

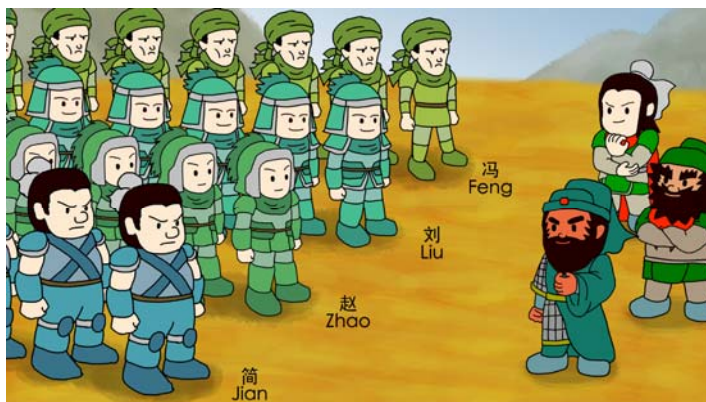


第一步

李浩文、彼得·斯塔基



军队招募



四个村庄

	冯 Feng	刘 Liu	赵 Zhao	简 Jian
	6	10	8	40
	\$13	\$21	\$17	\$100
	1000	400	500	150

3

约束和目标

约束



目标



4

关羽的建议

最好而又最贵的士兵

- 100名简家村士兵
- 战斗力4000



5

张飞的建议

大量最便宜的士兵

- 769名冯家村士兵,
- 战斗力4614



我们可以做得更好吗？

6



组建军队 (army.mzn)

```
solve maximize 6*F + 10*L + 8*Z + 40*J;

int: budget = 10000;
constraint 13*F + 21*L + 17*Z + 100*J
    <= budget;

var 0..1000: F;
var 0..400: L;
var 0..500: Z;
var 0..150: J;

output ["F = \(F), L = \(L), Z = \(Z),
    J = \(J)"];
```

7

组建军队 (army.mzn)

```
solve maximize 6*F + 10*L + 8*Z + 40*J;

int: budget = 10000;
constraint 13*F + 21*L + 17*Z + 100*J
    <= budget;

var 0..1000: F;
var 0..400: L;
var 0..500: Z;
var 0..150: J;

output ["F = \(F), L = \(L), Z = \(Z),
    J = \(J)"];
```

```

    maximize 6F + 10L + 8Z + 40J
    subject to
    13F + 21L + 17Z + 100J <= budget
    0 < F < 1000
    0 < L < 400
    0 < Z < 500
    0 < J < 150
    F, L, Z, J <= 2L budget = 10000
  
```

8

组建军队 (army.mzn)

```
int: budget = 10000;
```

参数定义

```
var 0..1000: F;
```

```
var 0..400: L;
```

```
var 0..500: Z;
```

```
var 0..150: J;
```

决策变量声明

```
constraint 13*F + 21*L + 17*Z + 100*J  
    <= budget;
```

约束

```
solve maximize 6*F + 10*L + 8*Z + 40*J;
```

目标

```
output ["F = \"(F)\", L = \"(L)\", Z = \"(Z)\",  
    J = \"(J)\""];
```

输出

9

参数

MiniZinc中有两种变量

第一种变量是参数

- 与标准编程语言中的变量相似。它们必须被赋值（但只能是一次）
- 它们被声明为某一类型如 `int`, `float`, 或者 `bool`（或者是一个范围 / 集合）
- 可以（在声明中）添上 `par` 作为前缀，但这不是强制的（可选的）
- 下面的表达式在逻辑上是等价的
 - `int: i=3;`
 - `par int: i=3;`
 - `int: i; i=3;`

10

决策变量

❏ 另一种变量是**决策变量**：

- 与数学中的变量相似
- 用var与一个类型（或者一个范围 / 集合）来声明
- 也可以由一个具有固定值的表达式来**赋值**（仅一次）

❏ **范围**：写作 $l..u$

- 一个从 l 到 u 的连续整数序列

❏ 下面的表达式在逻辑上是等价的

- `var int: i; constraint i >= 0; constraint i <= 4;`
- `var 0..4: i;`
- `var {0,1,2,3,4}: i;`

❏ 下面的表达式也是逻辑上等价的

- `var int: i = x + 3;`
- `var int: i; constraint i = x + 3;`

11

约束

❏ 基本的算术约束是基于标准算术关系操作符来创建的

`= != > < >= <=`

❏ 在MiniZinc中，约束以下面的形式来表示

`constraint <约束表达式>`

12



输出与字符串

- 一个输出项具有以下形式
`output <字符串列表>;`
- 字符串常量与 C 中的相似
 - 写在 " " 中
- 它们不会超过一行
- 反斜杠用于（输入）特殊字符如 `\n \t` 等等
- 内建函数有
 - `show(v)` 以字符串形式输出 `v` 的值
 - `\(v)` 在字符串常量中显示 `v`
 - `"house" ++ "boat"` 用于连接字符串

13

求解模型

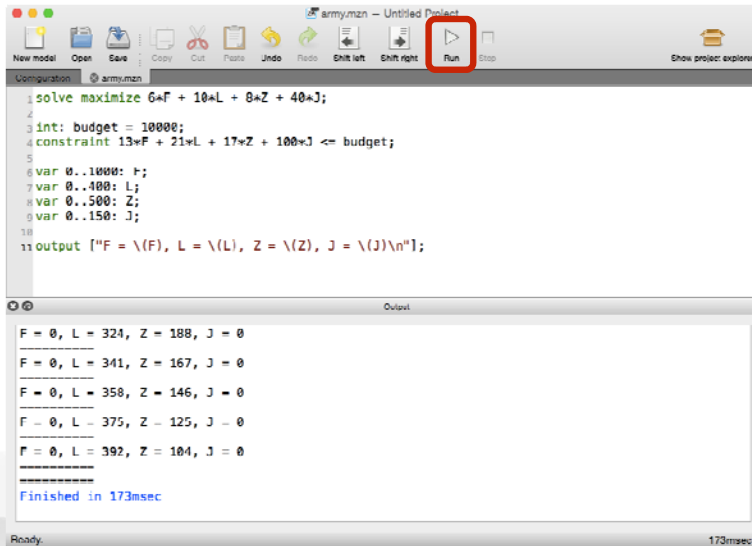
- 我们可以运行如下命令求解 MiniZinc 模型
`$ minizinc army.mzn`
- 运行结果为
`F = 0, L = 392, Z = 104, J = 0`

=====
- 战斗力为 **4752**，人数是 496
- 直线 ----- 标示解
- 直线 ===== 标示没有更好的解（也就是说这是最优解）
- MiniZinc 模型文件以 `.mzn` 为后缀
- MiniZinc 也有集成开发环境（IDE）

14

求解模型

- ❏ 我们可以在IDE中按Run来运行



```
1 solve maximize 6*F + 10*L + 8*Z + 40*J;
2
3 int: budget = 10000;
4 constraint 13*F + 21*L + 17*Z + 100*J <= budget;
5
6 var 0..10000: F;
7 var 0..400: L;
8 var 0..500: Z;
9 var 0..150: J;
10
11 output ["F = \F), L = \L), Z = \Z), J = \J)\n");
```

Output:

```
F = 0, L = 324, Z = 188, J = 0
F = 0, L = 341, Z = 167, J = 0
F = 0, L = 358, Z = 146, J = 0
F = 0, L = 375, Z = 125, J = 0
F = 0, L = 392, Z = 104, J = 0
=====
Finished in 173msec
```

15

小结

- ❏ MiniZinc使我们可以直接描述 / 表示并解决离散优化问题
- ❏ 线性模型是建模中最常见的
 - 第二次世界大战不久之后，线性规划就开始应用
 - 整数线性规划更困难 (NP-hard)
 - 依然是大多数现实世界离散优化的基础

16



图像引用

所有图像由Marti Wong设计提供, © 香港中文大学与墨尔本大学 2016

17