离散优化建模：作业八
草船借箭

1 问题描述

周瑜是孙权的军事统领。周瑜对于学者诸葛亮所带来的影响很不快，于是命令诸葛亮在 10 天生产 100,000 支箭。诸葛亮答应如果 10 天之内不能完成任务就接受处罚，实际上他说：“三天足矣。”周瑜相信诸葛亮会失败，他命令他的部队不要给诸葛亮造箭提供任何材料。

诸葛亮想到了一条妙计。在第三天的早上他雇佣士兵召集船只，并在船只上绑满稻草人。然后他在大雾弥漫河面之时把船开向曹营。

当船队靠近曹营时，诸葛亮让士兵们大喊并击鼓，假装马上要进攻。当曹操听见这些呐喊跟鼓声时，他觉得这是一次奇袭。由于他不能看到河面，他召集了 3000 弓箭手，令其放箭。很快稻草人身上便插满了箭。

黎明将至，诸葛亮命令士兵们返回孙营。士兵们大喊：“谢曹丞相的箭。”返回孙权营地之后，他们在稻草人身上收集了多于 100,000 根箭。

2 数据格式说明

这次计划重要的一步是诸葛亮需要将绑满稻草人的船在河上尽量排开，以便可以接到最多的箭。这就是本次作业的问题。

草船借箭问题的输入是名为 `data/arrows_`p`.dzn` 的文件，其中 p 是问题的序号。$length$ 是靠近曹营可用于放置船只的河的长度，$width$ 是靠近曹营的河的宽度。$ntypes$ 表示有多少种不同类型的船只可供使用。$number$ 是一个数组，代表不同的船只可提供的数量。$nroff$ 代表有多少种用于定义船只形状的带偏移的矩形。$rectoff$ 是一个二维数组，每一行是一个带偏移的矩形的元組（x 位移，y 位移，长度，宽度）。$nshape$ 代表船只形状的数量（船只形状由一个带偏移矩形的集合表示）。$shape$ 是一个数组，代表船只形状及其对应的元组的集合。$config$ 代表不同种类的船只与其对应的船只摆放状态。$mist$ 是一个数组，代表不同位置雾气弥漫的范围。$arrows$ 是一个数组，代表距离曹营不同宽度的位置上会有多少根箭会落在该区域，其中 1 代表 1000 根箭。$price$ 是一个数组，代表不同的船只需要多少钱来雇佣。$budget$ 是一个整数，说明有多少钱可以用于收买船只。$stage$ 是一个枚举型变量，它的值可以是 {A,B,C,D,E,F} 中的一个。

因此数据声明如下：

```
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;
```

这次作业会分不同的阶段进行。有一些数据和决策变量在之前的阶段可能并不需要。每一个阶段的输出都应该按照以下格式：

```
x = [船只的 x 坐标];
y = [船只的 y 坐标];
k = [每艘船所选择的形状]
tarrows = 船只收集的箭的总数量（其中 1 代表 1000 根箭）;
```

3 A 阶段

在 A 阶段，你的模型可以假设所有船的形状都是长方形，而且在河中只能有一种放置形态（没有旋转的可能）。在 A 阶段，ntypes = nshapes = nroff。而且你可以忽略 config 和 shape 这两个数组，因为他们总是取相同的数值 $[\{i\}|i \text{ in } ROFF]$。同样地，对于每一个 ROFF，偏移可以确定为 0。总之，每种船只 i 是一个长度为 rectoff[i,3]，宽度为 rectoff[i,4] 的长方形。

在这个阶段你可以忽略 mist, arrow, price 和 budget 数据。stage 变量取值为 A。

阶段 A 的输出只需要 x 和 y 为可行解即可。k 数组可以设置成任意合法的数值，而 $tarrows$ 可以是任意的数值。问题的解必须是可以把船放置在河中而没有重叠的正确打包。

一个 A 阶段的样本数据文件如下：

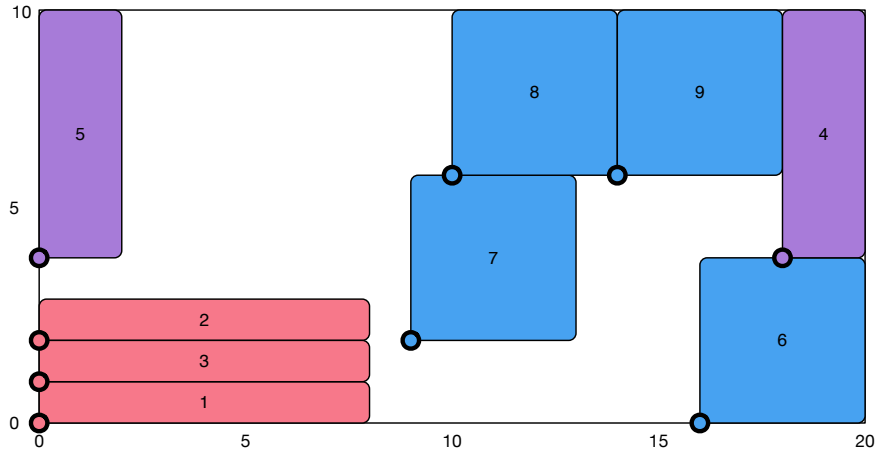
```
length = 20;
width = 10;

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

上面定义了三种不同的船，有 8×1 的长方形，2×6 的长方形，和 4×4 的正方形。

一个可行解是：

```
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];
tarrows = 0;
```

以上解表示需要类型一的船有 3 条，类型二的船有 2 条，类型三的船有 4 条。下图说明了船的排布。需注意到每一艘船的 (x,y) 坐标用不同颜色的圆圈表示。



4 B 阶段

在 B 阶段，我们需要考虑更加现实的的船只形状，不过仍然不需要考虑旋转。在 B 阶段，`ntypes` = `nshapes`。你可以忽略 `config` 数组，这个数组会取值为 $[\{i\}|i \in SHAPE]$。

你仍然可以不用考虑 `mist`, `arrow`, `price` 和 `budget` 数据。

阶段 B 的输出只需要 x 和 y 为可行解即可。k 数组可以设置为任意合法的数值，$tarrows$ 可以设置为任意数值。问题的解必须是可以把船放置在河中而没有重叠的正确打包。

B 阶段的一个样本数据文件会改变以下参数，其他的则维持不变：

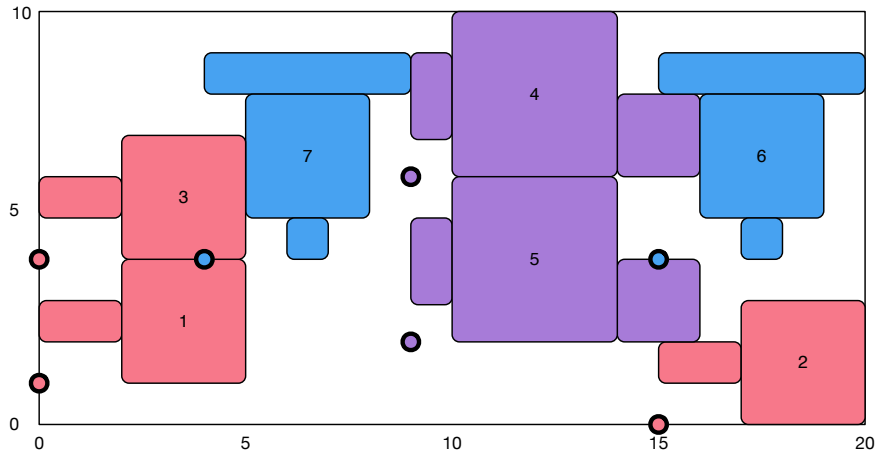
```
ntypes = 3;
number = [3,2,2];
config = [{1},{2},{3}];

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;
```

一个可行解如下：

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

这表示了 3 条类型一的船，2 条类型二的船和 2 条类型三的船的放置。下图说明了船的排布。



5 C 阶段

在 C 阶段，我们需要考虑雾气在河面上的位置。每一艘船都需要隐藏在大雾之下。`mist` 表明雾气开始的位置。你仍然可以忽略 `arrow`,`price` 和 `budget` 数据。

在 B 阶段的模型不能把船放置在大雾之下，所以 C 阶段的数据文件需要改变雾气开始的位置不同形状的船的数量，以及 stage 变量：

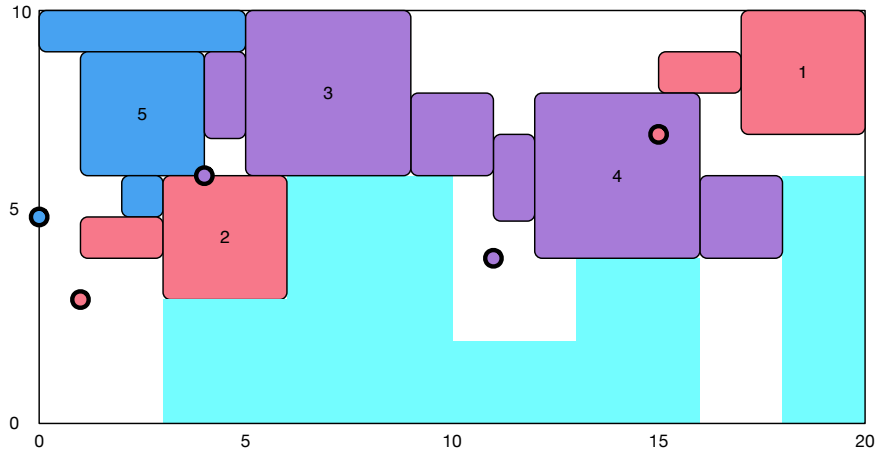
```
mist = array1d(LENGTH,[0,0,0,3,3,3,6,6,6,6,2,2,2,4,4,4,0,0,5,5]);
number = [2,2,1];
stage = C;
```

C 阶段的输出只需要 x 和 y 为可行解即可。k 数组可以设置为任意合法的变量，$tarrows$ 则可以设置为任意的值。问题的解必须是可以把船放置在河中而没有重叠的正确打包，而且所有船的所有部分需要被覆盖在大雾之下。

一个可行解如下：

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

这表示了 2 条型号一的船，2 条型号二的船和 1 条型号三的船的放置。下图说明了船的排布。其中亮蓝色的部分是河上没有被大雾覆盖的部分。



6 D 阶段

在 D 阶段，我们需要考虑让船摆放得可以接收最多数量的箭。你仍然可以忽略 `price` 和 `budget` 数据。`stage` 变量取值为 D。

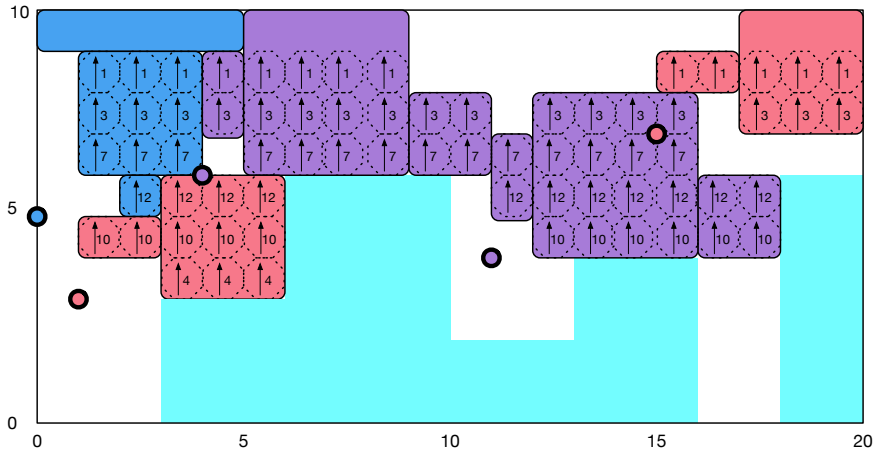
D 阶段的输出需要 x 和 y 为可行解。k 数组可以设置为任意合法的值。$tarrows$ 的数值应该为所有船可以获得的箭的数量。问题的解必须是可以把船放置在河中而没有重叠的正确打包，而且在大雾覆盖之下。

C 阶段的解拓展到 D 阶段中应该是：

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

```
tarrows = 416;
```

这个解说明了收集有多少箭。下图说明了船的排布和在每个方格中有多少箭（1 代表 1000 支箭）。



7 E 阶段

在 E 阶段，我们允许船有旋转。现在每一艘船都有一些可能的摆放方式，代表可能的旋转或其他放置方法。

你仍然可以忽略 `price` 和 `budget` 数据。`stage` 变量取值为 E。

E 阶段的输出的解现在需要考虑 x, y 和 k 变量。k 数组代表每一艘船在可选的放置方式中选取的一种。*tarrows* 需要表示所有船获得的箭的总数。问题的解必须是可以把船放置在河中而没有重叠的正确打包，而且所有船在大雾覆盖之下。

一个 E 阶段的样本数据文件更改了如下参数：

```
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
```

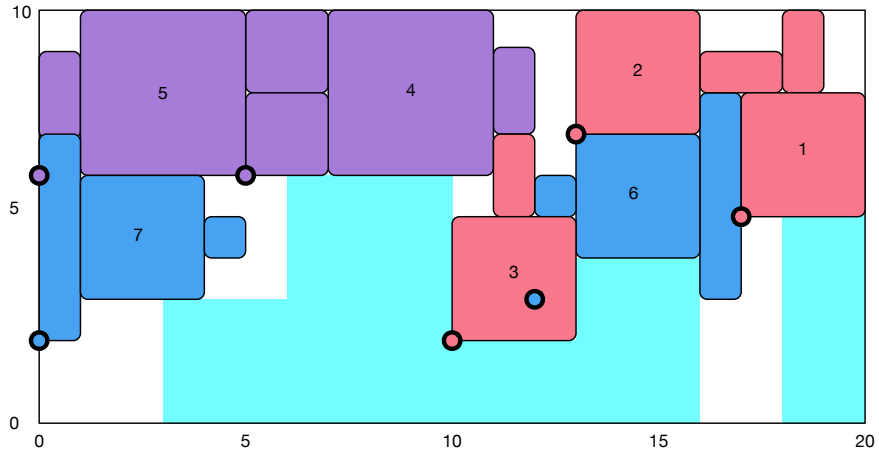
```
            | 0,0,3,3 | 1,3,1,2
            | 3,1,2,1
            | 0,2,3,3 | 1,0,1,2

            | 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;
```

以上数据使用跟阶段 B 相同的三种类型的船只，不过现在每艘船有 4 种不同的摆放方式。

一个可行解为：

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

下图说明了船的排布。



8 F 阶段

在 E 阶段需要考虑预算限制。诸葛亮只能给出一定的金额来购买船只。对每艘租用的类型 t 的船，他必须支付 price[t], 但总额不能超过 `budget`。

F 阶段的输出需要是考虑 x, y 和 k 变量。k 数组需要表示每一艘船可选的放置方式中选取一种，或取值为 0，代表这艘船没有被使用。一艘没有使用的船的坐标可以是任意的数值。*tarrows* 表示获得的箭的数目。。问题的解必须是可以把船放置在河中而没有重叠的正确打包，而且所有船需要在大雾覆盖之下。

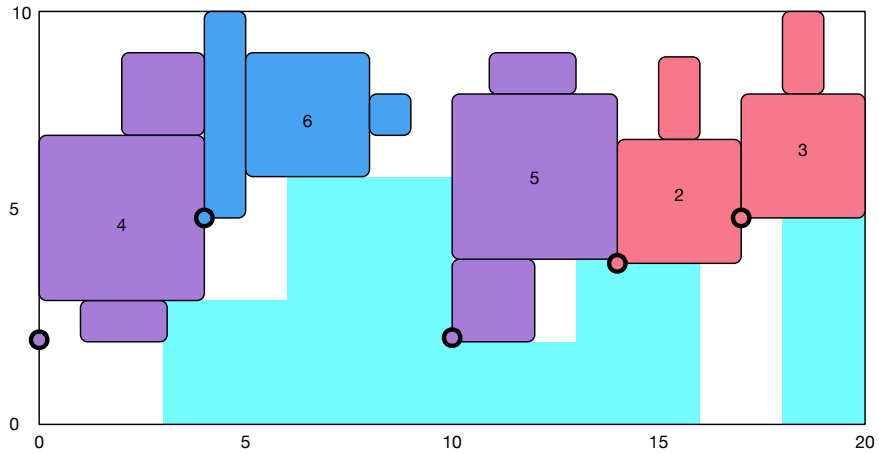
F 阶段的样本数据文件更改了如下参数：

```
stage = F;
```

一个可行解如下：

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

下图说明了船的排布。



按照 Coursera 评分系统的要求，你需要建构一个模型 `arrows.mzn` 去完成全部三个阶段。在你的模型中，你需要以 `stage` 参数去判断应该考虑哪些约束是有效的以符合不同阶段的要求。

9 指引

你可以编辑 `arrows.mzn` 模型文件来解决上述优化问题。你实现的模型 `arrows.mzn` 可以用提供的数据文件进行测试。在 MiniZinc IDE 中，你可以通过点击 *Run* 按钮在本地测试和运行。或

者在命令行中输入

```
mzn-gecode ./raid.mzn ./data/<inputFileName>
```

进行本地测试和运行。两种情况下，你的模型都是用 MiniZinc 进行编译同时用 GECODE 求解器求解。

参考资料 你可以在 `data` 文件夹下找到讲义中的几个问题实例（的数据文件）。

提交作业 这次的作业包含有 12 个答案提交部分和 6 个模型提交部分。对于答案提交部分，我们将会提交求解器求解你的模型所得到的最好 / 最后的答案，然后检查它的正确性和得分。对于模型提交部分，我们将会提交你的模型文件 (.mzn) 然后用一些隐藏的数据文件来做进一步检查。

在 MiniZinc IDE，点击 *coursera* 图标可以用于提交作业。若采用命令行方式，`submit.py` 可以用于提交作业。无论采用那种方法，你都需要根据本指引中的要求完成作业各部分的 MiniZinc 模型。你可以多次提交，最终作业分数是你的最高的一次。[1]作业的打分过程可能需要几分钟，请耐心等候。你可以在课程网站上的*编程作业* 版块查看你的作业提交状况。

10 软件要求

为了完成作业,你需要安装MiniZinc 2.1.x和GECODE 5.0.x 求解器。这些软件都会包含在MiniZinc IDE 2.1.2 (`http://www.minizinc.org`)的集成版本中。如果你需要通过命令行提交作业，你需要安装Python 3.5.x。

[1]答案提交部分并没有次数限制。但是，模型提交部分只能提交有限次。