DM841

Discrete Optimization

Part I

**Lecture 5**

# More Examples

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

# Outline

1. Examples
   n-Queens, Grocery, Magic Square

# Outline

# Outline

1. Examples
   n-Queens, Grocery, Magic Square

# Constraints: No Attack

- not in same column
  - by choice of variables
- not in same row
  - $x_i \neq x_j$ for $i \neq j$
- not in same diagonal
  - $x_i - i \neq x_j - j$ for $i \neq j$
  - $x_i - j \neq x_j - i$ for $i \neq j$
- $3 \cdot n \cdot (n - 1)$ constraints

# Fewer Constraints…

- Sufficient by symmetry

  $i < j$ instead of $i \neq j$
- Constraints
  - $x_i \neq x_j$ for $i < j$
  - $x_i - i \neq x_j - j$ for $i < j$
  - $x_i - j \neq x_j - i$ for $i < j$
- $3/2 \cdot n \cdot (n - 1)$ constraints

# Even Fewer Constraints

- Not same row constraint

  $x_i \neq x_j$ for $i < j$

  means: values for variables pairwise distinct
- Constraints
  - distinct($x_0$, …, $x_7$)
  - $x_i - i \neq x_j - j$ for $i < j$
  - $x_i - j \neq x_j - i$ for $i < j$

# Pushing it Further…

- Yes, also diagonal constraints can be captured by distinct constraints
  - ~~see assignment~~

```
distinct(x0,   x1,   ...,  x7)
distinct(x0-0, x1-1, ..., x7-7)
distinct(x0+0, x1+1, ..., x7+7)
```

# Script: Variables

```
Queens(void) : q(*this,8,0,7) {
  …
}
```

# Script: Constraints

```
Queens(void) : q(*this,8,0,7) {
  distinct(*this, q);
  for (int i=0; i<8; i++)
    for (int j=i+1; j<8; j++) {
  rel
      post(*this, x[i]-i != x[j]-j);
      post(*this, x[i]-j != x[j]-i);
    }
  …
}
```

# Script: Branching

```
Queens(void) : q(*this,8,0,7) {
  …
  branch(*this, q,
         INT_VAR_NONE,
         INT_VAL_MIN);
}
```

# Good Branching?

- Naïve is not a good strategy for branching

- Try the following (see assignment)
  - first fail
  - place queen as much in the middle of a row
  - place queen in knight move fashion

# Summary 8 Queens

- Variables
  - model should require few variables
  - good: already impose constraints
- Constraints
  - do not post same constraint twice
  - try to find “big” constraints subsuming many small constraints
    - more efficient
    - often, more propagation (to be discussed)

# Grocery

# Grocery

- Kid goes to store and buys four items
- Cashier: that makes $7.11
- Kid: pays, about to leave store
- Cashier: hold on, I multiplied!
  let me add!
  wow, sum is also $7.11
- You: prices of the four items?

# Model

- Variables
  - for each item A, B, C, D
  - take values between {0, …, 711}
  - compute with cents: allows integers
- Constraints
  - A + B + C + D = 711
  - A * B * C * D = 711 * 100 * 100 * 100

The unique solution (upon the symmetry breaking of slide 87) is:
A=120, B=125, C=150, D=316.

# Script

```
class Grocery : public Space {
protected:
    IntVarArray abcd;

    const int s = 711;
    const int p = s * 100 * 100 * 100;
public:
    Grocery(void) … { … }

    …
}
```

# Script: Variables

```
Grocery(void) : abcd(*this,4,0,711) {
  …
}
```

# Script: Sum

```
…
// Sum of all variables is s
linear(this, abcd, IRT_EQ, s);

IntVar a(abcd[0]), b(abcd[1]),
       c(abcd[2]), d(abcd[3]);
```

# Script: Product

```
IntVar t1(*this,1,p);
IntVar t2(*this,1,p);
IntVar t3(*this,p,p);

mult(*this, a, b, t1);
mult(*this, c, d, t2);
mult(*this, t1, t2, t3);
```

# Branching

- Bad idea: try values one by one
- Good idea: split variables
  - for variable $x$
  - with $m = (\min(x) + \max(x)) / 2$
  - branch $x < m$ or $x \geq m$
- Typically good for problems involving arithmetic constraints
  - exact reason needs to be explained later

# Script: Branching

```
branch(*this, abcd,
       INT_VAR_NONE,
       INT_VAL_SPLIT_MIN);
```

# Search Tree

- 2829 nodes for first solution
- Pretty bad…

# Better Heuristic?

- Try branches in different order
  split with larger interval first
  - try: `INT_VAL_SPLIT_MAX`
- Search tree: 2999 nodes
  - worse in this case

# Symmetries

- Interested in values for A, B, C, D
- Model admits equivalent solutions
  - interchange values for A, B, C, D
- We can add order A, B, C, D:

  $A \le B \le C \le D$
- Called “symmetry breaking constraint”

# Script: Symmetry Breaking

```
…
rel(this, a, IRT_LQ, b);
rel(this, b, IRT_LQ, c);
rel(this, c, IRT_LQ, d);
…
```

# Effect of Symmetry Breaking

- Search tree size 308 nodes
- Let us try `INT_VAL_SPLIT_MAX` again
  - tree size 79 nodes!
  - interaction between branching and symmetry breaking
  - other possibility: $A \geq B \geq C \geq D$
  - we need to investigate more (later)!

# Any More Symmetries?

- Observe: 711 has prime factor 79
  - that is: $711 = 79 \times 9$
- Assume: A can be divided by 79
  - add: $A = 79 \times X$
    for some finite domain var X
  - remove $A \leq B$
  - the remaining B, C, D of course can still be ordered

# Any More Symmetries?

- In Gecode

```
IntVar x(*this,1,p);
IntVar sn(*this,79,79);
mult(*this, x, sn, a);
```

- Search tree 44 nodes!
  - now we are talking!

# Summary: Grocery

- Branching: consider also
  - how to partition domain
  - in which order to try alternatives
- Symmetry breaking
  - can reduce search space
  - might interact with branching
  - typical: order variables in solutions
- Try to really understand problem!

# Domination Constraints

- In symmetry breaking, prune solutions without interest
- Similarly for best solution search
  - typically, interested in just one best solution
  - impose constraints to prune some solutions with same "cost"

# Another Observation

- Multiplication decomposed as

  $$A \cdot B = T_1 \quad C \cdot D = T_2 \quad T_1 \cdot T_2 = P$$

- What if

  $$A \cdot B = T_1 \quad T_1 \cdot C = T_2 \quad T_2 \cdot D = P$$

  - propagation changes: 355 nodes
  - propagation is not compositional!
  - another point to investigate

# Magic Squares

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

Unique solution for n=3, upon the symmetry breaking of slide 99.

# Magic Squares

- Find an $n \times n$ matrix such that
  - every field is integer between 1 and $n^2$
  - fields pairwise distinct
  - sums of rows, columns, two main diagonals are equal

- Very hard problem for large $n$
- Here: we just consider the case $n=3$

# Model

- For each matrix field have variable $x_{ij}$
  - $x_{ij} \in \{1, .., 9\}$
- One additional variable s for sum
  - $s \in \{1, .., 9 \times 9\}$
- All fields pairwise distinct
  - distinct($x_{ij}$)
- For each row i have constraint
  - $x_{i0} + x_{i1} + x_{i2} = s$
  - columns and diagonals similar

# Script

- Straightforward
- Branching strategy
  - first-fail
  - split again: arithmetic constraints
  - try to come up with something that is really good!

- Generalize it to arbitrary *n*

# Symmetries

- Clearly, we can require for first row that first and last variable must be in order
- Also, for opposing corners
- In all (other combinations possible)
  - $x_{00} < x_{02}$
  - $x_{02} < x_{20}$
  - $x_{00} < x_{22}$

# Important Observation

- We know the sum of all fields

  $1 + 2 + \ldots + 9 = 9(9+1)/2=45$

- We "know" the sum of one row

  $s$

- We know that we have three rows

  $3 \times s = 45$

# Implied Constraints

- The constraint model already implies

  $3 \times s = 45$

  - implies solutions are the same
- However, adding a propagator for the constraint drastically improves propagation
- Often also: redundant or implied constraint

# Effect

| | |
|---|---|
| Simple model | 92 nodes |
| Symmetry breaking | 29 nodes |
| Implied constraint | 6 nodes |

# Summary: Magic Squares

- Add implied constraints
  - are implied by model
  - increase constraint propagation
  - reduce search space
  - require problem understanding
- Also as usual
  - break symmetries
  - choose appropriate branching

# Outlook…

- Common modeling principles
  - what are the variables
  - finding the constraints
  - finding the propagators
  - implied (redundant) constraints
  - finding the branching
  - symmetry breaking

# Modeling Strategy

- Understand problem
  - identify variables
  - identify constraints
  - identify optimality criterion
- Attempt initial model simple?
  - try on examples to assess correctness
- Improve model much harder!
  - scale up to real problem size

# Summary



Experiment with:

- different branching strategies
- different models (eg, adding redundat constraints)
- different propagation strength (to come)