ID2204: Constraint Programming

# Getting Practical… Modeling Introduction

Lecture 02, 2010-03-25

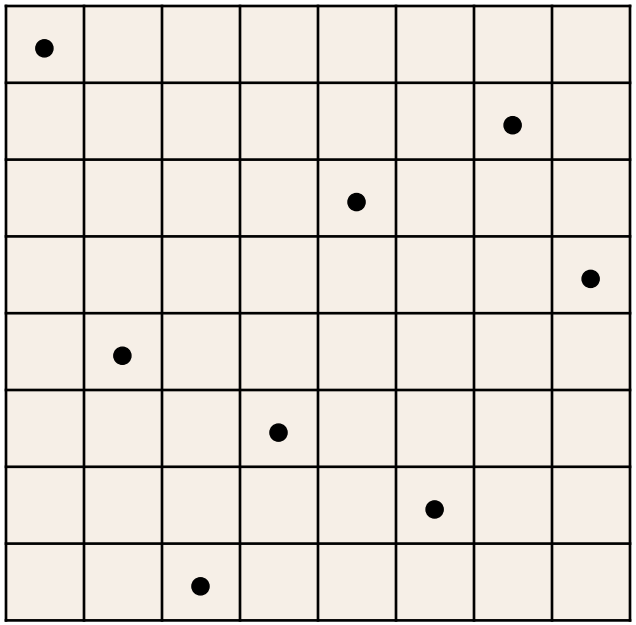
Christian Schulte
cschulte@kth.se

Software and Computer Systems
School of Information and Communication Technology
KTH – Royal Institute of Technology
Stockholm, Sweden


KTH Information and Communication Technology

# 8-Queens

# Problem Statement



- Place 8 queens on a chess board such that the queens do not attack each other
- Straightforward generalizations
  - place an arbitrary number: *n* Queens
  - place as closely together as possible

# What Are the Variables?

- Representation of position on board
- First idea: two variables per queen
  - one for row
  - one for column
  - $2 \cdot n$ variables
- Insight: on each column there will be a queen!

# Fewer Variables…

- Have a variable for each column
  - value describes row for queen
  - *n* variables

- Variables: $x_0, \ldots, x_7$

  where $x_i \in \{0, \ldots, 7\}$

# Other Possibilities

- For each field: number of queen
  - which queen is not interesting, so…
  - $n^2$ variables
- For each field on board: is there a queen on the field?
  - 8×8 variables
  - variable has value 0: no queen
  - variable has value 1: queen
  - $n^2$ variables

# 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

# 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++) {
      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)