Discrete Optimization Specialization: Assignment 12
Refugee Assignment

1 Problem Statement

As Wukong was busy fanning the fires of the Flaming Mountain, a wildfire began that ended up destroying many villages in the mountains. All the villagers fled to the safety of the valley, where they are now living as refugees. Given the crowded conditions, Shennong is endeavouring to split the refugees into k different camps. The aim is to make the camps as harmonious as possible, hence he wants each person to know many people in the same camp. Obviously pairs of people from the same family must be sent to the same camp, and known pairs of people who dislike each other intently must be sent to different camps. Note that if it is more harmonious then Shennong will not use all the campsites.

2 Data Format Specification

The input data for the Refugee Assignment is contained in a file named `data/refugee_`p`.dzn`, where

- $PROBLEM\_STAGE$ is a unique identifier for each testing instance and only for grading purpose (you may ignore it),
- p is the problem number,
- n is total number of people,
- k is the number of campsites available,
- $maxsize$ is the maximum number of people who can fit at any campsite,
- E is a two dimensional array representing the knowledge graph—each row represents two people who know each other,
- ML is a two dimensional array of people numbers—each row represents two people from the same family, and
- CL is a two dimensional array of people numbers—each row represents two people who intensely dislike each other.

The aim is to generate an assignment x of persons to the different campsites.

The data declarations and main decisions are hence:

```
int: PROBLEM_STAGE;  % can be ignored for modeling purposes

int: n;
set of int: PERSON = 1..n;
int: k;
```

```
set of int: CAMPSITE = 1..k;
int: maxsize;
array[int,1..2] of PERSON: E;
set of int: EDGE = index_set_1of2(E);
array[int,1..2] of int: ML;
array[int,1..2] of int: CL;
set of int: FAMILY = index_set_1of2(ML);
set of int: DISLIKE = index_set_1of2(CL);

array[PERSON] of var CAMPSITE: x;
```

The output required for every stage is of the form

```
x = [assignment of campsite to each person];
obj = harmony of the assignment;
```

Shennong has some strong ideas about what constitutes the most harmonious arrangement. Let m be the number of pairs of people who know each other and d_i be the number of people that person i knows.

$$Harmony = \sum_{i=1}^{n}\sum_{j=1}^{n}(2m((i,j) \in E) - d_i d_j)(x[i] = x[j])$$

An alternate way of calculating the value is

$$Harmony = \sum_{c=1}^{k}\left(4m \sum_{(i,j)\in E}(x[i] = c \wedge x[j] = c) - (\sum_{i=1}^{n} d_i(x[i] = c))^2\right)$$

The aim is to maximize the harmony while satisfying that pairs in the same family are at the same campsite, and pairs that dislike each other are at different campsites.

For example given the data file

```
PROBLEM_STAGE = 99; % please ignore
n = 11;
k = 4;
maxsize = 6;
E =  [| 1,2  | 1,3  | 1,6
      | 1,10 | 2,3  | 4,5
      | 4,6  | 4,10 | 5,6
      | 7,8  | 7,9  | 7,10
      | 8,9  | 8,11 |];
ML = [| 1,2  | 3,10 | 6,7  |];
CL = [| 3,5  | 7,10 | 3,11 |];
```

which involves 11 people and 4 campsites each with a maximum capacity of 6.

An optimal solution is given by

```
x = [1, 1, 1, 2, 2, 2, 2, 3, 3, 1, 3];
obj = 226;
```

using 3 of the 4 campsites available. Note how family members are in the same camp ($\{1,2\} \subseteq 1$, $\{3,10\} \subseteq 1$, $\{6,7\} \subseteq 2$), and dislikes are in separate camps ($3 \in 1, 5 \in 2$), ($7 \in 2, 10 \in 1$), and ($3 \in 1, 11 \in 3$).

Note that you may want to change the way your model works depending on the size of the inputs. Some of the inputs will be small, and some much larger.

You may even want to produce an entirely separate approach for solving the problem, by building a bespoke local search solver for the problem and recording the best solution it finds, and using that to create a "fake" MiniZinc model to answer that problem instance (to be explained in more details later).

In order to ease your local search solver in reading the input data files, we provide the model `dzn2ascii.mzn` which will write out the data files as plain ASCII numbers and ignore the `PROBLEM_STAGE` parameter. The format is in order, n, k, $maxsize$, m (number of know relations), $2m$ numbers representing the know relations, ff (number of family relations), $2ff$ numbers representing the family relations, dd (number of dislike relations), and $2dd$ numbers representing the dislike relations. For example the data file above is translated to

```
11
4
6
14
1 2 1 3 1 6 1 10 2 3 4 5 4 6 4 10 5 6 7 8 7 9 7 10 8 9 8 11
3
1 2 3 10 6 7
3
3 5 7 10 3 11
```

3 Instructions

Edit `refugee.mzn` to model and solve the optimization problem described above. You may use whatever modeling and search techniques that you have ever learned in the course. Your `refugee.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 ./refugee.mzn ./data/<inputFileName>
```

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

All of the tests for this assignment are solution submissions. Thus, you should use either your MiniZinc model or your bespoke local search solver to generate the best answer for each data file, and then build a *separate fake model* specifically to make solution submissions. The fake model simply outputs the best answer you can find for the data file, as identified by the unique `PROBLEM_STAGE` parameter. The following is a sample fake model, provided in `dummy.mzn`, with also fake best solutions.

```
int: PROBLEM_STAGE;
int: n;
```

```
set of int: PERSON = 1..n;
int: k;
int: maxsize;
array[int,1..2] of int: ML;
array[int,1..2] of int: CL;
array[int,1..2] of PERSON: E;
solve satisfy;    % dummy solve
output [
  if PROBLEM_STAGE == 1 then
  "x = [1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 3];\n" ++
  "obj = 174;\n"

  elseif PROBLEM_STAGE == 2 then
  "x = [1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 3];\n" ++
  "obj = 174;\n"

  ...

  elseif PROBLEM_STAGE == 12 then
  "x = [1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 3];\n" ++
  "obj = 174;\n"

  else
  "ERROR: Did you modify the PROBLEM_STAGE in any .dzn file?"
  endif
];
```

Note that you need only one fake model for all solution submissions.

This means you can use ANY method to solve the problem. You may wish to build a specific local search algorithm to do so. Alternatively you can just use the same MiniZinc model to generate solutions for some or all submissions.

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

Handin This assignment contains 12 solution submissions. For solution submissions, we will retrieve the best/last solution the solver has found using your model and check its correctness and quality.

From the MiniZinc IDE, the *Submit to 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.

[1] Solution submissions can be graded an unlimited number of times.

4 Technical Requirements

For completing the assignment you will need MiniZinc 2.1.x and the GECODE 5.0.x solver. Both of these are included in the bundled version of the MiniZinc IDE 2.1.x (`http://www.minizinc.org`). To submit the assignment from the command line, you will need to have Python 3.5.x installed.