Constraint Programming: Consistency techniques

Constraint Propagation & Fixed point algorithm

Prof. Dr. Carmen Gervet

Core Techniques

- Constraint propagation
  - Consistency notions
  - Consistency algorithms
  - Fixed point
- Basic tree search algorithm
  - Standard backtracking
  - Forms of lookahead

Constraint Programming

- Paradigm with roots in Artificial Intelligence
  - Tackle combinatorial search problems mainly
  - Holy Grail: “The user states the problem, the system solves it!”
- Foundation principle
  - Constraint Programming = Model + Solver
- Applications
  - Puzzles: 8-queens, zebra, social golfer, Sudoku
  - Planning & scheduling, timetabling, vehicle routing, networks, combinatorial designs, computational biology

Constraint Modelling & Solving

- Core principle
  - Manipulation of elements from a finite set, or domain
  - Combine the deterministic removal of inconsistent values from domains with the non-deterministic search for solution(s)
- Concepts studied
  - CSP
  - Consistency methods
  - Basic search methods
- Existing Languages & systems
  - Sicstus Prolog , ECL^{i}PS^e (Cisco Systems), Solver (IBM-ILOG),, COMET, …

CSP: The Sudoku analogy

- Sudoku problem

Given a square of order 3, consisting of

- 3^4 cells formed into a 3^2 x 3^2 grid with values from 1 to 3^2 such that
- Cell values in each row, each column and in each of 3^2 major blocks are all different.
- Some cells are given
- It has a unique solution!

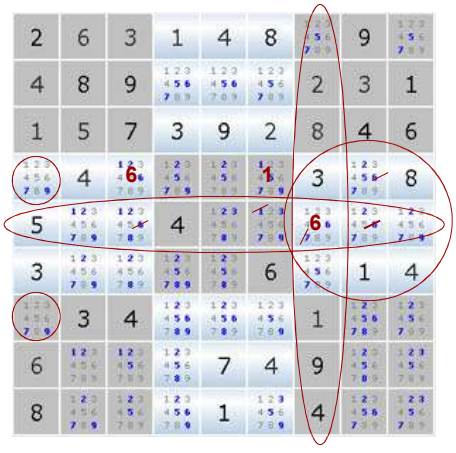
Sudoku Instance of order 3

- Solution

A solution to a Sudoku is an assignment of values to each cell such that all the constraints are simultaneously satisfied.

The Sudoku analogy

- CSP
  - Attach a domain of values to each cell
- Domain reduction
  - Identify and remove values that can never be assigned to a cell
- Propagate changes
  - Until no more domain filtering occurs



Constraint Satisfaction Problems

- Definition
  - A CSP is a triple <V,D,C>
    - V: set of Variables,
    - D: set of Domains associated with each variable (in N, 2^N, $\Re$)
    - C: set of constraints applied to the variables
  - A solution to a discrete CSP
    - Tuple ($x_1=v_1$, $x_2=v_2$, $x_j=v_j$,…,$x_n=v_n$)
- Tasks
  - Model a combinatorial search problem as a CSP
    - There can be more than one formulation
  - Solve a CSP
    - Assign a value to each variable such that all the constraints are satisfied simultaneously

Classification of CSPs

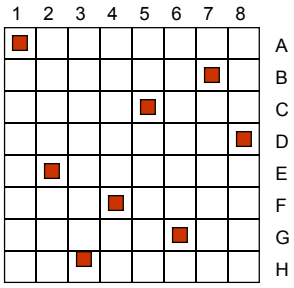
- Binary CSPs
  - Constraints over 2 variables at most
- N-ary CSPs
  - Constraints over any size set of variables
- Type of solution searched for
  - One solution (satisfiability problem)
  - All solutions (complete search)
  - Optimal solution (optimization problem)

CSP example

- The n-queens problem

  Given an integer N, place N queens on N distinct square in a N*N chessboard, such that no two queens should threaten each other

A solution to the 8-queens problem



Building a CSP model

- Choice of decision variables

  e.g. each queen Q_i
- Choice of domain elements
  1. A position on the 8x8=64 board
  2. The assignment of a row to each column
- Combinatorics of the model
  1. Allows 64^8 combinations of an 8-tuple solution
  2. Allows 8^8 combinations of an 8-tuple solution

     This formulation in fact has built into it the constraint that no two queens can be on the same row

Building a CSP model

- Choice of decision variables
  e.g. each queen Q_i
- Choice of domain elements
  1. A position on the 8x8=64 board
  2. The assignment of a row to each column
- Combinatorics of the model ($|d|^{|v|}$)
  1. Allows 64^8 combinations of an 8-tuple solution
  2. Allows 8^8 combinations of an 8-tuple solution

     This formulation in fact has built into it the constraint that no two queens can be on the same row

Problem formulation: Constraints

- If Q_i represents a row

$$\forall i: Q_i \in \{1,..,8\}$$
$$\forall i,j: Q_i \neq Q_j$$
$$\forall i,j,i<j: Q_i - Q_j \neq j - i$$
$$\forall i,j,i<j: Q_i - Q_j \neq i - j$$

- If Q_i represents a position on the board

$$\forall i: Q_i \in \{1,..,64\}$$
$$\forall i: R_i \text{ is } (Q_i \text{ div } 8) + 1$$
$$C_i \text{ is } (Q_i \text{ mod } 8) + 1$$
$$\forall i,j: R_i \neq R_j, C_i \neq C_j$$
$$\forall i,j,i<j: R_i - R_j \neq C_i - C_j$$
$$\forall i,j,i<j: R_i - R_j \neq C_i - C_j$$

General case: Representing Constraints in CSPs

- Constraints as relationships

$Q_1 \overset{\neq}{\longleftrightarrow} Q_2$

- Constraints as collection of allowed tuples

$Q_1 \longleftrightarrow Q_2$

<1,2>
<1,3>
...
<1,8>
<2,1>
<2,3>
…
<2,8>
...

Characteristics of the n-queens problem

- Binary CSP
- Properties of the constraint graph described
  - N-queens describes a fully connected *complete graph*
  - N *var (nodes)* 1/2 Nx(N-1) *(arcs)*
- Tightness of the model
  - Each potential variable label conflicts with atmost 3 values of each other variable
  - Constraints get looser as N grows larger (problem tightness)
- Problem size; scalability issues
  - In general there are N^N candidate solutions to be considered (second model)

General Characteristics of CSPs (2/2)

- Structure of the constraint graph
  - graph/hypergraph/tree
  - Efficiency of search affected by connectivity of nodes
- Degree of tightness
  - For a constraint =

    $$\frac{\text{Number of tuples satisfying a constraint}}{\text{Number of all tuples between the variables}}$$

  - For a CSP problem =

    $$\frac{\text{Number of solution tuples}}{\text{Number of all distinct tuples for all variables}}$$

  For loose problems many leaves of the search tree represent solutions

General Characteristics of CSPs (1/2)

- NP-complete
- Type of solution required

  1, all, optimal (CSOP)
- Problem size
  - Nb of variables: determines depth of the search tree
  - Domain sizes: determines number of branches
  - Nb of constraints: number of consistency checks to be performed
- Types of vars & constraints
  - Vars affect the type of methods to be used
  - Constraints: binary, n-ary, global

Constraint Network/ graph

- Definition
  - Set of variables and constraints that inter-relate and define the valid values for the variables
    - Nodes are variables
    - Arcs are constraints between variables
- Remark
  - A constraint involving more than 2 variables belongs to a hyper-graphs
    - Hyper-arcs: arcs involving more than 2 variables

Constraint Programming techniques

- Consistency notions
  - 1-consistency
    every value in every domain satisfies the unary constraints over the variable
  - k-consistency
    removes all inconsistencies involving all subsets of size *k* of the variables
- Inference mechanism
  - Transform a CSP into an equivalent one
  - Remove domain values that cannot belong to any solution
  - Identical sets of variables & solution tuples

Consistency over binary CSPs

- Consider a network of binary constraints

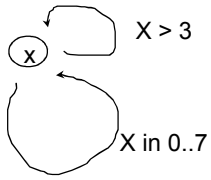
  Nodes are variables $_i$ with domain D_i and arcs constraints P_i

- Node-consistency

  1-Consistency of unary constraints.

  Node $_x$ is consistent iff for any value v in D_x, $P_x(v)$ holds

Example



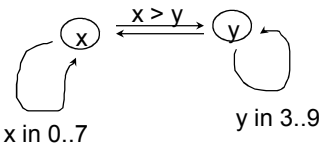
not node consistent because 0,1,2,3 $\not>$ 3

Arc-consistency

- **Definition**

  An arc $P_{xy}(v_x, v_y)$ is arc-consistent (2-consistent) iff for any value v_x in D_x such that $P_x(v_x)$, there is a value v_y in D_y such that $P_y(v_y)$ and $P_{xy}(v_x , v_y)$

- Example



not arc-consistent because x=0,1,2,3 $\not>$ 3
and y = 7,8,9 $\not<$ x

Path-consistency

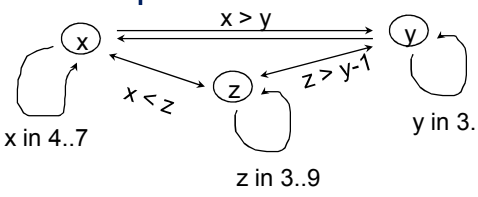
- Definition

  A path (x,z_1,..,z_i, …,y) is path consistent iff for any values v_x in D_x and v_y in D_y, such that $P_x(v_x)$, $P_y(v_y)$ and $P_{xy}(v_x , v_y)$ hold,

  there is a sequence of values v_{z_1} in D_{z1}, …, $v_{z_{m-1}}$ in $D_{z_{m-1}}$ such that

  - 1) $P_{z1}(v_{z_1}),\ldots, P_{z_{m-1}}(v_{z_{m-1}})$
  - 2) $P_{xz_1}(v_x, v_{z_1})$, $P_{z_1z_2}(v_{z_1}, v_{z_2})$, …, $P_{z_{m-1}y}(v_{z_{m-1}}, v_y)$.

- Example



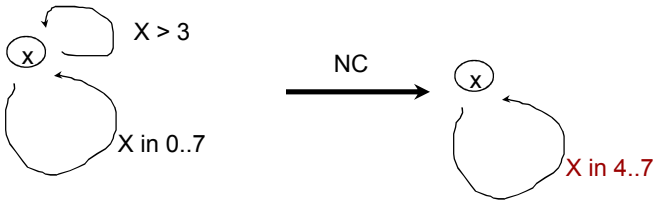
it is path-consistent because for any x and y there is a value of z that holds

Node consistency algorithm

Procedure NC(x):

$$D_x \leftarrow D_x \cap \{v_x | P_x(v_x)\}$$

- begin
- for x in {set of nodes} do NC(x)
- End



- Worst case time complexity O(*dn*), d largest domain size, n number of variables

How to achieve arc consistency

Main procedure

```
procedure REVISE(x, y) :
begin
DELETE ← false
for each v_x ∈ D_x do
  if there is no v_y ∈ D_y such that C_xy(v_x, v_y) then
    begin
      delete v_x from D_x;
      DELETE ← true
    end
return DELETE
```

REVISE (x,y)

- Remove from X's domain all values that do not have support in the domain of Y
- Worst case time complexity $O(d^2)$
  d: largest domain size

Arc-consistent system of constraints: AC-3

```
for x ∈ {set of nodes} do NC(x)                                   O(dn)
Q ← {(x,y) | (x,y) ∈ arcs(G), x ≠ y}                              O(2e)
while Q not empty do                                              2e + d(2e-n)
  begin
    select and delete any arc (k,m) from Q
    if REVISE(k, m) then Q ← Q ∪ {(i,k) | (i,k) ∈ arcs(G), i ≠ k, i ≠ m}
  end   O(d^2)                                                    max number of arcs added O(e_k-1)
end
```

AC-3

- Time complexity results

Number of entries to the while loop

$$\sum_{k=1}^{n} d \times (e_k - 1) = d \times (e_1 + e_2 + \ldots + e_n) - d \times n = d \times 2e - d \times n$$

Worst time complexity $O(d^2(2e + d(2e-n)))$

The graph is at least connected (otherwise independent components) in which case e >= n-1.

Worst time complexity $O(ed^3)$

For a complete graph e= 1/2 n(n-1)

Worst time complexity $O(d^3n^2)$

Implementing consistency algorithms

- Built-in constraints & user-defined constraints have their own definition of REVISE which might achieve different levels of consistency
  - e.g. x>y, x = y, x = y + 1
- The generic fixed point algorithm (AC-3) calls each REVISE or inference rule in a data driven way

Semantics of inference rules

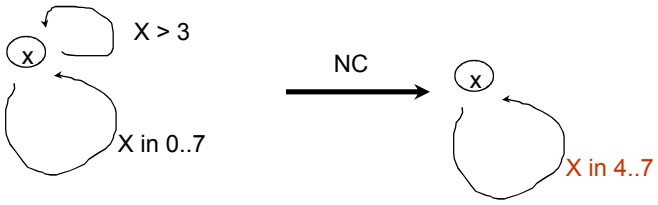
- Define reduction on domains
  - For each constraint and each variable per constraint
- Semantic
  - Given some condition, update variable domains such that the consistency notion is enforced
- Layout

Conditions to be satisfied

{Changes to the constraint store}

Example: enforce node consistency for X> 3

- Constraint network



- Reduction rule

min' = max(min,3+1)

{X ∈ min..max, X > 3} → {X ∈ min'..max}

Reduction rules to achieve AC

- Define inference rules to achieve AC in the following constraints?

$x \in D_x, y \in D_y, z \in D_z$

$x = y$

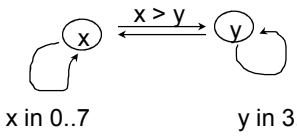
$x \geq y$

$x = y + 3$

Interval/bound reasoning

- Considering all the domain values can be expensive
- Reasoning over the domain bounds alone can be effective enough
  - Check that all bounds are possible instances of the variables wrt the connected constraints
- When to consider interval consistency?
  1. For n-ary constraints
     x=y+z+t+u+v
  2. For binary *monotonic* constraints
     - (equivalent to arc-consistency)

Example of Bound Reductions



- Inference rules

  $\min(x) \leftarrow \max(\min(x), \min(y))$

  $\max(y) \leftarrow \min(\max(x), \max(y))$

Notion of fixed point

- Propagating domain changes
  - If we reduce a domain, some constraint in the network might become inconsistent
  - Make sure the whole constraint network remains consistent or fails
- Fixed point algorithm
  - The consistency algorithm is applied till no further variable domain is reduced or a failure (empty domain) is detected
  - Process referred to as constraint propagation

Embedded in the generic algorithm

- Example: AC-3

for x ∈ {set of nodes} do NC(x)
$Q \leftarrow \{(x,y) \mid (x,y) \in arcs(G), x \neq y\}$
while Q not empty do
  begin
    select and delete any arc (k,m) from Q
    if REVISE(k, m) then $Q \leftarrow Q \cup \{(i,k) \mid (i,k) \in arcs(G), i \neq k, i \neq m\}$
  end
end

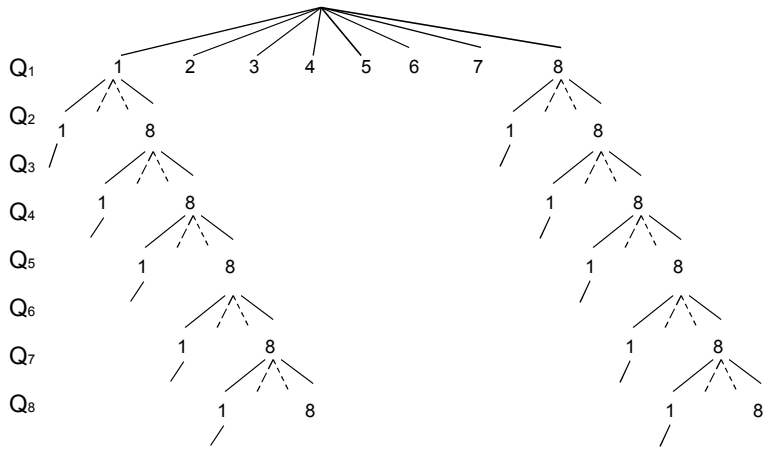
General Constraint Solver

- Objective
  - Assign a value to each variable such that all the constraints are satisfied simultaneously
- Components
  - Inference
    - Domain reduction & constraint propagation
  - Search
    - Backtracking, variable ordering heuristics, advanced search techniques
    - X = 2 …. or X = 3 ….

What is search

- Inference
  - Removes inconsistent values that can never be part of a solution
  - Entirely deterministic procedure
  - Inference is often not enough to obtain a solution (or complete assignment)
- Search
  - Seeks one/all or the optimal solution by choosing values for variables
  - In combinatorial problem this is a nondeterministic procedure
  - Effective search: minimize the bad choices!

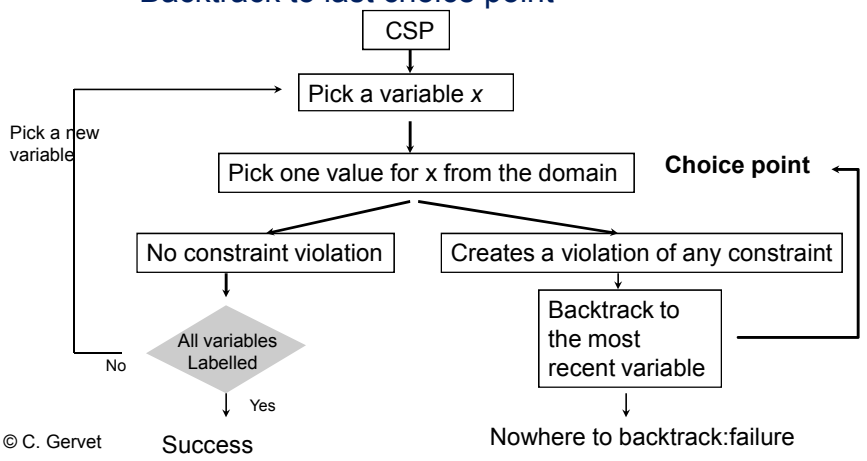
Example: Search space associated with 8-queens



Standard backtracking

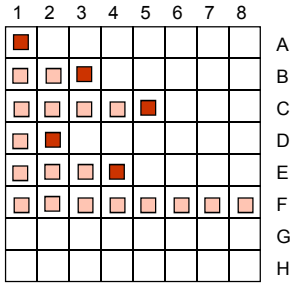
- Also called chronological backtracking

  Backtrack to last choice point



Example

- Standard backtracking for 8-queens problem



A 1 (queen A in column 1)

B 123 (queen B in column 1: fails, column2: fails,

column 3: succeeds)

C 12345

D 12

E 1234

F 12345678 fail backtrack to E

- Complexity
  - n^n candidate solutions, for each candidate solution all the constraints must be checked once in the worst case. $O(n^n e)$

Issues

- Expensive procedure
  - In the worst case time complexity
  - Tries every single combination
- Naïve
  - Does not take into account the constraint semantics
  - Runs separately from the constraint inference algorithms
- Improvements with lookahead
  - Embed degrees of inference into search

Summary

- CP = Model + Solver
- Solver = Inference + Search
- Inference
  - Filters domains & propagates constraints to reduce domains till no changes occur
  - If variables are unassigned search is required
- Search
  - Make non deterministic choices to label variables and seek a complete assignment
- Next time
  - Advanced search techniques and optimization