sorts of cakes.

A banana cake which takes 250g of self-raising flour, 2 mashed bananas, 75g sugar and 100g of butter, and a chocolate cake which takes 200g of self-raising flour, 75g of cocoa, 150g sugar and 150g of butter. We can sell a chocolate cake for $4.50 and a banana cake for $4.00. And we have 4kg self-raising flour, 6 bananas, 2kg of sugar, 500g of butter and 500g of cocoa.

How many cakes of each sort should we bake to maximize profit?

```
% Baking cakes for the school fete

var 0..100: b; % no. of banana cakes
var 0..100: c; % no. of chocolate cakes

% flour
constraint 250*b + 200*c <= 4000;
% bananas
constraint 2*b  <= 6;
% sugar
constraint 75*b + 150*c <= 2000;
% butter
constraint 100*b + 150*c <= 500;
% cocoa
constraint 75*c <= 500;

% maximize our profit
solve maximize 400*b + 450*c;

output ["no. of banana cakes = ", show(b), "\n",
        "no. of chocolate cakes = ", show(c), "\n"];
```

MiniZinc

MiniZinc is a constraint modelling language that offers a higher level modelling interface for specifying problems to a number of lower level solvers like Flatzinc or gecode.

One can easily download the MiniZinc packages from here

One can just download the command line executables or the IDE if he pleases.

Dissecting the first example

EXAMPLE

We know how to make two sorts of cakes.

A banana cake which takes 250g of self-raising flour, 2 mashed bananas, 75g sugar and 100g of butter, and a chocolate cake which takes 200g of self-raising flour, 75g of cocoa, 150g sugar and 150g of butter. We can sell a chocolate cake for $4.50 and a banana cake for $4.00. And we have 4kg self-raising flour, 6 bananas, 2kg of sugar, 500g of butter and 500g of cocoa.

How many cakes of each sort should we bake to maximize profit?

Imagine for a second that this were a Math class. How would you go about formulating the problem?

Here is a possible formulation:

Let b be the number of banana cakes, c the number of chocolate cakes.

Maximize $p(b, c) = 400b + 500c$

subject to

$$\begin{aligned} 250b + 200c &\leq 4000 \\ 2b &\leq 6 \\ 75b + 150c &\leq 2000 \\ 100b + 150c &\leq 500 \\ 75c &\leq 500 \end{aligned}$$

Notice that our model is exactly the same as that.

First, we begin with declaring our *decision variables*, i.e, the things we wish for the solver to solve.

```
var int: b; % no. of banana cakes
var int: c; % no. of chocolate cakes
```

Then, we specify the constraints:

```
% flour
constraint 250*b + 200*c <= 4000;
% bananas
constraint 2*b  <= 6;
% sugar
constraint 75*b + 150*c <= 2000;
% butter
constraint 100*b + 150*c <= 500;
% cocoa
constraint 75*c <= 500;
```

And then comes what we want the solver to do:

```
% maximize our profit
solve maximize 400*b + 450*c;
```

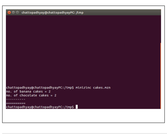
Running Models

If you're using the IDE, this section will not help much but it is still recommended you learn the tricks

The next most important thing you should do is to download the model and run it. Note that minizinc files always end with a `.mzn` extension.

Go to the appropriate directory, and type `minizinc cakes.mzn`.

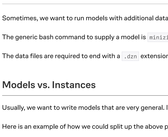
On my computer, I see something like this.



A couple of remarks to make:

- The `----------` marks the end of a solution.
- The `==========` tells you that the solver has proven this solution to be the optimal solution (according to the criteria given).
- You might get an `====UNSATISFIABLE====` implying that your model has no feasible solutions.

If you are interested in seeing *all feasible* solutions, run `minizinc --all-solutions cakes.mzn`.



Sometimes, we want to run models with additional data files. (See the next section).

The generic bash command to supply a model is `minizinc <model.mzn> <data.dzn>`

The data files are required to end with a `.dzn` extension.

Models vs. Instances

Usually, we want to write models that are very general. In contrast, an instance is a combination of a model with some data that is passed on to a solver.

Here is an example of how we could split up the above problem into a generalized model and a data file:

EXAMPLE

`cakes2.mzn`

```
% Baking cakes for the school fete (with data file)

int: flour;  %no. grams of flour available
int: banana; %no. of bananas available
int: sugar;  %no. grams of sugar available
int: butter; %no. grams of butter available
int: cocoa;  %no. grams of cocoa available

constraint assert(flour >= 0,"Invalid datafile: " ++
                  "Amount of flour is non-negative");
constraint assert(banana >= 0,"Invalid datafile: " ++
                  "Amount of banana is non-negative");
constraint assert(sugar >= 0,"Invalid datafile: " ++
                  "Amount of sugar is non-negative");
constraint assert(butter >= 0,"Invalid datafile: " ++
                  "Amount of butter is non-negative");
constraint assert(cocoa >= 0,"Invalid datafile: " ++
                  "Amount of cocoa is non-negative");

var 0..100: b; % no. of banana cakes
var 0..100: c; % no. of chocolate cakes

% flour
constraint 250*b + 200*c <= flour;
% bananas
constraint 2*b  <= banana;
% sugar
constraint 75*b + 150*c <= sugar;
% butter
constraint 100*b + 150*c <= butter;
% cocoa
constraint 75*c <= cocoa;

% maximize our profit
solve maximize 400*b + 450*c;

output ["no. of banana cakes = ", show(b), "\n",
        "no. of chocolate cakes = ", show(c), "\n"];
```

`pantry.dzn`

```
flour = 4000;
banana = 6;
sugar = 2000;
butter = 500;
cocoa = 500;
```

Structure of a MiniZinc Model