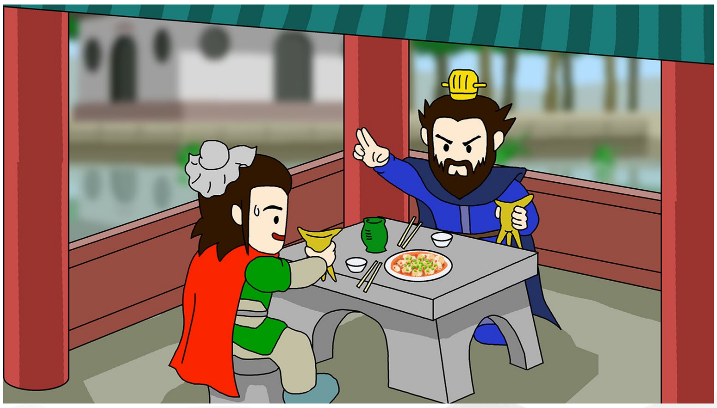
THE UNIVERSITY OF MELBOURNE 香港中文大學 The Chinese University of Hong Kong

Multiple Modeling

Jimmy Lee & Peter Stuckey

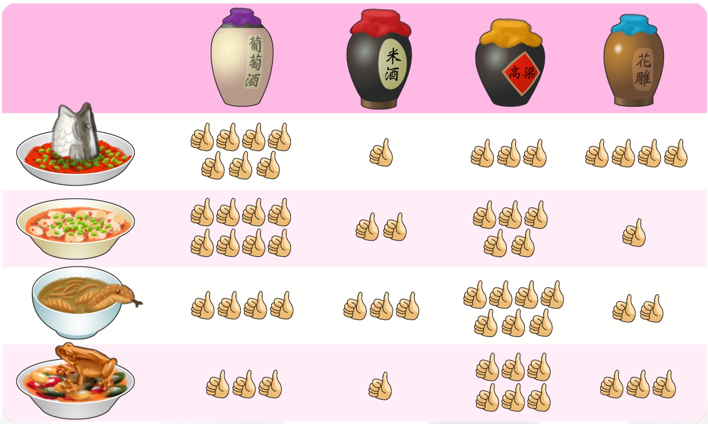
香港中文大學 The Chinese University of Hong Kong THE UNIVERSITY OF MELBOURNE

Judging Heroes Cooking Wine



2

The Joy of Food-Wine Pairing



3

Another Pure Assignment Problem

- To determine a function f: DOM ➔ COD again
  - DOM = food and COD = wine
- Constraint: Pair up each dish with a different drink
- Objective: maximize the joyfulness of the culinary (and political) occasion

4

The Cooking Wine Problem (foodToWine.mzn)

```
include "globals.mzn";

enum FOOD;
enum WINE;
array[FOOD, WINE] of int: joy;

array[FOOD] of var WINE: drink;

constraint alldifferent(drink);

solve maximize sum(f in FOOD)(joy[f, drink[f]]);
```

5

Solving the Food-Wine Model

```
Pair Food CHILIFISHHEAD with Wine HUADIAO
Pair Food MAPOTOFU with Wine GRAPE
Pair Food SNAKESOUP with Wine RICE
Pair Food GONGBAOFROG with Wine GAOLIANG


Joy: 21
----------
==========
```

6

Multiple Models

- Discrete optimization problems often have
  - multiple viewpoints on the same problem
- We can build two (or more) completely distinct models to solve the same problem
- We can also combine them

7

Viewpoints

- Function f: DOM ➔ COD is special when
  - |DOM| = |COD|
  - Function f is a bijection
- A viewpoint looks at the decisions of the problem from a specific angle
- This is a complete matching:
  - match each *d* in DOM with a different *c* in COD
  - or, equivalently, match each *c* in COD with a different *d* in DOM
  - two different **viewpoints** ==> two complementary models

8

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

Complete Matching

- A bijective function has two viewpoints
- The (usual) function

```
array[DOM] of var COD: f;
```

- And the inverse function

```
array[COD] of var DOM: finv;
```

- As much as we can pair drinks to food, we can also pair food to drinks
- In the Cooking Wine problem, the inverse function is

```
array[WINE] of var FOOD: eat;
```

9

The Cooking Wine Problem (wineToFood.mzn)

```
include "globals.mzn";

enum FOOD;
enum WINE;
array[FOOD, WINE] of int: joy;
```

```
array[WINE] of var FOOD: eat;

constraint alldifferent(eat);

solve maximize sum(w in WINE)(joy[eat[w], w]);
```

10

Solving the Inverse Wine-Food Model

```
Pair Food MAPOTOFU with Wine GRAPE
Pair Food SNAKESOUP with Wine RICE
Pair Food GONGBAOFROG with Wine GAOLIANG
Pair Food CHILIFISHHEAD with Wine HUADIAO


Joy: 21
----------
==========
```

11

Matching Food and Wine

- Which model is likely better?
  - Original

```
alldifferent(drink);
maximize sum(f in FOOD)(joy[f, drink[f]]);
```

  - Inverse

```
alldifferent(eat);
maximize sum(w in WINE)(joy[eat[w], w]);
```

12

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

Channeling Constraints

- Some constraints of the problem may be easier to express using the function, or its inverse
- Why not use both!?

- We can combine the two models
- We need to make the two functions agree by using channeling constraints

```
forall(w in WINE, f in FOOD)
      (eat[w] = f <->
       drink[f] = w);
```

13

Combining Models Using inverse

- This channeling constraint can also be captured by the global constraint
  - `inverse(eat, drink)`
  - or `inverse(drink, eat)`
  - Note we can **remove** the `alldifferent` constraints, made redundant by `inverse`

- Why would we combine models?

14

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

The Cooking Wine Problem (combined.mzn)

```
include "globals.mzn";

enum FOOD;
enum WINE;
array[FOOD, WINE] of int: joy;

array[FOOD] of var WINE: drink;
array[WINE] of var FOOD: eat;

constraint inverse(eat, drink);

solve maximize sum(f in FOOD)(joy[f, drink[f]]);
% solve maximize sum(w in WINE)(joy[eat[w], w]);
```

15

The Cooking Wine Problem (combined.mzn)

```
include "globals.mzn";

enum FOOD;
enum WINE;
array[FOOD, WINE] of int: joy;

array[FOOD] of var WINE: drink;
array[WINE] of var FOOD: eat;

constraint inverse(eat, drink);

solve maximize sum(f in FOOD)(joy[f, drink[f]]);
% solve maximize sum(w in WINE)(joy[eat[w], w]);
```

16

Why Combining Models?

- CP-based solvers can benefit from combining models to improve solving efficiency if done properly
- **Ease** of expression of constraints.
  - The Cooking Wine problem, as is, is a pure assignment. **NO!**
  - But what about side constraints?

17

Example Side Constraints

- Richness in taste

3 2 1 4

```
array[FOOD] of int: taste;
```

- The dish paired with GAOLIANG should be richer in taste than that paired with GRAPE wine

18

Example Side Constraints

- Richness in taste

3 2 1 4

```
array[FOOD] of int: taste;
```

- The dish paired with GAOLIANG should be richer in taste than that paired with GRAPE wine

```
taste[eat[GRAPE]] < taste[eat[GAOLIANG]];
```

- This constraint is difficult, if not impossible, to express in the food-wine model

19

Example Side Constraints

- Alcohol contents

1 3 4 2

```
array[WINE] of int: alcohol;
```

- The drink paired with GONGBAOFROG should be stronger in alcohol than that paired with SNAKESOUP

20

Example Side Constraints

- Alcohol contents

1 3 4 2

```
array[WINE] of int: alcohol;
```

- The drink paired with GONGBAOFROG should be stronger in alcohol than that paired with SNAKESOUP

```
alcohol[drink[SNAKESOUP]] <
    alcohol[drink[GONGBAOFROG]];
```

- This constraint is difficult, if not impossible, to express in the wine-food model

21

Example Side Constraints

- MAPOTOFU is paired with RICE wine if and only if SNAKESOUP is paired with HUADIAO wine

22

Example Side Constraints

- MAPOTOFU is paired with RICE wine if and only if SNAKESOUP is paired with HUADIAO wine

```
eat[RICE] = MAPOTOFU <->
   eat[HUADIAO] = SNAKESOUP;
```

OR

```
drink[MAPOTOFU] = RICE <->
   drink[SNAKESOUP] = HUADIAO;
```

OR

```
eat[RICE] = MAPOTOFU <->
   drink[SNAKESOUP] = HUADIAO;
```

OR

```
drink[MAPOTOFU] = RICE <->
   eat[HUADIAO] = SNAKESOUP;
```

23

Summary

- Multiple viewpoints of the problem leads to
  - multiple models
- Different viewpoints can express different constraints
  - more naturally and succinctly
  - better for the solvers (usually succinct is better)
- Channeling constraints make the viewpoints agree and unite them
- Combining the models can sometimes improve solving efficiency on either single model

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