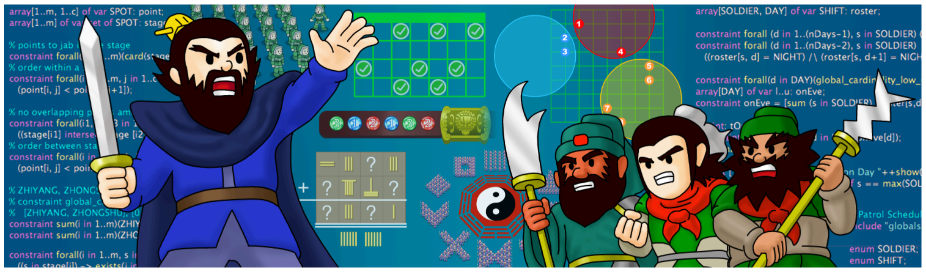
Suitor Schedule

Advanced Modeling for Discrete Optimization: Assignment 3



Problem Statement

Liu Bei is seriously smitten by Sun Shangxiang, the sister of Sun Quan. After he successfully composed a piece of music and a poem for her, she next demanded help scheduling the visits of her various suitors, including Liu Bei. But her real aim is to fit in as many kung fu lessons into the schedule as possible without her mother being aware. Liu Bei's aim is to ensure that he has the most possible time with her during the schedule, but that must be secondary to maximizing the number of kung fu lessons.

Sun Shangxiang has a number of suitors, and her mother is keen that she spends the most possible time with the suitors. She can make use of the Red room, the Gold room and the Blue room for entertaining suitors, except for some timeslots when others are using the rooms. Each suitor has a fixed number of visits to schedule, and the time of the visit has a minimum and maximum time which is dependent on the room where the visit happens. The schedule must respect the time limits, but Sun Shangxiang also wants to finish visits early to squeeze in kung fu lessons, but the false schedule omitting the training must also respect the time limits. The schedule must also include the time to move from one room to another.

Liu Bei has to make a schedule for Sun Shangxiang which respects the time limits with and without the kung fu lessons. The aim is to first maximize king fu lessons and then the time with Liu Bei.

Data Format Specification

The input form for the Suitor Schedule is files named `data/suitor_schedule_*.dzn`, where

- `SUITOR` is an enumerated type of suitors including `LiuBei`,
- `n` is the total number of meetings,
- `suitor` is a mapping from meeting to which suitor attends,
- `mintime` is a 2D mapping from suitors and rooms to the minimum hours required for a meeting with that suitor in that room (you can assume it's at least 1),

- maxtime is a 2D mapping from suitors and rooms to the maximum hours allowed for a meeting with that suitor in that room (you can assume that maxtime[s,r] >= mintime[s,r]),
- move is a 2D mapping giving the move time from one room to another (you can assume its takes 0 to move from a room to itself),
- ndays is the number of days to schedule,
- earliest is the earliest hour in the day for starting a meeting,
- latest is the latest hour in the day for ending a meeting,
- lessontime is the number of hours required for a kung fu lesson,
- minsep is the minimum separation in hours from the end of a kung fu lesson to start of the next,
- usedstart is a mapping from rooms to the start hour when it is used for other purposes each day, and
- useddur is a mapping from rooms to the duration in hours it is used for other purposes each day.

Finally, since this is a multistage assignment, there is a stage which indicates which stage the data file pertains to.

The data declarations and decisions are hence:

```
% scheduling suitors
enum SUITOR;
SUITOR: LiuBei; % which suitor is LiuBei
int: n;         % number of meetings
set of int: MEETING = 1..n;
array[MEETING] of SUITOR: suitor;

enum ROOM = { Red, Gold, Blue };
array[SUITOR,ROOM] of int: mintime;
array[SUITOR,ROOM] of int: maxtime;
constraint forall(s in SUITOR, r in ROOM)
                 (assert(mintime[s,r] >= 1 /\ maxtime[s,r] >= mintime[s,r],
                 "error in mintime/maxtime at [\(s),\(r)]\n"));

array[ROOM,ROOM] of int: move;
constraint forall(r in ROOM)(assert(move[r,r] = 0,"move[\(r),\(r)] != 0\n"));

int: ndays;    % number of days
set of int: DAY = 1..ndays;
int: earliest; % time for earliest meeting start
int: latest;   % time for end of latest meeting

int: lessontime; % time for kung fu lesson;
int: minsep;     % minimum time between consecutive kung fu lessons

array[ROOM] of int: usedstart;  % time others use each room each day
```

```
array[ROOM] of int: useddur;    % durations of others use

enum STAGE = {A,B,C};
STAGE: stage;

set of int: TIME = 0..24*ndays;

array[MEETING] of var TIME: start;
array[MEETING] of var TIME: dur;    % duration in false schedule
array[MEETING] of var ROOM: room;
set of int: KUNGFU = 1..n;
array[KUNGFU] of var int: kungfu; % start time for each lesson
              % unused lessons have start times after 24*ndays
```

An example data file is

```
SUITOR = { S1, S2, S3 };
LiuBei = S1;
n = 2 + 2 + 2 ;
suitor = [ S1, S1, S2, S2, S3, S3 ];

mintime = [| 1, 2, 3
           | 2, 2, 1
           | 3, 3, 3 |];

maxtime = [| 2, 3, 4
           | 2, 4, 3
           | 3, 4, 3 |];

move = [| 0, 1, 1
        | 1, 0, 2
        | 1, 2, 0 |];

ndays = 2;
earliest = 8;
latest = 18;

lessontime = 1;
minsep = 2;

usedstart = [ 10, 12, 14 ];
useddur   = [ 1, 2, 1 ];
stage = A;
```

which considers 3 suitors each with 2 meetings over a 2-day period.

Stage A

In stage A of your model you should ignore the kung fu lessons and transition times between rooms, and simply build a schedule for Sun Shangxiang to meet all the suitors.

Your model's output for this stage and all stages should give the decision variables, and the objective value which is 100 for each kung fu lesson scheduled in the given `ndays` plus the number of hours of meetings scheduled for Liu Bei. For this stage the kung fu lessons can all be set to time greater than $24\times$`ndays`. For example, for the data above your program might give the following solution.

```
start = [32, 8, 16, 12, 13, 36];
dur = [4, 4, 1, 1, 3, 3];
room = [Blue, Blue, Blue, Blue, Red, Red];
kungfu = [49, 49, 49, 49, 49, 49];
obj = 8;
```

This gives a schedule for the 6 meetings where Liu Bei has 8 hours with Sun Shangxiang. Notice how the schedule is impossible given the move restrictions since it finishes a meeting at 13 in the Blue room and then immediately attends a meeting at 13 in the Red room.

```
     1   1   1   1   1
 8   0   2   4   6   8
|B B B B|B|R R R|B|. .
|   LB  |2|  3  |2|

 3   3   3   3   4   4
 2   4   6   8   0   2
|B B B B|R R R|. . . .
|  LB   |  3  |
```

Stage B

In stage B you should take into account of the transition times between rooms, but still ignore the kung fu lessons.

For the sample data above, with instead `stage = B;`, your program might find the following solution.

```
start = [32, 8, 40, 16, 37, 13];
dur = [4, 4, 2, 2, 3, 3];
room = [Blue, Blue, Red, Red, Red, Red];
kungfu = [49, 49, 49, 49, 49, 49];
obj = 8;
```

Now we can see that the transition from the Blue room to the Red room causes an hour delay.

```
     1   1   1   1   1
 8   0   2   4   6   8
|B B B B|.|R R R|R R|.
|   LB  | |  3  | 2 |

 3   3   3   3   4   4
 2   4   6   8   0   2
|B B B B|.|R R R|R R|.
|   LB  | |  3  | 2 |
```

Stage C

Now you should include handling of the kung fu lessons. For the sample data above, with instead `stage = C;`, your program might find the following solution.

```
start = [8, 16, 36, 32, 13, 38];
dur = [4, 2, 2, 3, 3, 3];
room = [Blue, Red, Red, Gold, Red, Red];
kungfu = [11, 17, 34, 49, 52, 55];
obj = 304;
```

Note that this is not the optimal solution!

There are 3 kung fu lessons scheduled at time 11 (the last hour of the first meeting with Liu Bei), at time 17 (the last hour of the second meeting with Liu Bei), and at time 34 (the last hour of the first meeting with S2). Note the (real) kung fu lesson times must be given in non-decreasing order. Notice how the number of hours Liu Bei gets to spend with Sun Shangxiang is 4, although the schedule reads 6, since 2 hours are lost to kung fu lessons.

```
     1   1   1   1   1
 8   0   2   4   6   8
|B B B B|.|R R R|R R|.
|   LB  | |  3  | LB|
       K           K

 3   3   3   3   4   4
 2   4   6   8   0   2
|G G G|.|R R|R R R|. .
|  2  | | 2 |  3  |
     K
```

Note that in order to complete all stages simultaneously (which is required by the Coursera grading infrastructure) you need to construct a single model `suitor_schedule.mzn` that answers all stages, by using the `stage` parameter to determine what constraints need to be applied.

As the **interface to the grader**, your model should contain variable declarations for `start`, `dur`, `room`, and `kungfu`, and it must calculate the correct objective value. Note that you are allowed to use `::output_only` variable declarations, if you want to use a different point of view.

Instructions

Edit the provided `mzn` model files to solve the problems described above. Your implementations can be tested locally by using the **Run + Check** icon in the MiniZinc IDE. Once you are confident that you have solved the problem, submit your solution for grading.

Technical Requirements To complete this assignment you will a new version of the MiniZinc Bundle (`http://www.minizinc.org`). Your submissions to the Coursera grader are currently checked using MiniZinc version 2.6.1.

Handin This assignment contains 6 solution submissions and 3 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. 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]. You can track the status of your submission on the **programming assignments** section of the course website.

[1]Please limit your number of submissions and test on your local machine first to avoid congestion on the Coursera servers