Discrete Optimization Specialization: Workshop 2
Surrender Negotiations

1 Problem Statement

After defeating the Bagua army formation, Liu Bei must send a party to accept the surrender. To do this he must select a set of negotiators to make up the party. The party must be made up of between l and u negotiators. The negotiators will work in teams of two, quickly changing when they reach an impasse, and each pair of negotiators has a joint negotiation strength. The total negotiation strength of the party is the sum of the negotiation strength of all the pairs in the party and it needs to reach some threshhold value m for the the surrender negotiation to be successful. The aim is to maximize the honor of the negotiation party which is given by the minimum honor of the negotiators in the party, since this will determine how rapidly the negotiations proceed.

2 Data Format Specification

The input form for surrender is a file named `data/surrender_`p`.dzn`, where p is the problem number with l the minimum size of the negotiation party, u the maximum size of the negotiation party, m the minimum total negotiation strength required for success, *NEGOTIATOR* is an enumerated type defining the negotiators, including a *dummy* negotiator, which has a value lower than any other negotiator, *honor* is the honor of each negotiator ($honor[dummy] = 0$), and *joint* is a two dimensional array of integer values defining the negotiation strength of each pair of negotiators (the negotiation strength of all pairs with the dummy negotiator are guaranteed to be 0, and the array is guaranteed to be symmetric).

For example

```
l = 2;
u = 5;
m = 45;
NEGOTIATOR = { D, N1, N2, N3, N4, N5, N6, N7 };
dummy = D;
honor = [ 0, 10, 8, 6, 3, 5, 7, 3 ];
joint = [| 0, 0, 0, 0, 0, 0, 0, 0
         | 0, 4, 7, 8, 4, 2, 9, 7
         | 0, 7, 9, 3, 1, 8, 6, 6
         | 0, 8, 3, 5, 2, 9, 2, 9
         | 0, 4, 1, 2, 8, 2, 4, 1
         | 0, 2, 8, 9, 2, 7, 2, 5
         | 0, 9, 6, 2, 4, 2, 1, 9
         | 0, 7, 6, 9, 1, 5, 9, 4 |];
```

Your model should include a variable `obj` defining the objective, and a variable `party` defining the selected negotation party For example it might output

```
party = {N1, N2, N3, N5, N6};
obj = 5;
```

The negotiation strength is calculated as 56 which is 7 (N1,N2) + 8 (N1,N3) + 2 (N1,N5) + 9 (N1,N6) + 3 (N2,N3) + 8 (N2,N5) + 6 (N2,N6) + 9 (N3,N5) + 2 (N3,N5) + 2 (N5,N6). The template file `surrender.mzn` is provided to demonstrate reading the input data.

Note that you are free to use any set representation for deciding the `party`, but for the automatic checking you need the `party` variable to be a set of NEGOTIATOR. You can define the `party` variable in terms of other representations using a conversion of the results simply for output.

For example to map a fixed cardinality set `s` of OBJ to a set of OBJ you can use

```
array[1..n] of var OBJ: s;  %% actual decision variables
set of OBJ: os :: output_only
               = { fix(s[i]) | i in 1..n };
```

The `output_only` annotation requires that the `os` variables is not used in your model except in output, where the variables `s` will be fixed. This is sufficient for the automatic checking.

You can similarly map a bounded cardinality set `s` of OBJ to a set of OBJ using

```
array[1..n] of var OBJx: s; %% actual decision variables
set of OBJx: os :: output_only %% remove the dummy value
                = { fix(s[i]) | i in 1..n } diff { dummy };
```

3 Technical Requirements

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