Modeling for Discrete Optimization Assignment: **Fox Geese Corn**

1 Problem Statement

Every one knows the puzzle of a farmer with a fox, a goose, and a bag of corn to take to the market. He has to cross the river with a boat that can carry one thing, if he ever leaves the fox with the goose, the goose is eaten, or the goose with the corn the corn is eaten, and he has to get everything across the river. This is a generalization of that problem.

The farmer has f foxes, g geese, and c bags of corn on the west side of the river. He has a boat that has capacity k and the time to make t trips. Note that the first trip is west to east, then the second trip is east to west, then the third trip is west to east, etc. When ever the farmer leave some goods alone on the side of the river then by the time he returns the following happens,

- if there is only one kind of good, nothing.
- if there are only foxes and corn, out of boredom one fox eats a bag of corn, its stomach explodes and it dies.
- if there are foxes and geese,
  - if there are more foxes than geese one fox dies in argument over geese, no geese die, and no geese eat any corn
  - if there are no more foxes than geese, each fox eats a goose, and no geese eat any corn.
- if there are no foxes and geese and corn,
  - if there is no more geese than corn each goose eats a bag of corn
  - otherwise all the geese fight, one dies and one bag of corn is eaten.

At the market on the east side the farmer receives pf for each fox, pg for each goose and pc for each corn. The aim is to maximize profit.

2 Data Format Specification

The input is given as a file `data/foxgeesecorn_`p`.dzn`, where p is the problem number, in MiniZinc data format:

f = f;
g = g;
c = c;
k = k;
t = t;
pf = pf;
pg = pg;
pc = pc;

The solution should be given in MiniZinc data format, with three arrays of size t,

`fox = [` number of foxes taken in boat on each trip $1..t$ `];`
`geese = [` number of geese taken in boat on each trip $1..t$ `];`
`corn = [` number of bags of corns taken in each trip $1..t$ `];`
`trips =` number of trips actually made`;`

Note that the entries in the `fox`, `geese` and `corn` arrays should all be 0 for entries after `trips`.

The classic problem, assuming the max trips is 5, the price of the fox is 10, the goose 5 and the corn 2 is given by the data file

```
f = 1;
g = 1;
c = 1;
k = 1;
t = 5;
pf = 10;
pg = 5;
pc = 2;
```

The standard solution of taking the goose across first doesn't work in this version (since the fox eats the corn and dies). It is represented as

```
fox   = [0,0,0,0,0];
geese = [1,0,0,0,0];
corn  = [0,0,0,0,0];
trips = 1;
```

The optimal solution is

```
fox   = [1, 0, 0, 0, 0];
geese = [0, 0, 1, 0, 0];
corn  = [0, 0, 0, 0, 0];
trips = 3;
```

Note that these planning style problems are very difficult to solve if we dont try to build plans in growing size. By default you should use a search strategy that increments the `trips` variables, so you can generate solutions quickly, e.g.

```
solve :: int_search([trips], input_order, indomain_min, complete)
      maximize obj;
```

You may want to use this as the first component of a sequential search.

3 Instructions

Edit `foxgeesecorn.mzn` to solve the optimization problem described above. Your `foxgeesecorn.mzn` implementation can be tested with the command,

```
mzn-gecode ./foxgeesecorn.mzn ./data/<inputFileName>
```

Resources You will find several problem instances in the `data` directory provided with the handout.

Handin Run `submit.py` with the command, `python ./submit.py`. Follow the instructions to apply your MiniZinc model(s) on the various assignment parts. You can submit multiple times and your grade will be the best of all submissions. However, it may take several minutes before your assignment is graded; please be patient. You can track the status of your submission on the *feedback* section of the assignment website.

4 Technical Requirements

For completing the assignment you will need MiniZinc 2.0 (`http://www.minizinc.org/2.0/`) and the gecode solver (`http://www.gecode.org/download.html`). To submit the assignment for grading, you will need to have python 2.7.x installed on your system (installation instructions, `http://www.python.org/downloads/`).