# Discrete Optimization Specialization: Assignment 3
# **Beauty Trap: Part 2**

## 1 Problem Statement

Wang Yun plots to help defeat Dong Zhou. He sets out to have both Dong Zhou and his best fighter Lü Bu fall for his foster daughter Diao Chan. Diao Chan is a remarkable beauty. When Lü Bu is invited to Wang Yun's house she performs a dance of seduction in order to trap him into love. This is the dance trap part of the "beauty trap".

Diao Chan must plan the most seductive dance possible. During each phase of the dance she must choose what her legs are doing, what her arms are doing and what her face is doing. There are constraints on sequences of possible actions for legs, arms and face.

Since Lü Bu is easily bored she must also choose the length of the dance to be most attractive, the longer it is the more interesting it has to be. Also there are upper limits on how many times she can use the same move, otherwise Lü Bu's boredom will be overwhelming.

The moves she can make with her legs are: spin, leap, waltz, curtsey, prep (are) for a leap, and stand. She must prep before any leap. Directly after a spin she can only curtsey, prep or stand. Directly after a leap she can only spin, waltz, or stand. She can waltz at most three times in a row. She cannot prep directly after a curtsey. Finally, in between any waltz and a following curtsey there must be a stand.

The moves she can make with her arms are beckon, hands out, hands up, hands wrapped around her body, and neutral. She can beckon only directly after out or up (and hence not as the first move). She can wrapped only directly after beckon or neutral, including the possibility that wrapped is the first move (since the previous move can be considered neutral). She cannot do any of the two same moves in a row except neutral.

The facial gestures she can make are smile, wink, batt (ing) her eyelids, think (ing), glow with beauty, or blank. She cannot make more than two of the same gestures in a row except blank. She cannot think directly after smile (ing) or batt (ing) her eyelids.

The combination of her legs move and arms move at each step determines the value of her dance. The combination of her arms move and facial gesture at each step determines the enticement that she achieves.

Some combinations of legs move and arms moves are impossible. Similarly some combinations of arms move and face would be very off putting and hence must be avoided.

After the dance ends she must stay in the base position stand, neutral, and blank.

Her aim is to maximize the seductive power of the dance, which is given by the value of the dance, plus the enticements of the dance minus the boredom level of Lü Bu multiplied by the length of the dance.

## 2 Data Format Specification

The input form for Dance Trap is a file named `data/dancetrap_*.dzn`, where $p$ is the problem number with *maxlen* the maximum length of the dance, *boredom* the boredom value of Lü Bu, *dance_value* an 2D array mapping a leg move plus an arm move to a dance value (which is negative for disallowed combinations), *entice_value* an 2D array mapping an arm move plus a facial gesture to an enticement value (which is negative for disallowed combinations), *maxlegs* is an array for each leg move giving the maximum number of times it can be used. *maxarms* is a similar array for arms moves. *maxface* is a similar array for facial gestures.

It is guaranteed that dance value of the combination stand and neutral is 0. It is guaranteed that enticement value of the combination neutral and blank is 0. It is guaranteed that the maximum usages of each of the null moves: stand, neutral, and blank is at least *maxlen*, hence doing nothing for the whole dance is always possible.

The data declarations and decisions are hence:

```
enum LEGS = {spin, leap, waltz, curtsey, prep, stand};
enum ARMS = {beckon, out, up, wrapped, neutral};
enum FACE = {smile, wink, batt, think, glow, blank};

int: maxlen;
set of int: STEP = 1..maxlen;
array[LEGS] of int: maxlegs;
array[ARMS] of int: maxarms;
array[FACE] of int: maxface;
constraint assert(maxlegs[stand] >= maxlen, "maxlegs[stand] smaller than maxlen");
constraint assert(maxarms[neutral] >= maxlen, "maxarms[neutral] smaller than maxlen");
constraint assert(maxface[blank] >= maxlen, "maxface[blank] smaller than maxlen");
array[LEGS,ARMS] of int: dance_value;
array[ARMS,FACE] of int: entice_value;
constraint assert(dance_value[stand,neutral] = 0, "incorrect dance_value array");
constraint assert(entice_value[neutral,blank] = 0, "incorrect entice_value array");
int: boredom; % how bored each step makes the viewer

var STEP: len;
array[STEP] of var LEGS: legs;
array[STEP] of var ARMS: arms;
array[STEP] of var FACE: face;
```

An example data file is

```
maxlen = 10;
boredom = 8;
dance_value = [| -1,  5,  3,  2,  2
               |  2,  4,  2, -1,  1
               |  4,  4,  0, -1,  2
               |  5,  1, -1,  2,  2
               |  0,  0,  0,  0,  0
               |  2,  1,  1,  2,  0 |];
entice_value = [|  4,  6,  5, -1,  3, -1
                |  3,  2,  3,  3,  1,  1
                |  2,  4,  5,  4, -1,  0
```

```
            | 5, 3, 2, 5, 6, 0
            | 4, 2, 1, 4, -1, 0 |];
maxlegs = [3, 3, 6, 2, 4, 20];
maxarms = [3, 3, 3, 3, 20];
maxface = [5, 2, 2, 7, 3, 20];
```

which allows a dance of at most length 10, the boredom factor is 8, and e.g. she can use `waltz` at most 6 times, and `curtsey` at most 2. Your model's output should give the decision variables, and the objective value. For example it might output (not necessarily the best solution)

```
len = 9;
legs = [stand, stand, spin, curtsey, spin, curtsey, waltz, waltz, spin, stand];
arms = [wrapped, up, out, beckon, out, beckon, out, beckon, wrapped, neutral];
face = [glow, batt, smile, batt, smile, wink, smile, wink, glow, blank];
obj = 4;
```

Notice how the dance is only length 9, and in step 10 she is base position. You can check that the dance meets all the constraints, and has an objective value of 4 calculated as summing the dance value at each step [2, 1, 5, 5, 5, 5, 4, 4, 2, 0] and the enticement value at each step [6, 5, 3, 5, 3, 6, 3, 6, 6, 0] and subtracting $8 \times 9$ for boredom.

## 3 Instructions

Edit `dancetrap.mzn` to solve the optimization problem described above. Your `dancetrap.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 ./dancetrap.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** This assignment contains 5 solution submissions and 1 model submissions. For solution submissions, we will retrieve the best/last solution the solver has found using your model and check its correctness and quality. For model submissions, we will retrieve your model file (.mzn) and run it on some hidden data to perform further tests.

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. However, there is a limit on grading of **model submissions**.

## 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.