CSP: Constraint Programming

Pedro Meseguer
IIIA-CSIC
Bellaterra, Spain

Overview

Constraint Programming

- Modelling
  - Search space size
  - Primal / Dual models
  - Global constraints
- Solving
  - Guidelines
  - CP Styles

Constraint Programming

CP:
- provides a platform for solving CSPs
- proven useful in many real applications

Platform:
- set of common structures to reuse
- best known algorithms for propagation & solving

Two stages:
- modelling
- solving

CP: Modelling

Modelling decisions: select among alternatives
- the choice of the variables
- the choice of the domains

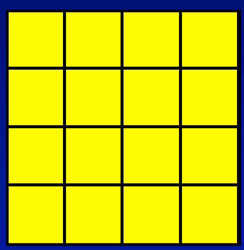
$\Big\}$ *search space size*

- how we state the constraints ← *space reduction*

Example: Map Colouring
- variables: are regions or colours?

Any CSP can be modelled in different ways
- Efficiency of algorithms can vary dramatically
- No strong results are known
- Formulating an effective model is not easy, requires considerable skills in modelling

N-queens: Model 1



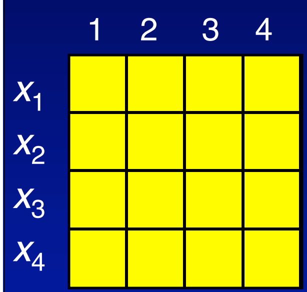
Variables: n^2, one per cell, matrix B n x n

Domains: {0,1}, B[a,b]=0, no queen
B[a,b]=1, queen

Constraints: If B[a,b] = 1 then

same row B[_,b]=0 □
same column B[a,_]=0 □
same diagonal B[a+□,b+□]=0, B[a-□,b-□]=0 □
same diagonal B[a-□,b+□]=0, B[a+□,b-□]=0

N-queens: Model 2



Variables: n, one per row

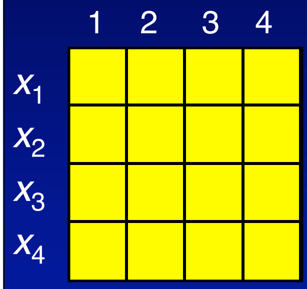
Domains: {0,1,…,n-1}, queen column

Constraints:

different columns $x_i \neq x_j$ □
different diagonals $| x_i - x_j | \neq | i - j |$

Different row constraint is included in the formulation!!

N-queens: Model 3



Variables: *n*, one per row

Domains: {0,1,…,*n*-1}, queen column

Constraints:

different columns   all-different($x_1, x_2, \ldots, x_n$)

different diagonals $| x_i - x_j | \neq | i - j |$

Different row constraint is included in the formulation!!

N-queens Models

	Model 1	Model 2	Model 3
Search space size $d^{\#vars}$	2^{n2} 4 65,536 10 1.27 E30 20 ERROR!!	n^n 256 1.00 E10 1.05 E26	n^n 256 1.00 E10 1.05 E26
Constraints number prunning	n rows n columns 2(n-1) diagonals	n columns 2(n-1) diagonals Equal model 1	1 all-diff 2(n-1) diagonals More than model 2

Constraint Formulations

Binary (arity ≤ 2) :

- conceptually simple, easy to implement
- may generate weak formulations

Non-binary (arity > 2) :

- more complex constraints
- GAC: stronger (filter more) than AC on equivalent binary decomposition

Equivalence: any non-binary CSP can be reformulated as a binary one

Primal / Dual Formulations

Primal CSP: (X, D, C)

$X = \{x_1, x_2, \dots, x_n\}$, $D = \{d_1, d_2, \dots, d_n\}$, $C = \{c_1, c_2, \dots, c_r\}$

Dual CSP: (X', D', C')

$X' = \{x'_1, x'_2, \dots, x'_r\}$, $D' = \{d'_1, d'_2, \dots, d'_r\}$, $C' = \{c'_{ij}\}$

one variable per primal constraint

$d'_i = rel(c_i)$

values=permitted primal tuples

$var(c'_{ij}) = \{x'_i, x'_j\}$

$\forall c'_{ij} \in C'$ $\exists$

$var(c_i) \cap var(c_j) \neq \emptyset$

$rel(c'_{ij})$=same values for shared primal vars

$\boxed{\textbf{Always binary!!}}$

Example: Crossword puzzles

1	2	3	4	5
6	7	8		9
10	11	12	13	14
15		16	17	18
19	20	21	22	23

a
aardvark
aback
abacus
abaft
abalone
abandon
...

monarch
monarchy
monarda
...
zymurgy
zyrian
zythum

Primal model (Non-binary)

1	2	3	4	5
6	7	8		9
10	11	12	13	14
15		16	17	18
19	20	21	22	23

<u>variables</u>: cells

<u>domains</u>: ‘a’, …, ‘z’

<u>constraints</u>: contiguous letters must form words in dictionary

Dual model (binary)

1	2	3	4	5
6	7	8		9
10	11	12	13	14
15		16	17	18
19	20	21	22	23

<u>variables</u>: words across and down

<u>domains</u>: words from dictionary

<u>constraints</u>: intersecting words must agree on common letter

Hidden Variable Formulation

Primal CSP: (X, D, C) *binary* *non-binary*

$X = \{x_1,\ldots, x_n\}$, $D = \{d_1,\ldots,d_n\}$, $C = B \cup \{\underline{c}_a,\ldots,\underline{c}_q\}$

Hidden formulation: (X', D', C')

$X' = X \cup \{\underline{x}_a,\ldots,\underline{x}_q\}$, $D' = D \cup \{\underline{d}_a,\ldots,\underline{d}_q\}$, $C' = B \cup \{\underline{c}_{iq}\}$

a new variable per non-binary constraint

$\underline{d}_p = rel(\underline{c}_p)$

values=permitted primal tuples

$var(\underline{c}_{ip}) = \{x_i, \underline{x}_p\}$
$\forall \underline{c}_{ip} \in C' \Rightarrow$
$x_i \in var(\underline{c}_p) \Rightarrow$
$rel(\underline{c}_{ip})$= same values for x_i

hidden variables

Global Constraints

Real-life constraints: often complex, non-binary

c is global iff:

- arity(c) > 2
- c is logically equivalent to $\{c_1, c_2, \ldots, c_k\}$ binary
- **AC**(c) *prunes more* than **AC**($c_1, c_2, \ldots, c_k$)

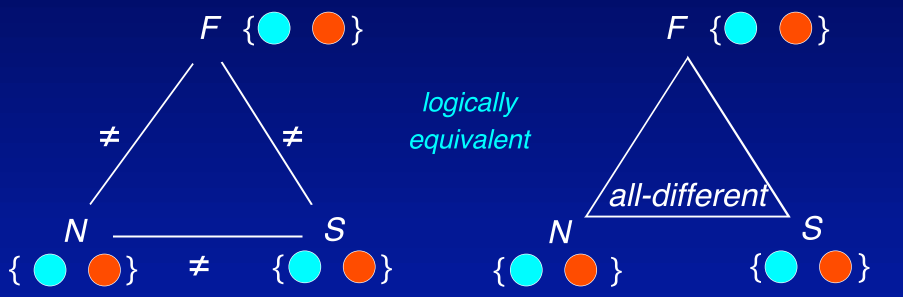
Propagation:

- specialized algorithms
- exploit constraint semantics

decrease AC complexity

Example: *all-different*

Var: F, N, S; Val: { ● ● }; Ctrs: $N \neq S \neq F \neq N$



3 binary constraints,
they are AC,
no pruning

1 ternary constraint, not AC,
AC pruning □ empty domain
no solution!!

Example: *all-different*

Enforcing arc-consistency:

- *n* variables, *d* values
- *n(n-1)/2* binary constraints : $O(n^2 d^2)$
- 1 *n*-ary constraint:
  - general purpose algorithm $O(d^n)$
  - specialized algorithm $O(n^2 d^2)$

CP: Solving

Solving decisions: select among alternatives

- search algorithm
- local consistency: level / how often
- heuristics: variable / value

interleaved

Example: Map Colouring

- static or dynamic variable ordering ?

Efficient solving:

- reasonable initial size of the search space
- drastic *reductions* of space during search

CP Solving: Some Guidelines

Easy/hard problems:

- hybrid search
- dynamic variable ordering: min domain / degree
- easy: FC / hard: MAC

One solution/All solutions:

- one solution: hybrid search
- all solutions: hybrid search or complete inference

For specific problems (scheduling, routing…) check:

- formulation, global constraints
- heuristics, experiences

CP: Declarative Programming

Declarative Programming: you declare

- Variables
- Domains
- Constraints

and ask the SOLVER to find a solution!!

SOLVER offers:

- Implementation for variables / domains / constraints
- Hybrid algorithm: backtracking + incomplete inference
- Global constraints + optimized AC propagation
- Empty domain detection
- Embedded heuristics

Constraint Logic Programming

Logic Programming:

- Depth-first search
- Unification: *substitute* equals by equals clauses/database

special case of constraint solving

substituted by

More general constraint solver → Constraint Logic Programming

Existing solvers:

- Chip, Eclipse, Mozart, Sictus Prolog (and many others)

Constraint Programming Libraries

Library to be included in your program:

- Imperative programming

Provides:

- Special objects:
  - Variables / Domains / Constraints (global)
- Special functions to find:
  - One solution / the next solution

Existing Solvers:

- Ilog Solver, Choco