G53CLP
Constraint Logic Programming

Dr Rong Qu

Issues in Modeling CSPs
How to Model a CSP

- Crucial part of constraint programming
  - Some models may be more efficient than the others
  - Iterative process in practice

- Experts in practice may add redundant constraints to the model
  - Do not change the problem
  - Reduce the number of solutions to be searched
How to Model a CSP

- Understand the problem

- Identify the variables
  - Represent entities in the problem
  - Identify domains of the variables

- Model the relationships among variables into constraints

- Test the model in a solver (language) chosen
How to Model a CSP

- Improve the models
  - The efficiency depends on the variables, domains and constraints
  - Reduce symmetry
  - Iterative process

Need to understand the underlying search strategy of the CP solver
The n-Queen Problem – symmetry

- For some problems, there are equivalent solutions
  - i.e. exchange values of variables return the same solutions
  - Transfer the solution in some way
  - n-queen problems, map coloring, ...
The n-Queen Problem – symmetry

- 8-queen problem
  - 92 solutions
  - only 12 solutions are different (unique)

http://en.wikipedia.org/wiki/Eight_queens_puzzle
The n-Queen Problem – symmetry

- This causes problems in searching
  - If one branch of the tree leads to no solution, then all branches of symmetric assignments also lead to dead-end
  - All branches will be explored, leading to maybe the same (symmetric) solutions
  - Great lost of time

- Symmetries should be avoided if possible
The n-Queen Problem – symmetry

- Re-formulation is the most effective way
  - No general technique - problem specific
  - add constraints before search

- Other approaches
  - adapt search algorithm to break symmetry
  - build search tree so that no symmetry arises
The n-Queen Problem – symmetry

- Break the symmetry*
  - \( x(r[i] = j) r[n-i+1] \neq j \) //vertically
  - \( y(r[i] = j) r[i] \neq n-j+1 \) //horizontally
  - \( d1(r[i] = j) r[j] \neq i \) //diagonally
  - \( d2(r[i] = j) r[n-j+1] \neq n-i+1 \)
  - \( r90(r[i] = j) r[j] \neq n-i+1 \) // turn the board
  - \( r180(r[i] = j) r[n-i+1] \neq n-j+1 \)
  - \( r270(r[i] = j) r[n-j+1] \neq i \)

*Gant & Smith, (2000)
Reuse of Models

Timetabling Problems

- A set of courses or exams taken by students
- A set of time slots in the timetable

- Allocate a time to each course or exam taken by students

- Avoid
  - A student sitting two courses or exams at the same time
Reuse of Models

- **Variables**
  - Courses/exams, \{x_1...x_i\}

- **Domain**
  - \{1, ..., t\}, t: number of timeslots

- **Constraints**
  - Conflicted courses/exams (with common students) can’t be assigned the same timeslot
Reuse of Models

Timetabling vs. Graph coloring

Nodes: course/exams
Edges: clashes (common students in adjacent nodes)

Assign colors to nodes – assign timeslots to courses or exams of enrolled students
enum Country {Belgium, Denmark, France, Germany, Netherlands, Luxembourg};
enum Colors {blue, red, yellow, gray};
var Colors color[Country];
solve {
    color[France] <> color[Belgium];
    color[France] <> color[Luxembourg];
    color[France] <> color[Germany];
    color[Luxembourg] <> color[Germany];
    color[Luxembourg] <> color[Belgium];
    color[Belgium] <> color[Netherlands];
    color[Belgium] <> color[Germany];
    color[Germany] <> color[Netherlands];
    color[Germany] <> color[Denmark];
};

Solution [1]
color[Belgium] = blue
color[Denmark] = blue
color[France] = red
color[Germany] = yellow
color[Netherlands] = red
color[Luxembourg] = gray
enum Modules {Math, Program, Algorithm, AI, Software, Database};
enum Slots {t1, t2, t3, t4};
var Slots ttable[Modules];
solve {
    ttable[Algorithm] ↔ ttable[Math];
    ttable[Algorithm] ↔ ttable[Database];
    ttable[Algorithm] ↔ ttable[AI];
    ttable[Database] ↔ ttable[AI];
    ttable[Database] ↔ ttable[Math];
    ttable[Math] ↔ ttable[Software];
    ttable[Math] ↔ ttable[AI];
    ttable[AI] ↔ ttable[Software];
    ttable[AI] ↔ ttable[Program];
};

Solution [1]
ttable[Math] = t1
ttable[Program] = t1
ttable[Algorithm] = t2
ttable[AI] = t3
ttable[Software] = t2
ttable[Database] = t4