Modeling Objects

Peter Stuckey

Modeling Objects

- Often combinatorial problems involve a set of objects which we need to make decisions about
- How can we efficiently represent these objects and their different characteristics?

- Arrays (indexed by object IDs)

Smuggler's Knapsack

A smuggler with a knapsack with capacity 18, needs to choose items to smuggle to maximize profit

Object	Profit	Size
Whiskey	29	8
Perfume	19	5
Cigarettes	8	3

$$\begin{aligned} \text{maximize} \quad & 29W + 19P + 8C \\ \text{subject to} \quad & 8W + 5P + 3C \leq 18 \end{aligned}$$

Smuggler's Knapsack

- But what if the data is different:
  - Capacity 200

Object	Profit	Size
Gold	1300	90
Silver	1000	72
Copper	520	43
Bronze	480	40
Tin	325	33

- We want a model to be reused with different sized data!

Knapsack Model

```
int: n; % number of objects
set of int: OBJ = 1..n;
int: capacity;
array[OBJ] of int: profit;
array[OBJ] of int: size;

array[OBJ] of var int: x; % how

constraint forall(i in OBJ)(x[i] >= 0);
constraint sum(i in OBJ)(size[i] * x[i]) <= capacity;
solve maximize sum(i in OBJ)(profit[i] * x[i]);

output [”x = “, show(x), “\n”];
```

set declarations

array declarations

array lookups

forall expressions

sum expressions

New MiniZinc Features

- Range:
  - l..u is integers {l, l+1, l+2, .., u}
  - Can also be a float range e.g. 1.5 .. 2.745
- Sets
  - `set of` *type*
- Arrays of parameters and variables
  - `array`[*range*] `of` *variable declaration*
- Array lookup
  - *array-name*[*index-exp*]
- Generator expressions
  - `forall`(*i* `in` *range*)(*bool-expression*)
  - `sum`(*i* `in` *range*)(*expression*)

Data Files

```
n = 3;
capacity = 18;
profit = [29,19,8];
size = [8,5,3];
```

knapsack1.dzn

```
$ minizinc knapsack.mzn knapsack1.dzn
```

```
x = [1, 2, 0]
----------
==========
```

solution

solution found

optimal proved

```
n = 5;
capacity = 200;
profit = [1300,1000,520,480,325];
size = [90,72,43,40,33];
```

knapsack2.dzn

```
$ minizinc knapsack.mzn knapsack2.dzn
```

```
x = [1, 1, 0, 0, 1]
```

Modeling Objects

- Create a set naming the objects: `OBJ`
- Create a parameter array for each attribute of the object: `size`, `profit`
- Create a variable array for each decision of the object: `x`
- Build constraints over the set using comprehensions
- Note a model may have many sets of objects

Overview

- (Fixed) Sets to represent sets of objects
- Arrays over the object set to represent
  - object attributes
  - decisions about objects
- Generator expressions
  - To construct expressions over multiple objects

EOF