CSP Search Techniques

1. Backtracking
2. Forward checking
3. Partially Look Ahead
4. Fully look Ahead
5. Back Checking
6. Back Marking
7. Modified Forward checking
1. Backtracking
2. Forward Checking
3. Partially Look Ahead

![Diagram of partially lookahead strategy]

- Q
- FC FC PL Q
- FC PL FC
- FC FC

- BT Q
- FC FC FC Q
- FC FC FC
- FC FC

- BT Q
- FC FC FC Q
- Q FC FC FC
- FC FC FC

- BT Q
- FC FC FC Q
- Q FC FC FC
- FC FC Q FC

- BT Q
- FC FC FC Q
- Q FC FC FC
- FC FC Q FC
4. Fully Look Ahead
Rest of CLP Search Techniques

5. Back Checking:
   Remembering Previous failures

6. Back Marking:
   Remembering Previous failures and successes

7. Modified Forward Checking:
   Representing constraints as bit patterns and using AND/OR operations to test the patterns
K-Satisfiability

A CLP Problem with n variables is $K$-satisfiable, (k $\leq$ n), if for every subset of K variables, there exists a k-label (values for k variables) that satisfies all the problem constraints.

If a problem is k-satisfiable, then it is k-1 satisfiable, too.
Consistency

Node consistency
- A node $X$ is node-consistent if every value in the domain of $X$ is consistent with $X$’s unary constraints.
- A graph is node-consistent if all nodes are node-consistent.

Arc consistency
- An arc $(X, Y)$ is arc-consistent if, for every value $x$ of $X$, there is a value $y$ for $Y$ that satisfies the constraint represented by the arc.
- A graph is arc-consistent if all arcs are arc-consistent.

To create arc consistency, we perform constraint propagation: that is, we repeatedly reduce the domain of each variable to be consistent with its arcs.
Arc Consistency

A$\rightarrow$B is consistent if for each remaining value in domain of A, there may be a consistent value in domain of B.

- Consistent:
  • SA$\rightarrow$NSW, NSW$\rightarrow$V,…

- Not Consistent:
  • V$\rightarrow$NSW, NT$\rightarrow$SA,…
Arc Consistency Checking Algorithm (AC-3)

function AC-3( csp ) returns the CSP, possibly with reduced domains
inputs: csp, a binary CSP with variables \( \{X_1, X_2, \ldots, X_n\} \)
local variables: queue, a queue of arcs, initially all the arcs in csp

while queue is not empty do
    \( (X_i, X_j) \leftarrow \text{REMOVE-FIRST}(queue) \)
    if RM-INCONSISTENT-VALUES(\( X_i, X_j \)) then
        for each \( X_k \) in NEIGHBORS[\( X_i \)] do
            add (\( X_k, X_i \)) to queue

function RM-INCONSISTENT-VALUES( \( X_i, X_j \) ) returns true iff remove a value
removed \( \leftarrow \) false
for each \( x \) in DOMAIN[\( X_i \)] do
    if no value \( y \) in DOMAIN[\( X_j \)] allows \( (x,y) \) to satisfy constraint(\( X_i, X_j \))
        then delete \( x \) from DOMAIN[\( X_i \)]; removed \( \leftarrow \) true
return removed
Arc consistency

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$
Arc consistency

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$
Arc consistency

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$

If $X$ loses a value, neighbors of $X$ need to be rechecked
Arc consistency

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$

If $X$ loses a value, neighbors of $X$ need to be rechecked

Arc consistency detects failure earlier than forward checking

[Diagram of arc consistency process]
K-consistency

K-consistency generalizes the notion of arc consistency to sets of more than two variables.

- A graph is K-consistent if, for legal values of any K-1 variables in the graph, and for any Kth variable $V_k$, there is a legal value for $V_k$

**Strong K-consistency = J-consistency for all J<=K**

**Node consistency = strong 1-consistency**

**Arc consistency = strong 2-consistency**

**Path consistency = strong 3-consistency**
Width of a constraint graph

• We can have different orderings of a constraint graph
• The number of backward arcs of a node, in an specific ordering, is called the node’s width
• An ordering width is the maximum width of its nodes
• A graph width is the minimum width of its different orderings
Why do we care?

If the width of the constraint graph of a CSP is D and it is **strongly K-consistent**, then if $K > D$, we can solve the CSP without backtracking, if we use an appropriate variable ordering (i.e., one with Minimal width ordering)
Intelligent Backtracking

Q $\leftarrow$ Red
NSW $\leftarrow$ Green
V $\leftarrow$ Blue
T $\leftarrow$ Red
SA $\leftarrow$ ?

Chronological Backtracking
Back Jumping

Q ← Red
NSW ← Green
V ← Blue
T ← Red
SA ← ?
Conflict Set of SA: {Q, NSW, V}

**NOTE:** Back Jumping doesn’t help Forward Checking.