Discrete Optimization Specialization: Assignment 5
Escape to Jin Province

1 Problem Statement

Liu Bei sees the Yuan Shao will certainly be defeated by Cao Cao regardless of his actions. He tells Yuan Shao that he will head to Jin province to bring allies back to help him, but in reality he is plotting to escape to Jin province.

He needs to get to Jin province over a complicated terrain, represented by a grid, where Cao Cao's soldiers are patrolling. Terrain elements in the grid are Plains, Mountain, Forest, City and River. He needs to find a path from where he is now to Jin province such that

- He doesn't enter any mountains, since it's too cold now to travel over them.
- He can enter at most one city. Once he enters a city, he is bound to be recognised and then if he enters another city he will be captured by Cao Cao's soldiers.
- He makes the journey quickly enough. Otherwise the bulk of Cao Cao's army will arrive and he will certainly be captured.
- The journey does not involve too many steps or he will get lost.

His aim is to minimize the number of soldiers he will meet along the route, since that will maximize the chance of a successful escape.

2 Data Format Specification

The input form for the Escape to Jin Province is a file named `data/to_jin_`*p*`.dzn`, where *p* is the problem number, *nrow* is the number of rows in the grid, *ncol* is the number of columns in the grid, *start_row* is the row where he starts, *start_col* is the column where he starts, *delay* is an array mapping terrain to the time in days for traversing a grid square of that terrain type, *timelimit* is the number of days in which he must complete his journey, *terrain* is a 2D array mapping each grid position to its terrain, *Jin* is a 2D array of Booleans saying for each grid position whether it is part of Jin province, *soldier* is a 2D array mapping each grid position to the number of squadrons of soldier stationed in that grid position, and *maxstep* is the maximum number of steps in the path.

Liu Bei has hurriedly constructed a model, and it seems to work for small data, but does not seem good enough to solve the actual problem data. The model he constructed (in file `to_jin.mzn`) is

```
int: nrow;
set of int: ROW = 1..nrow;
int: ncol;
set of int: COL = 1..ncol;

% Plains, Mountain, Forest, City, River
```

```
enum TERRAIN = { P, M, F, C, R };
array[TERRAIN] of int: delay;
int: timelimit;

array[ROW,COL] of TERRAIN: terrain;
array[ROW,COL] of int: soldier;
array[ROW,COL] of bool: Jin;

int: start_row;
int: start_col;

int: maxstep;
set of int: STEP = 1..maxstep;
set of int: STEP0 = 0..maxstep;

var STEP: steps;
array[ROW,COL] of var STEP0: visit;

% start at start position
constraint visit[start_row,start_col] = 1;

% only use steps moves
constraint sum(r in ROW, c in COL)(visit[r,c] >= 1) <= steps;
% reach Jin province
constraint exists(r in ROW, c in COL)(Jin[r,c] /\ visit[r,c] >= 1);

% visit at most one city
constraint not exists(r1,r2 in ROW, c1,c2 in COL)
                     ((r1 != r2 \/ c1 != c2)
                      /\ terrain[r1,c1] = C /\ terrain[r2,c2] = C
                      /\ visit[r1,c1] >= 1 /\ visit[r2,c2] >= 1);

% can't enter Mountain
constraint not exists(r in ROW, c in COL)(terrain[r,c] = M /\ visit[r,c] >= 1);

% visit only one place in every step
constraint forall(r1,r2 in ROW, c1,c2 in COL)
                 (r1 != r2 \/ c1 != c2
                  -> (visit[r1,c1] = 0
                      \/ visit[r2,c2] != visit[r1,c1]));
% steps form a path
constraint forall(s in 1..steps-1)
                 (exists(r1, r2 in ROW, c1, c2 in COL)
                        (abs(r1-r2) + abs(c1-c2) = 1
                         /\ visit[r1,c1] = s /\ visit[r2,c2] = s+1));
```

```
% no shortcuts on path
constraint forall(r1,r2 in ROW, c1,c2 in COL)
                 (abs(r1-r2) + abs(c1-c2) = 1 ->
                  visit[r1,c1] = 0 \/ visit[r2,c2] = 0 \/
                  abs(visit[r1,c1] - visit[r2,c2]) = 1);

% not too much delay
constraint time <= timelimit;
var int: time = sum(r in ROW, c in COL)(delay[terrain[r,c]]*(visit[r,c] >= 1));

% minimize the number of soldiers traversed
solve minimize obj;
var int: obj = sum(r in ROW, c in COL)((visit[r,c] > 0)*soldier[r,c]);

array[TERRAIN] of string: ter = [".", "#", "^", "C", "~"];

output
       ["%"] ++
       [ " " ++ ter[fix(terrain[r,c])] ++ if c = ncol then "\n%" else "" endif
       | r in ROW, c in COL ]
       ++ ["\n%"] ++
       [ if soldier[r,c] > 0 then show_int(2,soldier[r,c]) else " ." endif
         ++ if c = ncol then "\n%" else "" endif
       | r in ROW, c in COL ]
       ++ ["\n"] ++
       ["%"] ++
       [ if fix(visit[r,c]) > 0 then show_int(2,visit[r,c]) else " ." endif
         ++ if c = ncol then "\n%" else "" endif
       | r in ROW, c in COL ]
       ++ ["\nvisit = array2d(ROW,COL,\(visit));\nstesp=\(steps);\ntime = \(time);\nobj = \(ob
       ;
```

An example data file is

```
nrow = 5;
ncol = 5;

start_row = 5;
start_col = 5;

delay = [ 1, 9, 3, 1, 2 ];
timelimit = 8;

terrain = [| P, P, P, P, M
           | P, C, M, P, P
           | P, P, C, P, P
           | P, R, P, C, F
           | M, R, F, P, P |];

Jin = [|  true,  true,  true, false, false
       |  true, false, false, false, false
       |  true, false, false, false, false
       | false, false, false, false, false
       | false, false, false, false, false |];

soldier = [| 3,1,4,8,1
           | 2,1,9,5,4
           | 6,1,4,8,1
           | 3,1,7,1,2
           | 6,1,2,4,1 |];

maxstep = 8;
```

which considers a 5×5 map (in file `to_jin_0.dzn`). Liu Bei's model correctly finds an optimal solution for this small data file. His output also prints in comments the terrain map (“`.`’’ for Plains, “`#`” for Mountains, “`^`” for Forest, “`C`” for City and “`~`” for River) and soldier maps, as well as a representation of the path. For this data it outputs

```
% . . . . #
% . C # . .
% . . C . .
% . ~ . C ^
% # ~ ^ . .
%
% 3 1 4 8 1
% 2 1 9 5 4
% 6 1 4 8 1
% 3 1 7 1 2
% 6 1 2 4 1
%
% . . . . .
% . . . . .
% 7 6 . . .
% . 5 4 3 .
% . . . 2 1
%
visit = array2d(ROW,COL,[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 6, 0, 0,
0, 0, 5, 4, 3, 0, 0, 0, 0, 2, 1]);
steps = 7;
time = 8;
obj = 21;
----------
```

```
==========
```

Showing a path of length 7 steps which requires 8 days to traverse and passes 21 squadrons of soldiers.

The aim of this assignment is to improve the given model so that it is much more efficient. This may involve rewriting the constraints, or even changing the decision variables, but the data file format cannot be changed and the output must have the same format as Liu Bei's model (excluding the parts that are comments). To build a model that solves the largest examples will be very difficult, and is not expected of any but the highest achieving students.

3 Instructions

Edit `to_jin.mzn` to solve the optimization problem described above. Your `to_jin.mzn` implementation can be tested on the data files provided. In the MiniZinc IDE, use the *Run* icon to test your model locally. At the command line use,

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

to test locally. In both cases, your model is compiled with MiniZinc and then solved with the GECODE solver.

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

Handin From the MiniZinc IDE, the *coursera* icon can be used to submit assignment for grading. From the command line, `submit.py` is used for submission. In both cases, 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.[1] It may take several minutes before your assignment is graded; please be patient. You can track the status of your submission on the *programming assignments* section of the course website.

4 Technical Requirements

For completing the assignment you will need MiniZinc 2.0.x and the GECODE 4.4.x solver. Both of these are included in the bundled version of the MiniZinc IDE 0.9.9 (`http://www.minizinc.org`). To submit the assignment from the command line, you will need to have Python 2.7.x installed.

[1] Solution submissions can be graded an unlimited number of times. However, there is a limit on grading of **model submissions**.