Modelling for Constraint Programming

Barbara Smith

1. Definitions, Viewpoints, Constraints

Cork
Constraint
Computation
Centre

Background Assumptions

- A well-defined problem that can be represented as a finite domain constraint satisfaction or optimization problem
  - no uncertainty, preferences, etc.
- A constraint solver providing:
  - a systematic search algorithm
  - combined with constraint propagation
  - a set of pre-defined constraints
  - e.g. ILOG Solver, Ecl^{i}ps^{e}, SICStus Prolog, …

Solving CSPs

- Systematic *search:*
  - choose a variable x_i that is not yet assigned
  - create a *choice point*, i.e. a set of mutually exclusive & exhaustive choices, e.g. $x_i = a$ *v.* $x_i \neq a$
  - try the first & backtrack to try the other if this fails
- Constraint propagation:
  - add $x_i = a$ or $x_i \neq a$ to the set of constraints
  - re-establish local consistency on each constraint
    - $\rightarrow$ remove values from the domains of future variables that can no longer be used because of this choice
  - fail if any future variable has no values left

Representing a Problem

- If a CSP $M = <X,D,C>$ represents a problem P, then every solution of M corresponds to a solution of P and every solution of P can be derived from at least one solution of M
- More than one solution of M can represent the same solution of P, if modelling introduces symmetry
- The variables and values of M represent entities in P
- The constraints of M ensure the correspondence between solutions
- The aim is to find a model M that can be solved as quickly as possible
  - NB shortest run-time might not mean least search

Interactions with Search Strategy

- Whether *M1* is better than *M2* can depend on the search algorithm and search heuristics
- I'm assuming the search algorithm is fixed
- We could also assume that choice points are always $x_i = a$ v. $x_i \neq a$
- Variable (and value) order still interact with the model a lot
- Is variable & value ordering part of modelling?
  - I think it is, in practice
  - but here I will pretend it isn't

Viewpoints

- A viewpoint is a pair *<X,D>,* i.e. a set of variables and their domains
- Given a viewpoint, the constraints have to restrict the solutions of *M* to solutions of *P*
  - So the constraints are (to some extent) decided by the viewpoint
  - Different viewpoints give very different models
- We can combine viewpoints - more later
- Good rule of thumb: choose a viewpoint that allows the constraints to be expressed easily and concisely
  - will propagate well, so problem can be solved efficiently

Example: Magic Square

- Arrange the numbers 1 to 9 in a 3 x 3 square so that each row, column and diagonal has the same sum
- *V1* : a variable for each cell, domain is the numbers that can go in the cell
- *V2* : a variable for each number, domain is the cells where that number can go
- Constraints on row, column & diagonal sums are easy to express in *V1:*
  - $x_1+x_2+x_3 = x_4+x_5+x_6 = x_1+x_4+x_7 = \ldots$
- but not in *V2*

4	3	8
9	5	1
2	7	6

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

Constraints

- Given a viewpoint, the role of the constraints is:
- To ensure that the solutions of the CSP match the solutions of the problem
- To guide the search, i.e. to ensure that as far as possible, partial solutions that will not lead to a solution fail immediately

Expressing the Constraints

- For efficient solving, we need to know:
  - the constraints provided by the constraint solver
  - the level of consistency enforced on each
  - the complexity of the constraint propagation algorithms
  - Not very declarative!
- There is often a trade-off between time spent on propagation and time saved on search
  - which choice is best often depends on the problem

Auxiliary Variables

- Often, the constraints can be expressed more easily/more efficiently if more variables are introduced
- Example: car sequencing (Dincbas, Simonis and van Hentenryck, ECAI 1988)

Car Sequencing Problem

- 10 cars to be made on a production line, each requires some options
- Stations installing options have lower capacity than rest of line e.g. at most 1 car out of 2 for option 1
- Find a feasible production sequence

classes	1	2	3	4	5	6	Capacity
Option 1	1	0	0	0	1	1	1/2
Option 2	0	0	1	1	0	1	2/3
Option 3	1	0	0	0	1	0	1/3
Option 4	1	1	0	1	0	0	2/5
Option 5	0	0	1	0	0	0	1/5
No. of cars	1	1	2	2	2	2	

Car Sequencing - Model

- Variables : s_1, s_2, …, s_{10}
- Value of s_i is the class of car in position i in the sequence
- Constraints:
  - Each class occurs the correct number of times
  - Option capacities are respected - ?

Car Sequencing – Auxiliary Variables

- Introduce variables o_{ij} :
  - $o_{ij} = 1$ iff the car in the ith slot in the sequence requires option j
- Option 1 capacity is one car in every two:
  - $o_{i,1} + o_{i+1,1} \leq 1$ for $1 \leq i < 10$
- Relate the auxiliary variables to the s_i variables:
  - $\lambda_{jk} = 1$ if car class k requires option j
  - $o_{ij} = \lambda_{js_i}$, $1 \leq i \leq 10$, $1 \leq j \leq 5$

Global Constraints

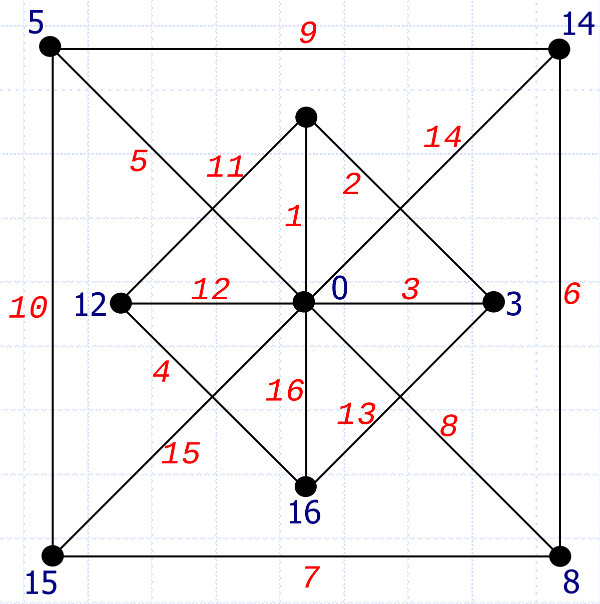
- A range of global constraints is provided by any constraint solver
- A global constraint replaces a set of simpler constraints on a number of variables
- The solver provides an efficient propagation algorithm (often enforcing GAC, sometimes less)
- A global constraint *should* reduce search, *may* reduce run-time (or may increase it)

The AllDifferent Constraint

- Commonest global constraint?
- allDifferent($x_1, x_2, \ldots, x_n$) replaces the binary $\neq$ constraints $x_i \neq x_j$, $i \neq j$
- There are efficient GAC & BC algorithms for allDifferent
  - i.e. more efficient than GAC on a general *n*-ary constraint
- Usually, using allDifferent gives less search than $\neq$ constraints
  - but is often slower
- Advice:
  - use allDifferent when the constraint is tight
  - i.e. the number of possible values is *n* or not much more
  - try BC rather than GAC

Graceful Labelling of a Graph

- A labelling f of the nodes of a graph with q edges is graceful if:
  - f assigns each node a unique label from $\{0,1,\dots, q\}$
  - when each edge xy is labelled with $|f(x) - f(y)|$, the edge labels are all different



Graceful Labelling: Constraints

- A CSP model has:
  - a variable for each node, $x_1, x_2, \ldots, x_n$ each with domain $\{0, 1, \ldots, q\}$
  - auxiliary variables for each edge, $d_1, d_2, \ldots, d_q$ each with domain $\{1, 2, \ldots, q\}$
- $d_k = |x_i - x_j|$ if edge k joins nodes i and j
- $x_1, x_2, \ldots, x_n$ are all different
- $d_1, d_2, \ldots, d_q$ are all different
- it is efficient to enforce GAC on the constraint allDifferent($d_1, d_2, \ldots, d_q$)
- but not on allDifferent($x_1, x_2, \ldots, x_n$)
  - in the example, $n = 9$, $q = 16$

Cork Constraint Computation Centre

One Constraint is Better than Several (maybe)

- If there are several constraints all with the same scope, rewriting them as a single constraint will lead to more propagation…
  - **if** the same level of consistency is maintained on the new constraint
- … more propagation means shorter run-time
  - **if** enforcing consistency on the new constraint can be done efficiently

Cork Constraint Computation Centre

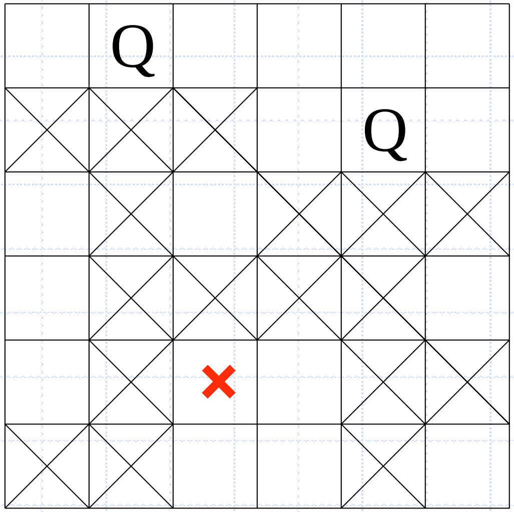
Example: *n*-queens

- A variable for each row, $x_1, x_2, \ldots, x_n$
- Values represent the columns, 1 to n
- The assignment (x_i, c) means that the queen in row i is in column c
- Constraints for each pair of rows i, j:
  - $x_i \neq x_i$
  - $x_i - x_i \neq i - j$
  - $x_i - x_i \neq j - i$

Cork Constraint Computation Centre

Propagating the Constraints

- A queen in row 5, column 3 conflicts with both remaining values for x_3
- But the constraints are consistent
  - $x_i \neq x_j$ thinks that $(x_3,1)$ can support $(x_5,3)$
  - $x_i - x_j \neq i - j$ thinks that $(x_3,3)$ can support $(x_5,3)$



- Enforcing AC on the conjunction $(x_i \neq x_j) \wedge (x_i - x_j \neq i - j) \wedge (x_i - x_j \neq j - i)$ would remove 3 from the domain of x_5
  - but how would you do it?

Summary

- The viewpoint (variables, values) largely determines what the model looks like
- Choose a viewpoint that will allow the constraints to be expressed easily and concisely
- Be aware of global constraints provided by the solver, and use them if they reduce run-time
- Introduce auxiliary variables if necessary to help express the constraints