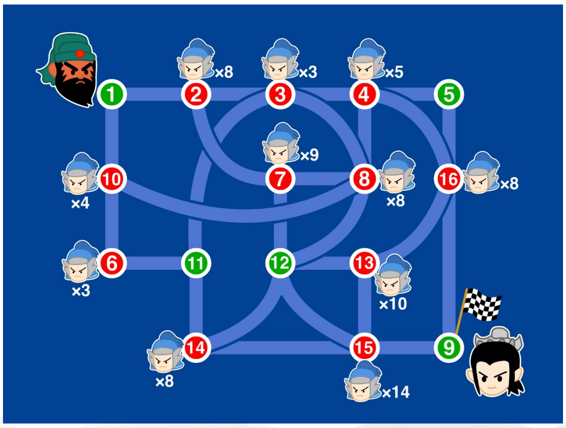
Missing Solutions

Jimmy Lee & Peter Stuckey

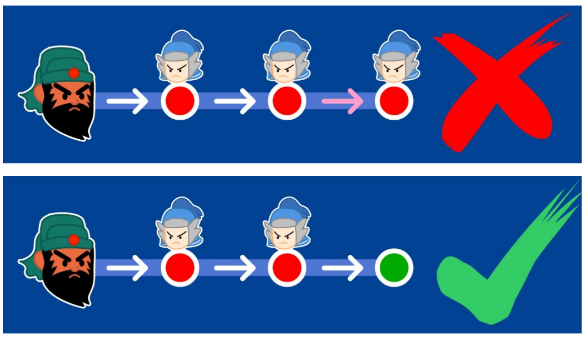
Planning an Escape

Planning an Escape



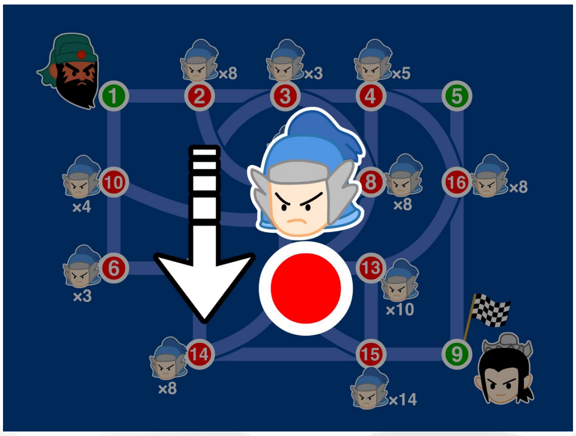
3

A Path with Resting Points



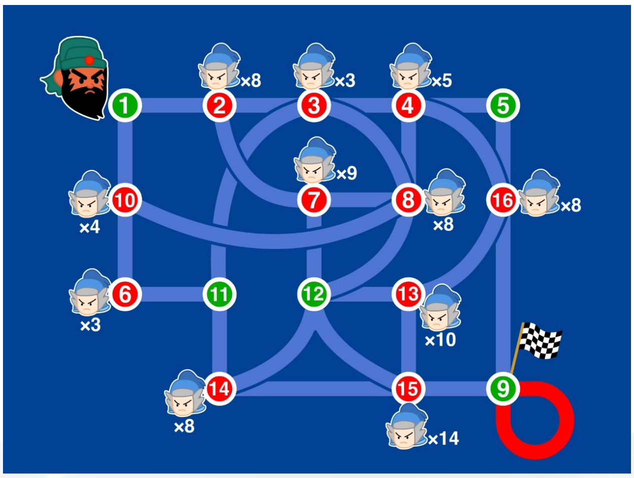
4

Objective



5

Data to Represent a Graph (escape.dzn)



6

Table Data for the Path (escape.dzn)

- Should use assert to check data!

```
n = 16;
guard = [0, 8, 3, 5, 0, 3, 9, 8, 0, 4, 0, 0, 10,
8, 14, 8];
m = 27;
edge = [| 1,2  | 1,10 | 2,3  | 2,7  | 3,4
        | 3,8  | 3,11 | 4,5  | 4,8  | 4,16
        | 5,16 | 6,10 | 6,11 | 7,8  | 7,12
        | 8,10 | 8,12 | 9,9  | 9,15 | 9,16
        | 11,14| 12,13| 12,14| 12,15| 13,15
        | 13,16| 14,15|];
rest = 3;
start = 1;
dest = 9;
maxstep = 10; % anticipated max number of steps
```

7

Planning an Escape (escape.mzn)

- Data

```
int: n; set of int: NODE = 1..n;
array[NODE] of int: guard;
int: m; set of int: EDGE = 1..m;
array[EDGE,1..2] of NODE: edge;
NODE: start;
NODE: dest;
int: rest; % resting every rest junctions
```

- Decisions

```
int: maxstep;
var int: step; % the actual number of steps
set of int: STEP = 1..maxstep;
array[STEP] of var NODE: path;
```

8

The Table Global Constraint

- A global constraint for representing abstract relations

```
table(array[1..n] of var int: x,
      array[1..T,1..n] of int: t)
```

- Enforces that the *x*'s take all take a value from one row of the table *t*
- E.g.
  - table([*x*,*y*,*z*], [| 0,0,0 | 1,2,3 | 4,2,0 |])
  - either $x = 0 \wedge y = 0 \wedge z = 0$
  - or $x = 1 \wedge y = 2 \wedge z = 3$
  - or $x = 4 \wedge y = 2 \wedge z = 0$

9

Representing a Path on a Graph (escape.mzn)

```
path[1] = start;
forall(i in 1..maxstep-1)
    (table([path[i],path[i+1]],edge]);
```

- The table constraint enforces that two consecutive path positions are connected by an edge
- **Why the self edge at the destination??**

10

A Path with Unknown Length (escape.mzn)

- Make an educated guess on the maximum possible length of the path
  - The smaller the better, but not too small
  - Too large a guess will make the problem harder

```
int: maxstep;
var int: step; % the actual number of steps
set of int: STEP = 1..maxstep;
array[STEP] of var NODE: path;
```

- Trick: **self edge** at the destination
  - Keep looping once the destination is reached
  - E.g. path = [1, 2, 3, 11, 14, 15, 9, 9, 9, 9]

```
forall(i in STEP)(i >= step -> path[i] = dest);
```

11

Sliding Sum

- The sliding sum global
  - `sliding_sum(int: low,`
  - `int: up,`
  - `int: seq,`
  - `array [int] of var int: vs)`
  - enforces that each subsequence vs[i] .. vs[i+seq-1] sums to between low and up
- It is used to enforce properties of sequences
- E.g. `sliding_sum(4, 8, 4, x)`

```
x = [1,4,2,0,0,3,4] ✅
x = [1,4,3,0,1,0,2] ❌
```

12

Resting Constraints and Objective (escape.mzn)

- The resting constraints
  - For our example, it requires in **every sequence** of "rest" number of nodes i in the path, at least one has guard[i] = 0

```
sliding_sum(1, rest, rest,
   [guard[path[i]] = 0 | i in STEP]);
```

- Objective

```
solve minimize sum(i in STEP)(guard[path[i]]);
```

13

Planning an Escape

- Running the model gives

```
=====UNSATISFIABLE=====
```

- **What do we do?**

14

Missing Solutions

- Possibly the worst things that can happen when running a model is
  - after longer than you want to wait
  - no solution is printed
- First Fix (for optimization problems)
  - Run with all solutions printed
    - -a, —all_solutions
  - Without this no solution is printed if its not proved optimal
  - This is the default behavior in the IDE
- And if still no solutions found!

15

Missing Solutions

- More general techniques
  - Adding a solution to the problem
  - Relaxing constraints

16

Missing Solutions: Add a Solution

- Construct a small instance that shows the problem
- Construct something that should be a solution
- Add it to the model, rerun!
- With luck you get a message like

```
Model inconsistency detected.
```

- And a line number indicating where!
- Maybe not :-(

```
warning: model inconsistency detected before
search.
```


Missing Solutions: Relaxing Constraints

- Then what?
- Try removing constraints one by one until a solution is found.
- Narrow down the responsible (set of) constraints
- Fix it!

Adding a Solution

- Add an expected solution to the model

```
path = [1,10,6,11,14,15,9,9,9,9];
step = 7;
```

- Unfortunately in this case no error message? Just

```
=====UNSATISFIABLE=====
```

- So lets start removing constraints
  - minus sliding sum

```
=====UNSATISFIABLE=====
```

  - minus table

```
----------
==========
```

19

Adding a Solution

- Seems like something to do with table?
- But what?
- Let's add a trace
  - note useless without the solution

```
forall(i in 1..maxstep-1)
        (trace("table([\(path[i]),\(path[i+1])],"
             ++ "edge)\n",
         table([path[i],path[i+1]],edge)));
```

20

THE UNIVERSITY OF MELBOURNE 香港中文大學 The Chinese University of Hong Kong

Adding a Solution

- Running the model

```
table([1,10],edge)
table([10,6],edge)
table([6,11],edge)
table([11,14],edge)
table([14,15],edge)
table([15,9],edge)
table([9,9],edge)
table([9,9],edge)
table([9,9],edge)
=====UNSATISFIABLE=====
```

- Now is it clear what is going on?
- We only have one direction of the edges

21

Correcting the Model (escape-fixed.mzn)

- We need to create an undirected form of the edges. Just add a reverse of each edge

```
array[1..2*m,1..2] of NODE: uedge =
   array2d(1..2*m,1..2,
      [ edge[i,j]   | i in EDGE, j in 1..2] ++
      [ edge[i,3-j] | i in EDGE, j in 1..2]);
```

- replace the use of `edge` with `uedge`

```
forall(i in 1..maxstep-1)
   (table([path[i],path[i+1]],uedge]);
```

22

No Solutions

- In the worst case
  - the model is too slow to solve!
  - so the solver just goes into a large search and never returns
- What can you do!?
  - simplify the model
    - take into account less of the problem
    - look at the solutions of the simplified problem
    - can you use these
  - decompose the model
    - break the problem into two parts
    - solve the first part
    - fix a solution to the first part to define the second part

23

Summary

- Debugging models is HARD
- Key methods
  - Find a small example which shows the problem
  - Make sure the constraints generated are the constraints that you think they are
  - Generate a solution you expect to find
  - Switch off and on constraints in the model
  - Solve a simpler version of the problem first
    - by ignoring constraints/parts of the problem
- Debugging is a vital skill to develop

24

THE UNIVERSITY OF MELBOURNE

香港中文大學
The Chinese University of Hong Kong

Image Credits

All graphics by Marti Wong, ©The Chinese University of Hong Kong and the University of Melbourne 2016