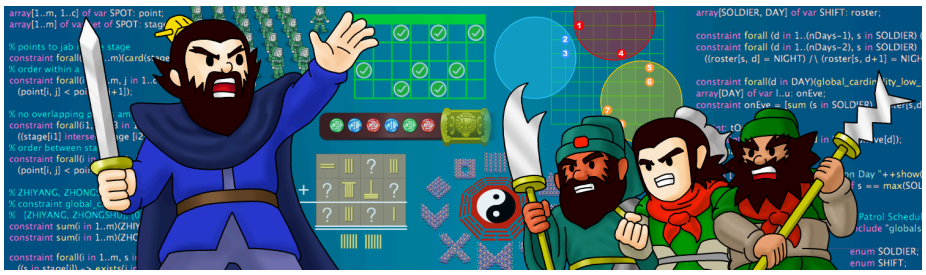
To Borrow Arrows from Thatched Boats

Advanced Modeling for Discrete Optimization: Assignment 4



Problem Statement

Zhou Yu is the chief military commander for Sun Quan. Unhappy with the influence of the scholar Zhuge Liang, Zhou ordered Zhuge to produce 100,000 arrows in 10 days. Zhuge agreed to be punished if he could not fulfil the order, and indeed claimed that "Three days are enough". Zhou believed that Zhuge was looking to fail, and he ordered his troops to not provide any materials to help Zhuge make the arrows.

Zhuge Liang came up with a cunning plan. On the morning of the third day he bribed soldiers crewing boats to fill the boats with straw scarecrows. He then sailed to the camp of Cao Cao, where a thick mist spread over the surface of the river.

When the fleet neared Cao Cao's camp, Zhuge set the soldiers to shout and beat drums to pretend that an attack was imminent. When Cao Cao heard the shouting and drums he assumed a surprise attack on his camp. Since he could see no parts of the river, he gathered 3000 bowmen and ordered them to shoot arrows. The scarecrows were quickly filled with arrows.

Just as day broke, Zhuge ordered the soldiers to return to Sun Quan's camp. The soldiers shouted "Thank you, Cao, for your arrows." On the return to Sun Quan they collected more than 100,000 arrows from the scarecrows.

Data Format Specification

An important feature in the success of this plan was Zhuge Liang's ability to pack the boats full of straw soldiers on the mist filled river in order to catch as many arrows as possible. This is the problem of this assignment.

The input form for the Borrowing Arrows are files named `data/arrows_*.dzn`, where

- `length` is the length of the river close to Cao Cao's camp in which to set boats,
- `width` is the width of the river at the point close to Cao Cao's camp,
- `ntypes` is the number of different kinds of ships available

- `number` is an array mapping ship types to the number of ships of that type available,
- `nroff` is the number of rectangle offset quadruples used to define ships,
- `rectoff` is a two-dimensional array where each row is a rectangle offset quadruple of the form (x offset, y offset, length, width),
- `nshapes` is the number of different shapes (sets of rectangle offset quadruples),
- `shape` is a mapping from shape to set of rectangle offset quadruples,
- `config` is a mapping from each type of ship to its set of possible shapes,
- `mist` is array mapping length positions to where the mist begins on the river,
- `arrows` is an array mapping width positions to how many thousands of arrows are likely to fall at that distance from Cao Cao's camp,
- `price` is an array mapping ship type to price to bribe,
- `budget` is an integer giving available money for bribes, and
- `stage` is an enumerated type variable giving the stage in `{A,B,C,D,E,F}`.

The data declarations are hence:

```
int: length; % length of river near Cao Cao's camp
set of int: LENGTH = 0..length-1;
int: width; % width of river near Cao Cao's camp
set of int: WIDTH = 0..width-1;

int: ntypes;  % number of types of ship
set of int: TYPE = 1..ntypes;
array[TYPE] of int: number;          % number of each type of ship
array[TYPE] of set of SHAPE: config; % configs for each type of ship

int: nshapes; % number of shapes
set of int: SHAPE = 1..nshapes;
set of int: SHAPE0 = 0..nshapes;     % shape 0 used in stage F
array[SHAPE] of set of ROFF: shape;

int: nroff; % number of rectangle offsets
set of int: ROFF = 1..nroff;
array[ROFF,1..4] of int: rectoff; % x offset, y offset, length, width

int: total_ships = sum(number);
set of int: SHIP = 1..total_ships;
array[SHIP] of var LENGTH:  x;
array[SHIP] of var WIDTH:   y;
array[SHIP] of var SHAPE0:  k;

array[LENGTH] of WIDTH: mist; % mist start at each width, goes downward
```

```
array[WIDTH] of int: arrow;   % number of arrows (1000's)
                              % that arrive this far into river
array[TYPE] of int: price;    % price to bribe boat type
int: budget;                  % total budget for bribes
enum STAGE = {A,B,C,D,E,F};
STAGE: stage;
```

The assignment will progress in stages. Some data and decisions won't be necessary until later stages. As the **interface to the grader**, for all stages your model should contain variable declarations for `x`, `y`, and `k`, 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.

Stage A

In stage A of your model you can assume each ship is rectangular and can only be placed in one configuration in the river (no rotations possible). For stage A, models `ntypes` = `nshapes` = `nroff` and you can ignore the `config` and `shape` arrays since they will both always take the values `[ {i} | i in ROFF]`. Also, the offsets in each `ROFF` are guaranteed to be 0. In summary, each ship type i is a rectangle of length `rectoff[i,3]` and width `rectoff[i,4]`.

At this stage you can ignore the `mist`, `arrow`, `price` and `budget` data. The `stage` variable will be `A`.

The output for stage A only needs to have meaningful answers for the `x` and `y` variables. The `k` array can be set to any legitimate value and the objective can be any value. The answer should be a correct packing of the ships into the river with no overlap.

A sample data file for stage A is

```
length = 20;
width  = 10;

ntypes = 3;
number = [3,2,4];
config = [{1},{2},{3}];

nshapes = 3;
shape   = [{1},{2},{3}];

nroff   = 3;
rectoff = [| 0,0,8,1
           | 0,0,2,6
```

```
          | 0,0,4,4 |];

mist   = array1d(LENGTH,[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]);
arrow  = array1d(WIDTH,[0,1,2,4,10,12,7,3,1,0]);
price  = [ 5, 6, 7 ];
budget = 40;
stage  = A;
```

which defines three types of ships, an 8×1 rectangle, a 2×6 rectangle and a 4×4 square.

A possible answer is

```
x = [0, 0, 0, 18, 0, 16, 9, 10, 14];
y = [0, 2, 1, 4, 4, 0, 2, 6, 6];
k = [0, 0, 0, 0, 0, 0, 0, 0, 0];
_objective = 0;
```

showing the packing of 3 ships of type 1, 2 ships of type 2 and 4 of type 3. An illustration of the packing is shown below. Note the (x, y) coordinates of each shape is shown as a coloured circle.

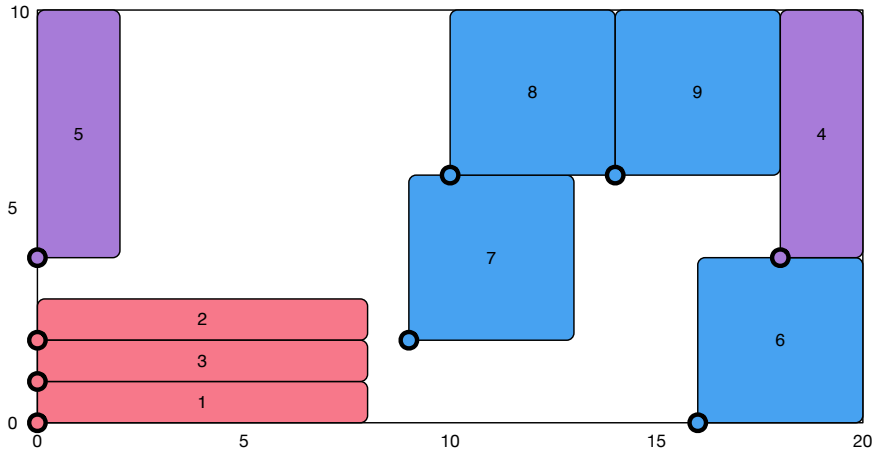


Figure 1: Example solution for Stage A

Stage B

In stage B we will consider more realistic boat shapes, but still not consider rotations. For stage B models `ntypes` = `nshapes`. You can ignore the `config` array since it will always take values `[ { i } | i in SHAPE ]`.

You can continue to ignore the `mist`, `arrow`, `price` and `budget` data.

The output for stage B still only needs to have meaningful answers for the `x` and `y` variables. The `k` array can be set to any legitimate value and the objective can be any value. The answer should be a correct packing of the ships into the river with no overlap.

A sample data file for stage B changes the following parameters, the rest remain unchanged.

```
ntypes = 3;
number = [3,2,2];
config = [{1},{2},{3}];

nshapes = 3;
shape = [{1,2},{3,4,5},{6,7,8}];

nroff = 8;
rectoff = [| 0,1,2,1 | 2,0,3,3
           | 0,1,1,2 | 1,0,4,4 | 5,0,2,2
           | 2,0,1,1 | 1,1,3,3 | 0,4,5,1
           |];
stage = B;
```

A possible answer is

```
x = [0, 15, 0, 9, 9, 15, 4];
y = [1, 0, 4, 6, 2, 4, 4];
k = [0, 0, 0, 0, 0, 0, 0];
_objective = 0;
```

showing the packing of 3 ships of type 1, 2 ships of type 2 and 2 of type 3. An illustration of the packing is shown below

Stage C

In stage C we will take into account the position of the mist on the river. Each of the boats must be hidden under the mist. The `mist` array shows the height at which the mist begins. You can continue to ignore the `arrow`, `price` and `budget` data.

The stage B data cannot be packed into the area covered by the mist, and so the stage C data file changes the positions of the mist (to be non-zero) and the numbers of each ship type, as well as the stage, to be

```
mist   = array1d(LENGTH,[0,0,0,3,3,3,6,6,6,6,2,2,2,4,4,4,0,0,5,5]);
number = [2,2,1];
stage  = C;
```

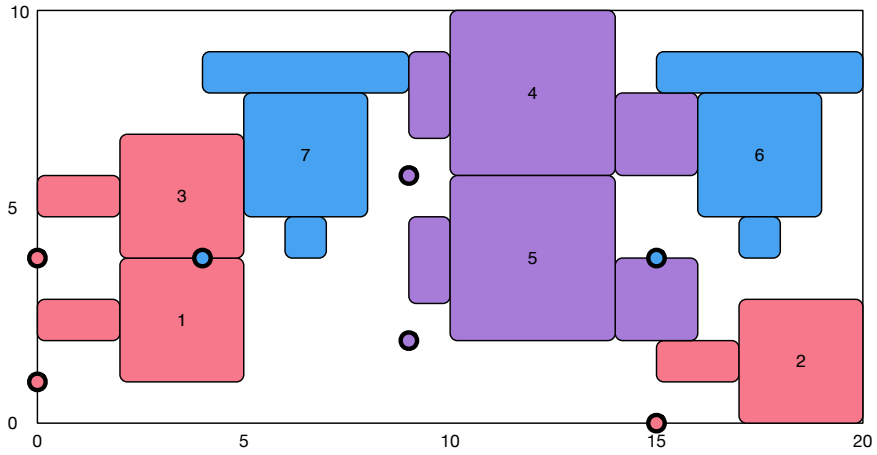


Figure 2: Example solution for Stage B

The output for stage C still only needs to have meaningful answers for the `x` and `y` variables. The `k` array can be set to any legitimate value and the objective can be any value. The answer should be a correct packing of the ships into the river with no overlap, where all parts of all ships are under the mist.

A possible answer is

```
x = [15, 1, 4, 11, 0];
y = [7, 3, 6, 4, 5];
k = [0, 0, 0, 0, 0];
_objective = 0;
```

showing the packing of 2 ships of type 1, 2 ships of type 2 and 1 of type 3. An illustration of the packing is shown below. The light blue area is the part of the river not covered by mist.

Stage D

In stage D we will take into account positioning the ships to gain the maximum possible numbers of arrows. You can continue to ignore the `price` and `budget` data. The `stage` variable will `D`.

The output for stage D needs to have meaningful answers for the `x` and `y` variables. The `k` array can be set to any legitimate value. The objective should be set to the number of arrows collected by all ships. The ships need to be packed without overlap in the misty part of the river.

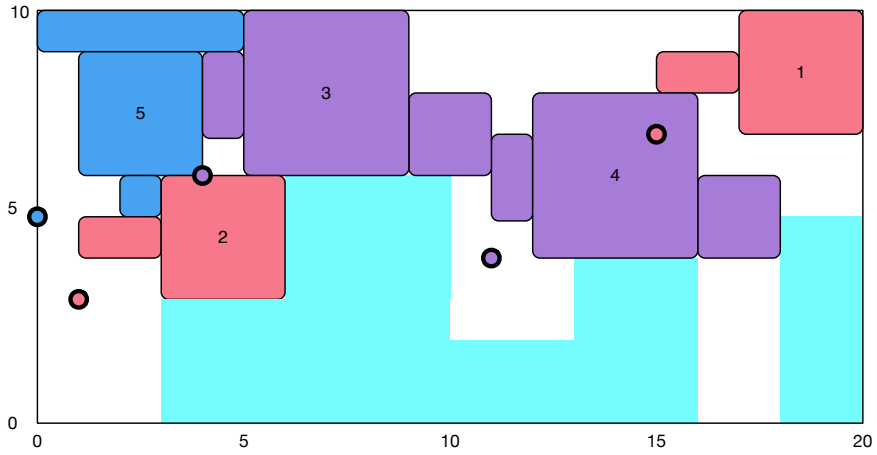


Figure 3: Example solution for Stage C

The answer from stage C extended for stage D is

```
x = [15, 1, 4, 11, 0];
y = [7, 3, 6, 4, 5];
k = [0, 0, 0, 0, 0];
_objective = 416;
```

showing the number of (thousands of) arrows collected. An illustration of the packing is shown below, with the number of thousands of arrows collected from each grid location shown.

Stage E

In stage E we will allow rotation and other reconfiguration of the boats. Now each boat will have a set of possible configurations representing rotations or other reconfiguration.

You can continue to ignore the `price` and `budget` data. The `stage` variable will `E`.

The output for stage E now needs to have meaningful answers for the `x`, `y` and `k` variables. The `k` array holds the shape selected for each ship from the configurations available for its type. The objective should be the amount of arrows collected by the ships. The answer should be a correct packing of the ships into the river with no overlap.

A sample data file for stage E changes the following parameters,

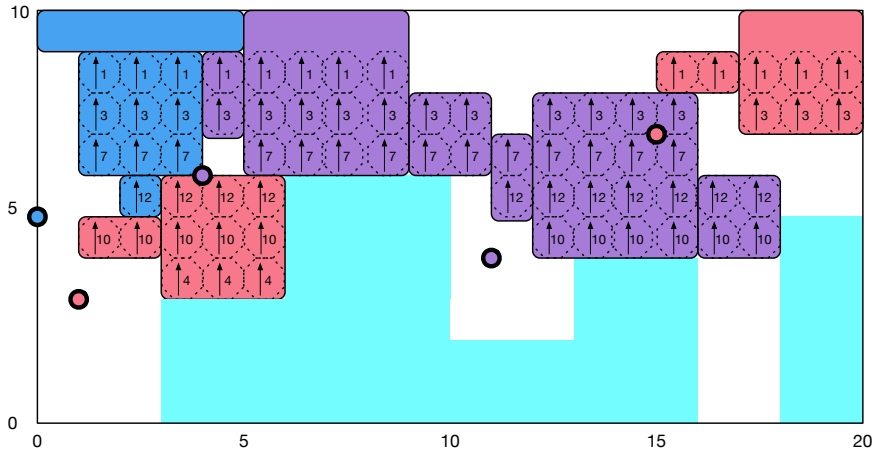


Figure 4: Example solution for Stage D

```
ntypes = 3;
number = [3,2,2];
config = [{1,2,3,4},{5,6,7,8},{9,10,11,12}];

nshapes = 12;
shape = [{1,2},{3,4},{3,5},{6,7},
         {8,9,10},{11,12,13},{14,15,16},{17,18,19},
         {20,21,22},{23,21,24},{25,21,26},{27,21,28}];

nroff = 28;
rectoff = [| 0,1,2,1 | 2,0,3,3
           | 0,0,3,3 | 1,3,1,2
           | 3,1,2,1
           | 0,2,3,3 | 1,0,1,2

           | 0,1,1,2 | 1,0,4,4 | 5,0,2,2
           | 1,6,2,1 | 0,2,4,4 | 0,0,2,2
           | 6,1,1,2 | 2,0,4,4 | 0,2,2,2
           | 1,0,2,1 | 0,1,4,4 | 2,5,2,2

           | 2,0,1,1 | 1,1,3,3 | 0,4,5,1
           | 0,2,1,1 | 4,0,1,5
           | 2,4,1,1 | 0,0,5,1
           | 4,2,1,1 | 0,0,1,5 |];
stage = E;
```

which uses the same 3 ship types as for stage B, but now with 4 possible

rotations.

A possible answer is

```
x = [17, 12, 10, 5, 0, 12, 0];
y = [5, 7, 2, 6, 6, 3, 2];
k = [2, 3, 2, 7, 5, 10, 12];
_objective = 524;
```

An illustration of the packing is shown below

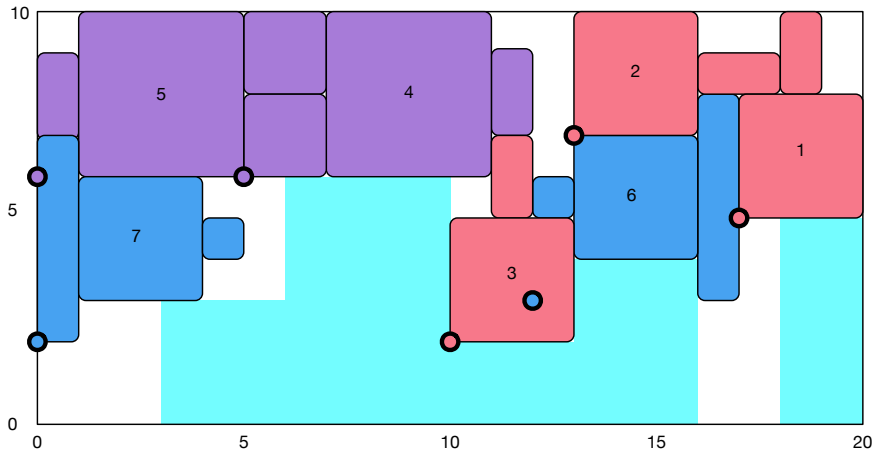


Figure 5: Example solution for Stage E

Stage F

In stage F we will take into account the limits of the budget. Zhuge Liang can only afford a certain number of boats, for each boat of type `t` he must pay `price[t]` up to the limit of `budget`.

The output for stage F now needs to have meaningful answers for the `x`, `y` and `k` variables. The `k` array holds the shape selected for each ship from the configurations available for its type, or 0 if the ship is not used. An unused ship can have any coordinates. The objective should be the correct value. The answer should be a correct packing of the ships into the river with no overlap, and under the mist.

A sample data file for stage F changes the following parameters,

```
stage = F;
```

A possible answer is

```
x = [2, 14, 17, 0, 10, 4, 5];
y = [0, 4, 5, 2, 2, 5, 4];
k = [0, 2, 2, 8, 6, 12, 0];
_objective = 503;
```

An illustration of the packing is shown below.

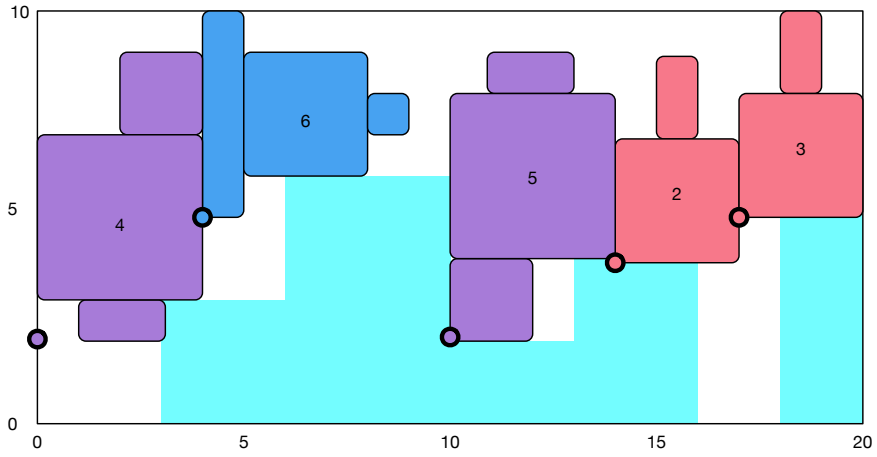


Figure 6: Example solution for Stage F

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

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 12 solution submissions and 6 model submissions. For solution submissions, we will retrieve the best/last solu-

tion 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