

# Introduction to constraint programming

## Lecture III

Per Kreuger `piak@sics.se` &  
Markus Bohlin `bola@sics.se`

November 2002

# Properties of solution methods

**Soundness** Each result returned by an algorithm is a valid solution

**Completeness** Every valid solution can be found by an algorithm

These are desirable properties but for practical purposes often compromised

# Classes of methods

**Search** for solutions in space of compound labels for all variable in problem

- Many methods exist

**Problem reduction** transform problem to *equivalent* one in which solutions are easier to find

- E.g. domain reduction

# Search

- Systematic search
  - Generate and test (GT)
  - Backtracking (BT)
  - Backtracking with look-ahead
- Heuristics (To be revisited)
  - Local search methods e.g.
  - Hill climbing

# Search algorithms

---

## Algorithm 1 Generate and Test (GT) in Prolog

---

```
gt(Variables, Constraints, Solution):-  
    generate(Variables, Solution),  
    test(Constraints, Solution).
```

```
generate([V::D|RemainingVariables],[V-X|PartialSolution):-  
    select_value(X,D),  
    generate(RemainingVariables,PartialSolution).  
generate([],[]).
```

```
test([C|RemainingConstraints],Solution):-  
    test_constraint(C,Solution),  
    test(RemainingConstraints,Solution).  
test([],_).
```

---

---

## Algorithm 2 Backtracking (BT) in Prolog

---

```
bt([V::D|RemainingVariables],Constraints,PartialSolution,Solution):-
    select_value(X,D),
    NewPartialSolution=[V-X|PartialSolution],
    test(Constraints,NewPartialSolution,RemainingConstraints),
    bt(RemainingVariables,RemainingConstraints,NewPartialSolution,Solution).
bt([],[],Solution,Solution).
```

```
test([C|RemainingConstraints],PartialSolution,NonTestedConstraints):-
    (can_be_tested(C,PartialSolution)
     -> test_constraint(C,PartialSolution),
        NonTestedConstraints=RemainingConstraints
     ; NonTestedConstraints=[C|RemainingConstraints]),
    test(RemainingConstraints,PartialSolution,RemainingConstraints).
test([],_,[]).
```

```
can_be_tested(Constraint,PartialSolution):-
    variables(Constraint,Variables),
    are_instantiated(Variables,PartialSolution).
```

---

# Problem reduction

In which we aim to reduce variable domains and strengthen constraints

**Equivalence** Same variables and solutions

$A$  **reduce to**  $B$  if and only if

1.  $A$  equivalent to  $B$
2. Every domain in  $B$  is a subset of corresponding domain in  $A$
3. For every constraint  $C_S$  in  $A$  over a set of variables  $S$ , the set of compound labels that satisfies *all* constraints  $C_S^i$  in  $B$  over the same set of variables is a subset of those satisfying  $C_S$

C.f. Strengthening

# Redundancy and minimality

## Redundancy

**Domains** Value in domain that does not occur in any solution

**Constraints** Constraint not a projection of any solution

## Minimality

- No redundant domain values
- No redundant constraint tuples
- NP-complete to achieve for general CSP
- Efficient *incomplete* methods exist

# Combination of search and problem reduction

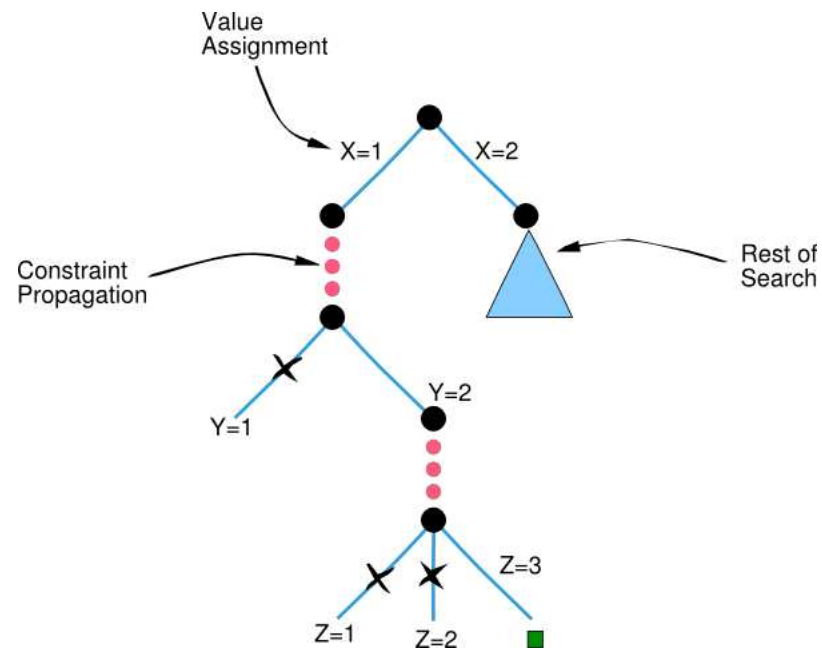


Figure 1: Interleaved problem reduction and search; (Only?) feasible method in practice

## Size and structure of search space

- Finite size:  $|D_{x_1}| \times \dots \times |D_{x_n}|$  leafs
- Depth is fixed ( $n$  is number of variables)
- No. nodes in general exponential but depends on variable order

# Three choices in search

## Next variable

**Static ordering** the order of the variables is specified before the search begins, and it is not changed thereafter

**Dynamic ordering** the choice of the next variable to be considered at any point in search depends on the current state

**Next value** or constraint strengthening (e.g. step up or binary search)

**Next constraint** to propagate (generally a fixed feature of CP system)

# Common heuristics to minimize search space

- First fail
  - Motivation
    - \* To succeed, try first where you are most likely to fail
  - Variable with least domain size
    - \* In tight satisfiability problem with sparse solution space or for optimiz.
    - \* Requires strong propagation for tight problems
  - Most constrained
    - \* Variables in many constraints or variables in tight constraints
    - \* How to measure constrainedness in presence of global constraints?
- For loose satisfiability problem with dense solution space the opposite heuristic may be used

# Constraint propagation (domain reduction)

- Necessary to avoid exponential search
- Immediate domain reduction — Enforce some form of consistency at each step in search e.g:
  - Arc-consistency
  - Path-consistency
- Delayed domain reduction — Use more sophisticated and expensive technique to prune search space at certain points in search e.g:
  - Forward checking (FC)
  - Full look ahead (MAC)

# Consistency techniques

$c$  consistent with  $C$

$$\Gamma \models (\exists) C \rightarrow (\exists) c \wedge C$$

$\chi$ -consistency

$$(\forall c \in C) (\exists) (c \chi C)$$

- Consistency techniques are generally incomplete
- Many consistency techniques work only on binary CSPs
- Consistency is *not* satisfiability

# Satisfaction and $k$ -satisfiability of a CSP

**Recall** the definition of satisfaction:

A compound label  $L$  *satisfies* a constraint  $C_{\langle x_1, \dots, x_k \rangle}$  if and only if there exists a projection  $S_k \subseteq L$  that is a  $k$ -compound label such that  $S \in C_{\langle x_1, \dots, x_k \rangle}$ .

A CSP  $\langle Z, D, C \rangle$  is  *$k$ -satisfiable* if and only if, for each subset  $\{x_1, \dots, x_k\} \subseteq Z$  of variables in  $Z$  there exists a  $k$ -compound label  $\{\langle x_1, v_1 \rangle, \dots, \langle x_k, v_k \rangle\}$  that satisfies *all* constraints  $C_S \in C$  for which  $S \subseteq \{x_1, \dots, x_k\}$ .

**Note** that a CSP  $\langle Z, D, C \rangle$  is satisfiable if and only if it is  $|Z|$ -satisfiable

**Note** that  $k$ -satisfiability of a CSP implies that it is also  $(k - 1)$ -satisfiable

## Node consistency

A variable  $x \in Z$  in a CSP  $\langle Z, D, C \rangle$  is 1-consistent (node consistent) if and only if all values  $v_i \in D_x$  satisfy the unary constraints  $C_{\langle x \rangle} \in C$

A CSP  $\langle Z, D, C \rangle$  is 1-consistent (node consistent) if and only if all its variables are 1-consistent

**Note** that 1-consistency does not imply 1-satisfiability since the domain of a variable in a CSP may be empty, in which case 1-consistency is trivial but the CSP is unsatisfiable

## A node consistency algorithm

---

**Algorithm 3** Node Consistency — Revise the domains of the variables in a constraint graph to ensure node consistency for CPS  $\langle Z, D, C \rangle$

---

```
procedure  $NC(\langle Z, D, C \rangle)$   
for all  $x \in Z$  do  
  for all  $v \in D_x$  do  
    if  $\exists C_{\langle x \rangle} \in C : \neg C_{\langle x \rangle}(v)$  then  
       $D_x \leftarrow D_x \setminus \{v\}$   
    end if  
  end for  
end for  
return  $\langle Z, D, C \rangle$ 
```

---

## $k$ -consistency

A CSP  $\langle Z, D, C \rangle$  is  $k$ -consistent if and only if

for each

$(k - 1)$ -compound label  $\{\langle x_1, v_1 \rangle, \dots, \langle x_{k-1}, v_{k-1} \rangle\}$  that satisfies all  $(k - 1)$ -ary constraints  $C_{\langle y_1, \dots, y_{k-1} \rangle} \in C$  for which  $\{y_1, \dots, y_{k-1}\} = \{x_1, \dots, x_{k-1}\}$  and additional variable  $x_k$  there exists a label  $\langle x_k, v_k \rangle$  such that  $\{\langle x_1, v_1 \rangle, \dots, \langle x_{k-1}, v_{k-1} \rangle, \langle x_k, v_k \rangle\}$  satisfies all  $k$ -ary constraints  $C_{\langle z_1, \dots, z_k \rangle} \in C$  for which  $\{z_1, \dots, z_k\} = \{x_1, \dots, x_k\}$

**Note** that  $k$ -consistency does *not* imply  $(k - 1)$ -consistency

## Strong $k$ -consistency and satisfiability

A CSP  $\langle Z, D, C \rangle$  is *strongly  $k$ -consistent* if it is  $j$ -consistent for all  $j \leq k$

- 1-satisfiability and strong  $k$ -consistency implies  $k$ -satisfiability

A CSP  $\langle Z, D, C \rangle$  is *satisfiable* if it is strongly  $|Z|$ -consistent and 1-satisfiable

- Not realistic to enforce during search
- Various weaker forms of consistency exist

## Arc-consistency of *binary* CSPs

An arc  $\langle x, y \rangle$  in the constraint graph of a *binary* CSP  $\langle Z, D, C \rangle$  and its corresponding constraint  $C_{\langle x, y \rangle}$  are *arc-consistent* if and only if for each value  $v_x \in D_x$  there exists at least one value  $v_y \in D_y$  for  $y$  such that  $\{\langle x, v_x \rangle, \langle y, v_y \rangle\}$  satisfies  $C_{\langle x, y \rangle}$ .

**Note** that the concept of arc-consistency in binary CSPs is directional, i.e., if an arc  $x \curvearrowright y$  is consistent, then it does *not* automatically mean that  $y \curvearrowright x$  is also consistent.

A CSP is *arc-consistent* if all its constraints are arc-consistent.

A *binary* CSP is *arc-consistent* if and only if it is strongly 2-consistent.

## Generalized arc-consistency

An hyper-arc  $\{x_1, \dots, x_k\} \subseteq Z$  in a constraint graph of a general (non-binary) CSP  $\langle Z, D, C \rangle$  and its corresponding constraint  $C_{\langle x_1, \dots, x_k \rangle}$  is *arc-consistent* if and only if, for each variable  $x_i$  and value  $v_{x_i} \in D_{x_i}$  there exists values  $v_{x_j} \in D_{x_j}$  for each  $j \leq k$  such that  $\{\langle x_1, v_{x_1} \rangle, \dots, \langle x_i, v_{x_i} \rangle, \dots, \langle x_k, v_{x_k} \rangle\}$  satisfies  $C_{\langle x, y \rangle}$ .

A general CSP is *arc-consistent* if all its constraints are arc-consistent.

**Note** that even if a general CSP is arc-consistent it is  $k$ -consistent only if e.g. all its constraints are  $k$ -ary

**Note** that propagation of global constraints in CP may still be incomplete, i.e. not achieve generalized arc-consistent

## Bounds (interval) consistency of $n$ -ary CSPs

An arc  $\{x_1, \dots, x_k\} \subseteq Z$  in a constraint graph of a CSP  $\langle Z, D, C \rangle$  and its corresponding constraint  $C_{\langle x_1, \dots, x_k \rangle}$  is *arc-B-consistent* if and only if, for each variable  $x_i$  and the bounds  $\underline{x}_i$  and  $\overline{x}_i$  on its domain  $D_{x_i} = [\underline{x}_i.. \overline{x}_i]$  there exists values  $v_{x_j}, v'_{x_j} \in D_{x_j}$  for each  $j \leq k$  such that both  $\{\langle x_1, v_{x_1} \rangle, \dots, \langle x_i, \underline{x}_i \rangle, \dots, \langle x_k, v_{x_k} \rangle\}$  and  $\{\langle x_1, v'_{x_1} \rangle, \dots, \langle x_i, \overline{x}_i \rangle, \dots, \langle x_k, v'_{x_k} \rangle\}$  satisfy  $C_{\langle x_1, \dots, x_k \rangle}$ .

A CSP is *arc-B-consistent* if all its constraints are arc-consistent.

- Bounds, interval or Arc-B-consistency can be defined on CSPs that have domains that can be represented as numerical intervals  $[a..b]$
- This is a weaker property than arc-consistency but cheaper to maintain and commonly used in practical CP systems

## Example of interval arithmetics

$$x = [\underline{x}, \bar{x}], y = [\underline{y}, \bar{y}]$$

$$x + y = [\underline{x} + \underline{y}, \bar{x} + \bar{y}], \quad (1)$$

$$x - y = [\underline{x} - \bar{y}, \bar{x} - \underline{y}], \quad (2)$$

$$x \times y = [\min(\underline{x} \cdot \underline{y}, \underline{x} \cdot \bar{y}, \bar{x} \cdot \underline{y}, \bar{x} \cdot \bar{y}), \max(\underline{x} \cdot \underline{y}, \underline{x} \cdot \bar{y}, \bar{x} \cdot \underline{y}, \bar{x} \cdot \bar{y})], \quad (3)$$

$$1/x = [1/\bar{x}, 1/\underline{x}] \quad \text{if } \underline{x} > 0 \text{ or } \bar{x} < 0 \quad (4)$$

$$x \div y = x \times (1/y) \quad (5)$$