Modeling is Magic

Peter Stuckey

Overview

- What is all this stuff about talking to dragons!
- Unpacking the metaphors
- Some first simple “spells”
- Why you should care!

Metaphors and Modeling

- “Dragons”
  - dragons are discrete optimization solvers
  - deep complex software
  - 10s-100s of PhD research behind them
- “Dragonspeech”
  - MiniZinc is a language for expression models of discrete optimization problems
- “True Names”
  - names of frequent combinatorial substructures

Dragons

- Powerful solving technology
  - has increased in speed orders of magnitude faster than the increase in CPU speed
- “Dragons”
  - CPLEX: commercial MIP solver (IBM)
  - Gurobi: commercial MIP solver
  - Gecode: open source CP solver
  - OR-tools: open source CP solver (Google)
  - Oscar: local search solver
  - many other solvers support MiniZinc

Modeling is Magic

- Sudoku solving
- Fill in the 9x9 grid with numbers 1 to 9 such that each row, and columns and sub square contains each of 1..9 exactly once

	6	8	4		1		7	
				8	5		3	
	2	6	8		9		4	
		7				9		
	5		1		6	3	2	
	4		6	1				
	3		2		7	6	9	

Modeling is Magic

- A MiniZinc model for this

```
include "alldifferent.mzn";
int: S;
int: N = S * S;
array[1..N,1..N] of     0..N: start; %% 0 = empty
array[1..N,1..N] of var 1..N: puzzle;

constraint forall(i,j in 1..N)
           (if start[i,j] > 0 then
               puzzle[i,j] = start[i,j]
            else true endif );
constraint forall(i in 1..N)
           ( alldifferent( [ puzzle[i,j] | j in 1..N ]) );
constraint forall(j in 1..N)
           ( alldifferent( [ puzzle[i,j] | i in 1..N ]) );
constraint forall(a, o in 1..S)
           ( alldifferent( [ puzzle[(a-1)*S+a1, (o-1)*S+o1]
                           | a1, o1 in 1..S ] ) );
solve satisfy;
```


Modeling is Magic

- Sudoku solving
- The unique solution in milliseconds!

5	9	3	7	6	2	8	1	4
2	6	8	4	3	1	5	7	9
7	1	4	9	8	5	2	3	6
3	2	6	8	5	9	1	4	7
1	8	7	3	2	4	9	6	5
4	5	9	1	7	6	3	2	8
9	4	2	6	1	8	7	5	2
8	3	5	2	4	7	6	9	1
6	7	1	5	9	3	4	8	2

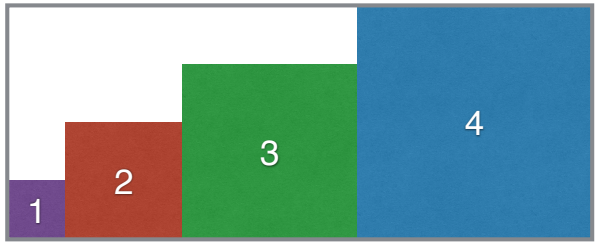
Beyond Puzzles

- Complex packing problems are an example of real world discrete optimization problems

- Carpet cutting
  - cut room shapes out of a carpet to minimize the amount of the roll required
- The client pays for the square area of carpet
- You pay for the length of carpet cut from the roll
- Better packing = better profits

Squares Packing

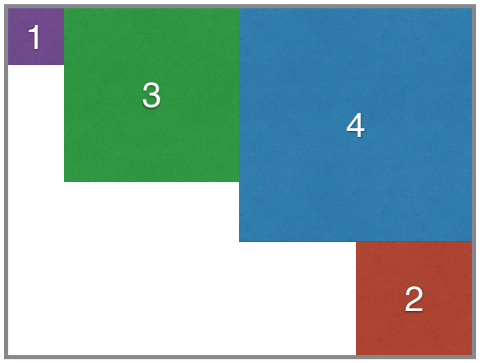
- Pack squares of size 1x1, 2x2, …, *n*x*n* into a rectangle of smallest total area



height = 4, width = 10, area = 40

Squares Packing

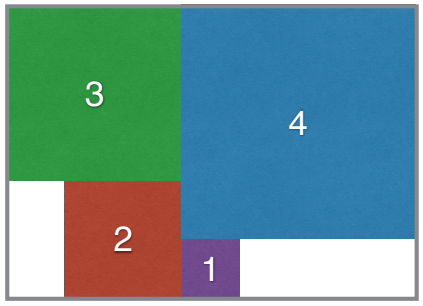
- Pack squares of size 1x1, 2x2, …, *n*x*n* into a rectangle of smallest total area



height = 6, width = 8, area = 48

Squares Packing

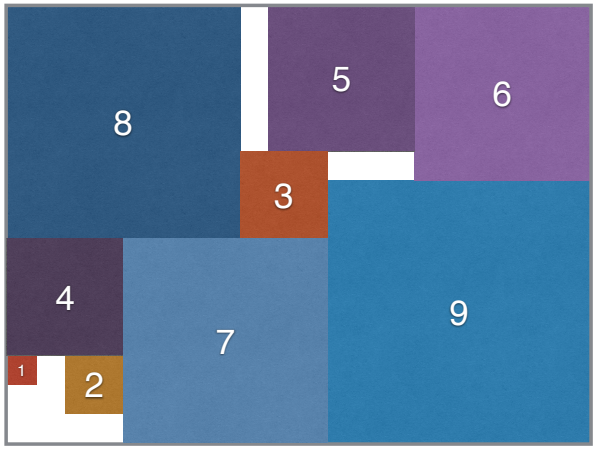
- Pack squares of size 1x1, 2x2, …, *n*x*n* into a rectangle of smallest total area



height = 5, width = 7, area = 35

Square Packing

- 1..9 in 15x20 = 300



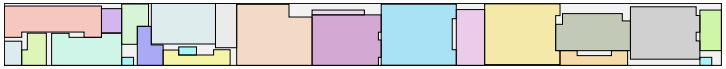
Square Packing Model

- MiniZinc model

```
int: n; % number of squares
set of int: SQUARE = 1..n;
int: maxl = sum(i in SQUARE)(i);
var n..maxl: height;
var n..maxl: width;
var 0..n*maxl: area = height * width;
array[SQUARE] of var 0..maxl: x;
array[SQUARE] of var 0..maxl: y;
% squares fit in the rectangle
constraint forall(s in SQUARE)(x[s] + s <= width);
constraint forall(s in SQUARE)(y[s] + s <= height);
% non overlap with global diffn
array[SQUARE] of int: size = [ i | i in SQUARE ];
include "diffn.mzn";
constraint diffn(x,y,size,size);
solve minimize area;
```


Carpet Cutting

- Complex packing problem
- Room shapes are complex
- Stair carpets have many cutting options
- Pattern of carpet is important



- Example layout for 30+ meters of carpet
- MiniZinc + powerful solver solution
  - first approach to prove optimal solutions
  - faster than the existing approach
  - reduced waste by 35%

True Names

- Combinatorial substructures occur in many variations of problems
- If we can “name” combinatorial substructures in our models
  - the model is more concise
  - the solvers can solve them more effectively
- Examples
  - `alldifferent`: assignment subproblem
  - `diffn`: 2D packing subproblem

What is Modeling

- Specify a set of decisions that need to be made
  - variables
- Specify a series of bi-directional relations
  - constraints on the decisions
- Specify a goal
  - the objective (ranking solutions)
- A model is
  - a specification of the problem
  - not a recipe on how to solve a problem

EOF

Why should we model?

Peter Stuckey

Satisfying Constraints

- The real world is full of constraints
  - physics
  - resource limits
  - work regulations
- The real world is full of decisions problems
  - determine the staff roster
  - determine a fair allocation of water
  - assign crews to airplanes
- To solve these problems we must
  - satisfy constraints

Constraint Example

- Imagine we wish to find a solution to
  - $x = -y^2 + 5, y = 2x^3, x \geq 0, y \geq 0$
- How do we do this
- Idea 1: guess and test
  - guess x = 0, y = 0 ✖
  - guess x = 1, y = 0 ✖
  - guess x = 5, y = 0 ✖
  - guess x = 3, y = 2 ✖

Constraint Example

- Imagine we wish to find a solution to
  - $x = -y^2 + 5, y = 2x^3, x \geq 0, y \geq 0$
- How do we do this
- Idea 2: iterative refinement
  - guess x = 0
  - calculate $y = 2 \times 0^3 = 0$ ✖
  - calculate $x = -0^2 + 5 = 5$ ✖
  - calculate $y = 2 \times 5^3 = 1250$ ✖
  - calculate $x = -1250^2 + 5 = 1562495$ ✖

Constraint Example

- Imagine we wish to find a solution to
  - $x = -y^2 + 5, y = 2x^3, x \geq 0, y \geq 0$
- Did you find the solution
  - $x = 1, y = 2$
- How did you find it?

Solving Constraint Problems

- Constraint problems are
  - much easier to state than to solve
- Decades of research has gone into
  - building efficient solving algorithms
- If we can communicate with these solving algorithms
  - we can solve very challenging problems

The Holy Grail of Programming

The user states the problem

The computer solves it

- "*Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.*"

[Freuder, E. C. (1997). In Pursuit of the Holy Grail. Constraints, 2(1), 57-61]

No Free Lunch Theorem

- "*state[s] that any two optimization algorithms are equivalent when their performance is averaged across all possible problems.*"
  [Wolpert, D.H., & Macready, W.G. (1997). No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation, 1, 67]

- But this theoretical result relies on problems that are not interesting, or even expressible on modern computers

- In short, we are not interested in "all possible problems"

BUT

- While we will teach you to state discrete optimization problems relatively easily
  - "with great power comes great responsibility"

- Most discrete optimization problems are
  - NP hard
- Which means we cant expect to solve instances as the size grows
- Beware: many models wont return answers
  - (a) the problem is very hard, and/or
  - (b) the model is very bad

Overview

- Why should we model?

- Solving constraint problems is hard
  - but there are technologies to do it

- Stating constraint problems is much easier
  - modeling is stating the problem formally

- Solver independent modeling allows us to
  - not commit to a solving technology
  - experiment with and compare technologies

EOF

What is MiniZinc

Peter Stuckey

Overview

- What is MiniZinc
- The MiniZinc toolchain
  - mzn2fzn
  - solns2out

What is MiniZinc

- MiniZinc is a modeling language
  - not a programming language
- It is designed to specify
  - variables,
  - constraints, and
  - objectives
- of discrete optimization problems
- MiniZinc is independent of the underlying solver

What is MiniZinc

- MiniZinc is a model compiler
- It translates a user model and data to
  - variables
  - constraints, and
  - objective
- for an underlying solver
- The translation is different for every solver.

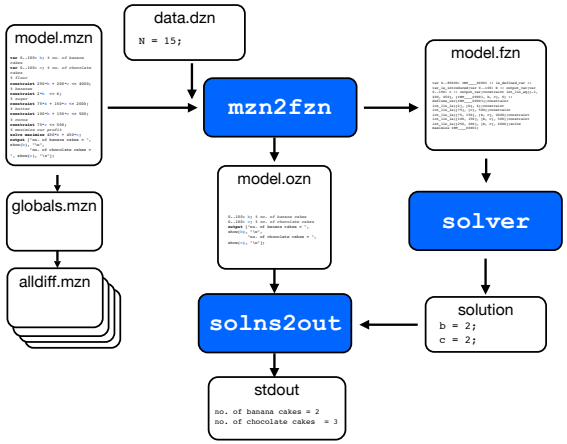
What is MiniZinc

- MiniZinc is a problem solver
- After translating the problem it
  - calls an underlying solver, and
  - converts the solvers answer to an output
- MiniZinc can also be used as an rich API to interact with multiple solvers

How does MiniZinc work

- MiniZinc Components
  - mzn2fzn: model translation
  - solns2out: answer translation
- Solving Components
  - solver: solves the problem
  - solver library: defines how constraints are treated by the solver
- Model Components
  - model
  - data

The MiniZinc Tool Chain



mzn2fzn

- MiniZinc to FlatZinc
  - Model + Data are combined
  - Global constraint definitions are included
    - from standard library
    - or solver specific library
  - Model is flattened to consist of
    - variable declarations
    - primitive constraints (a conjunction of constraints)
    - solve item
    - FlatZinc

solver

- Executes FlatZinc model
- Outputs the solutions
  - just in terms of values of variables of interest

```
var 0..5: x; var 1..10: y; var 3..7: z;
constraint x + y >= z
output [ show(x+y) ];
```

  - leads to FlatZinc

```
var 0..5: x  :: output_var;
var 1..10: y :: output_var;
var 3..7: z;
constraint int_lin_le([-1,-1,1],[x,y,z],0);
```

  - prints solution on output_vars

```
x = 2; y = 1;
```


solns2out

- Responsible for handling output items
- .ozn file
  - minimal MiniZinc to specify the output
- solns2out
  - Reads the solution from the solver as .dzn
  - Combines with the .ozn file
  - Prints output
- Example continued (ozn)

```
output [show(x+y)];
int: x;
int: y;
```

- Result “3”

Remainder of the course

- We examine how to use the MiniZinc language to
  - capture concise efficient models
  - and use underlying solvers
  - to solve complex discrete optimization problems
- MiniZinc
  - language features
  - combinatorial substructures
  - debugging and improving models

EOF

Why MiniZinc

Peter Stuckey

What is a Solver?

- A solver (i.e. a dragon) is an algorithm
  - input: declarative model
    - decision variables, constraints, objective function
  - output: the best solution (some times)
- Highly optimized codes
  - built on hundreds of PhD thesis
- Many different types
  - each with their own pros / cons

There are Lots of Solvers

- Constraint Programming (CP)
  - gecode, or-tools, cpx, choco, …
- Linear Programming (LP)
  - cpl, glpk, lp_solve, …
- Mixed Integer Programming (MIP)
  - cplex, gurobi, …
- NonLinear Programming (NLP)
  - ipopt, dfo, filterSD, …
- Mixed Integer NonLinear Programming (MINLP)
  - scip, couenne, …
- …

The Challenges of Using Solvers

- They all speak different languages
- Sometimes this is easy to fix
  - e.g. renaming keywords
  - "alldiff" becomes "alldifferent"
- Other times it is no obvious
  - Constraint Programming Solver - alldifferent

```
alldifferent(x);
```

  - Integer Programming Solver - alldifferent

```
forall(i in Items, v in Values)
      (x[i,v] = 1);
forall(i,j in Items, v in Values where i < j)
      (x[i,v] + x[j,v] <= 1);
```

The Challenges of Using Solvers

- Different Solvers have different strengths and weaknesses
  - Constraint Programming for Scheduling
  - Mixed Integer Programming for Network Design
- Need to rapidly switch between solvers

Why MiniZinc?

MiniZinc is…

One language to rule them all!

And through a model bind them.

- Learning MiniZinc unlocks the power of **all** the solvers.
- Bonus
  - free, open source
  - aids rapid integration / deployment in your software

EOF