CSP: Solving by Inference

Pedro Meseguer
IIIA-CSIC
Bellaterra, Spain

Overview

Complete Inference

- Motivation: tractability
- Backtrack-free problems
- Directional consistency
- Adaptive consistency
- Constraint operations
- Bucket elimination

Inference

Inference: $P \longrightarrow P'$

legal operations on variables, domains, constraints

- *P'* is equivalent to *P*: *Sol(P) = Sol(P')*
- *P'* is *presumably easier* to solve than *P*
  - *smaller* search space
  - constraints are *more* explicit

Inference can be:

- complete: produces the solution
  adaptive consistency
- incomplete: requires further search
  arc consistency

Tractability

CSP: NP-complete

- Unless $P = NP$, exponential solving algorithms
- Backtracking: $O(d^n)$

Any tractable classes?

- Polynomial complexity
- Search: decisions are permanent
  they don't have to be reconsidered
  ↓
  backtrack-free search

Tractability Dimensions

A CSP class could be tractable because its:

- Restricted Structure: topological properties of the constraint hypergraph
- Restricted Relations: particular properties of constraint relations
- Both combined: very little is known

K-Consistency

K-Consistency:

- for any subset of k-1 variables $\{X_1, X_2, \ldots, X_{k-1}\}$ consistently assigned;
- for any X_k there exists $d \in D_k$ such that $\{X_1, X_2, \ldots, X_k\}$ is consistent

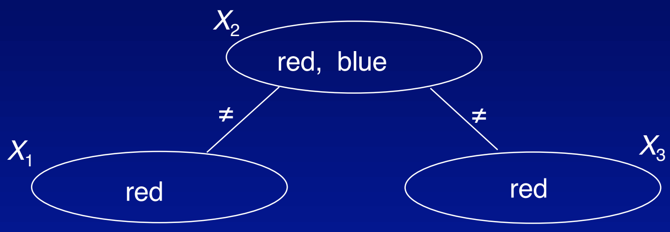
K-strong-consistency: J-consistent, for $1 \leq J \leq K$

Algorithms for K-strong-consistency:

- Freuder 82, Cooper 89, O(exp K)

Example: K-consistency

K-consistency does not imply K-strong consistency



Example:

- 3-consistent: for any pair of consistently assigned variables, there exists a consistent value for the third variable
- Not 2-consistent: arc X2 - X1

Primal Graph Width

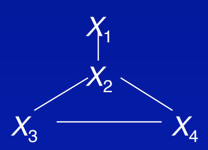
Variable ordering: $\{X_1, X_2, \ldots, X_n\}$

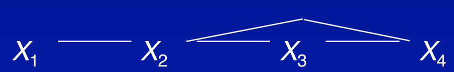
Node width: #arcs to previous nodes

Ordering width: $\max_i$ {node width X_i }

Graph width: min {ordering width}

Example:





width = 2

Backtrack-Free Problems

THEOREM: Given a variable ordering of width *K*, the problem can be solved without backtracking if the level of strong consistency is greater than *K* [Freuder 82]

Algorithms:

- K-strong consistency: O(exp *k*)
- Adds extra arcs, width increases
- No adding arcs for width = 1
- Trees have width = 1
- *Tree problems: backtrack-free after 2-strong consistency*

Tree Problems

Tree □ No cycles in the primal graph

- Limited 3-queens: a queen attacks adjacent rows

X_1 —— X_2 —— X_3 width = 1

c_{12} is not 2-strong consistent because value 2 of d_1

c_{21} is not 2-strong consistent because value 2 of d_2

c_{23} is not 2-strong consistent because value 2 of d_3

	1	2	3
x_1	Q		
x_2			Q
x_3	Q		

Polynomic cost !!

Directional Consistency

k

$\ldots$

$X_1, X_2, \ldots, X_i, \ldots, X_n$

K-strong-consistency: is more than needed

- variables will be assigned in order
- consistency bet. $(X_2, \ldots, X_i)$ and X_1 is not required

K-Directional consistency:

- for any subset of k-1 vars. $\{X_i, X_j, \ldots, X_l\}$ assig.cons.
- X_m, $m > i,j,..,l$, $\exists d \in D_k \ni \{X_i, X_j, \ldots, X_l, X_m\}$ is cons.

Previous results apply

ADC: Motivation

Backtrack-free theorem: WHY

- K+1 consistency?
  Because at least one node requires such a level
- K+1 strong consistency?
  Because some nodes may require cons < K+1
- Strong cons may increase graph width !

What about:

- Adjusting the consistency level for each node?
- Taking into account width increments ?

Adaptive Consistency

Adjusting the consistency level for each node:
- When processed, node must have the final width
- Achieve consistency with its K parents

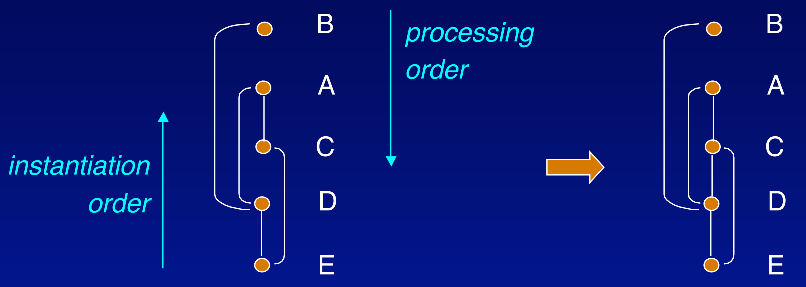
Taking into account width increments:
- Nodes are processed from last to first
- Width increments on nodes not processed yet:
  - New constraints: on parents
  - After processing a node, no changes in its width

Adaptive Consistency (II)

ADC(X_1, X_2,. . . .,X_n)
For $i = n$ to 1 do
  consistency(X_i , *parents*(X_i))
  connect by arcs all elements in *parents*(X_i))

THEOREM: An ordered constraint graph processed by ADC is backtrack-free

ADC: Induced Graph



Induced graph:

- Graph after processed by Adaptive consistency
- Induced width: width of the induced graph: w^*

ADC complexity:

- Time $O(n\,(2d)^{w^*+1})$ Space $O(n\,d^{w^*})$

Constraint Operations: Projection

Projection: c and $x \in var(c)$, projecting x out of c: $c\Downarrow_x$

- $var(c\Downarrow_x) = var(c) - \{x\}$
- $rel(c\Downarrow_x)$: tuples of $rel(c)$ removing x' value

duplicated tuples are removed

Example:

c

x	*y*	*z*
a	*b*	*b*
b	*a*	*c*
c	*b*	*a*
c	*b*	*b*

→

$c\Downarrow_x$

x	*y*	*z*
a	*b*	*b*
b	*a*	*c*
c	*b*	*a*
c	*b*	*b*

Constraint Operations: Join

Join: $c\ c'$, $c \bowtie c' = c''$

- $var(c'') = var(c) \cup var(c')$
- $t \in rel(c'')$ iff $t[var(c)] \in rel(c)$ and
  $t[var(c')] \in rel(c')$

Example:

x	*y*
a	*a*
b	*a*
b	*b*
c	*c*

$\bowtie$

x	*z*
a	*a*
b	*c*

$\longrightarrow$

x	*y*	*z*
a	*a*	*a*
b	*a*	*c*
b	*b*	*c*

Synthesis: A single constraint

A single constraint:

- Synthesis of all problem constraints
- Its tuples are the problem solutions

Join all problem constraints:

$$\bowtie_{c_i \in C} c_i$$

Limitations:

- Very costly
- More than needed $\rightarrow$ *bucket elimination*

Bucket Elimination

Problem P, var x, C_x ={constraints on x}

Idea:

- Substitute C_x by a new constraint <u>c</u>
- <u>c</u> summarizes the effect of C_x on P
- <u>c</u> does not mention x

now x is isolated: it can be eliminated

variable elimination

Process:

problems:	P →	P' →	P'' →	. . . →	$P^{(n-1)}$
#vars:	n	n-1	n-2	. . .	1

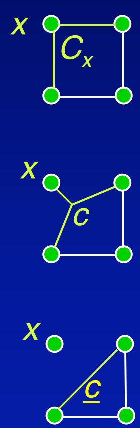
trivially solved

← *solution without search*

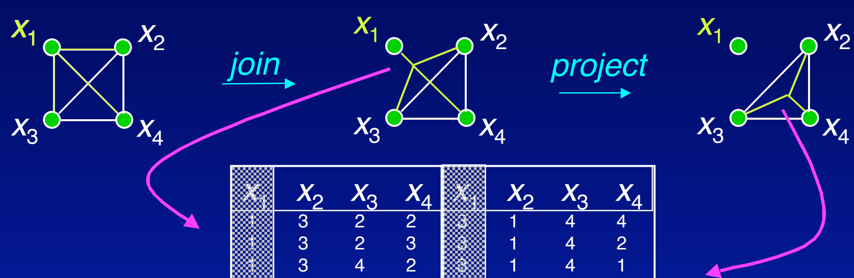
Variable Elimination

To eliminate var x:

- *Join* all constraints in C_x → c
- *Substitute* C_x by c
- *Project out* variable x from c → <u>c</u>
- *Substitute* c by <u>c</u>

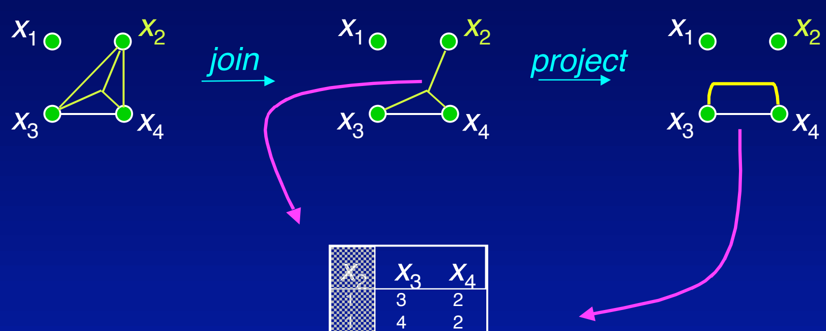


Example: 4-queens (x_1)



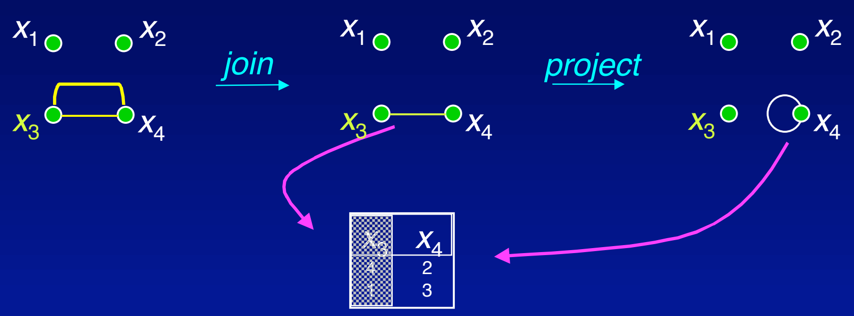
x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
1	3	2	2	3	1	4	4
1	3	2	3	3	1	4	2
1	3	4	2	3	1	4	1
1	3	4	3	3	1	2	4
1	4	2	2	3	1	2	2
1	4	2	3	3	1	2	1
1	4	4	2	4	2	3	3
1	4	4	3	4	2	3	2
2	4	1	1	4	2	1	3
2	4	1	3	4	2	1	2
2	4	1	4	4	1	3	3
2	4	3	1	4	1	3	2
2	4	3	3	4	1	1	3
2	4	3	4	4	1	1	2

Example: 4-queens (x_2)



x_2	x_3	x_4
1	3	2
1	4	2
1	4	4
4	2	3
4	1	3
4	1	1

Example: 4-queens (x_3)



Example: All Solutions 4-queens

	x_4	x_3	x_2	x_1
SOLUTION 1:	2	4	1	3

x_4
2
3

x_3	x_4
4	2
1	3

x_2	x_3	x_4
1	3	2
1	4	2
1	4	4
4	2	3
4	1	3
4	1	1

x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
1	3	2	3	3	1	4	4
1	3	2	3	3	1	4	2
1	3	4	2	3	1	4	1
1	3	4	3	3	1	2	4
1	4	2	2	3	1	2	2
1	4	2	3	3	1	2	1
1	4	4	2	4	2	3	3
1	4	4	3	4	2	3	2
2	4	1	1	4	2	1	3
2	4	1	3	4	2	1	2
2	4	1	4	4	1	3	3
2	4	3	1	4	1	3	2
2	4	3	3	4	1	1	3
2	4	3	4	4	1	1	2

Example: All Solutions 4-queens

SOLUTION 2: x_4 = 3, x_3 = 1, x_2 = 4, x_1 = 2

x_4
2
3

x_3	x_4
4	2
1	3

x_2	x_3	x_4
1	3	2
1	4	2
1	4	4
4	2	3
4	1	3
4	1	1

x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
1	3	2	3	3	1	4	4
1	3	2	3	3	1	4	2
1	3	4	2	3	1	4	1
1	3	4	3	3	1	2	4
1	4	2	2	3	1	2	2
1	4	2	3	3	1	2	1
1	4	4	2	4	2	3	3
1	4	4	3	4	2	3	2
2	4	1	1	4	2	1	3
2	4	1	3	4	2	1	2
2	4	1	4	4	1	3	3
2	4	3	1	4	1	3	2
2	4	3	3	4	1	1	3
2	4	3	4	4	1	1	2