Discrete Optimization Specialization: Workshop 11
Shopping for Arrows

1 Introduction

Hou Yi needs magic arrows of each of 30 different kinds in order to break the rock and release the trapped flood waters. He goes to the arrow market to shop for them. Each merchant sells every type of arrow. The laws of the market are very strict. There is a set price for every kind of arrow, and no bargaining is allowed. The shoppers complained about the lack of bargaining and the merchants are not happy with the lack of competition. As a compromise, the merchants now offer deals of the form, "buy X get Y free", where the arrow types which you buy must be no less expensive than the arrow types you get for free. However, a shopper can only use a deal once at each merchant.

Shopping for Arrows — `shopping.mzn`

The data for the problem is encoded as: n is the number of arrow types to buy, *price* is the cost for each arrow type, m is the number of merchants, and the i^{th} deal offers "buy `buy[i]` get `free[i]` free". You can only use the deal at each merchant once. Note that the arrows bought to enable the deal cannot be cheaper than any obtained for free. The aim is to obtain all the arrows for the minimum total price.

An example data file is

```
n = 5;
price = [50,60,90,70,80];
m = 4;
buy = [1,2,1,0];
free = [2,3,1,1];
```

which has 5 different types of arrow to buy, and 4 merchants each with a different deal. Note that you can partially use a deal, as long as you buy enough arrows to enable the deal, you do not have to use all the free arrows.

A model for this problem is given in `shopping.mzn`. It determines for each arrow type a decision

- $-m$ if it was bought to enable a deal at merchant m
- m if it was obtained for free from a deal with merchant m
- 0 if it was simply bought without any deal

For example an optimal answer for the data above is

```
how = [0, 1, 4, 1, -1];
objective = 130;
```

indicating arrows of type 1 were bought without any deal, arrows of type 5 were bought from merchant 1 to enable the deal to get arrows of type 2 and 4 for free, and arrows of type 3 were obtained for free from merchant 4 (who really needs to look at his deal). The total cost was 50 for arrows of type 1 and 80 for arrows of type 5.

The aim of this workshop is to create a new model `shopping01.mzn` which is written specifically for mixed integer programming solvers. You can only use integer variables, and **all** constraints appearing in the model must be linear constraints. Technically if you look at the FlatZinc output, it should only contain constraints `int_lin_eq` and `int_lin_le`.

Rewrite the decisions and the constraints of the model to be expressed only as linear constraints. You should compare the execution of your model on the various data sets using the COIN-OR CBC bundled solver, and compare it with the behaviour of the same solver on the original model.

In order to use the solution checker you will need to add a calculation of the `how` array to the linear model using a line

```
array[ARROW] of ASSIGN: how :: output_only = [ ... ];
```

which calculates the value of the how array from the decisions of your model, using an array comprehension. You will need to wrap any variables you examine in `fix( .. )` in order to treat them as fixed during this output processing.

Shopping for Arrows Again — `shopping_again.mzn`

Can you take the model you have built for a MIP solver, and use it to adjust the original model, still using the same decisions variables, to make it more efficient? Rewrite the model as `shopping_again.mzn` and compare Gecode on the original model and the new one. Sometimes thinking about a model from a different perspective can give you insights to improve the model. It should be the case here!

Note that a much better model for shopping can be built, from scratch, by thinking about dominance relationships that can be used!

2 Technical Requirements

For completing the workshop you will need MiniZinc 2.2.x (`http://www.minizinc.org/software.html`).