G53CLP
Constraint Logic Programming

Dr Rong Qu
Basic Search Strategies
CP Techniques

- Constraint propagation*
  - ... 

- Basic search strategies
  - Look back, look ahead ...
  - Search orders
  - B & B

*So far what we’ve seen is made upon the CSP itself before the search
Search Tree + Backtracking

- CSP search tree
  - Depth first search vs. breadth first search*
  - Complexity*
    - time and memory
  - CSP of size $n$
    - All solutions will be at level $n$
    - Open nodes during the search

*more details of “Blind Searches” in G51IAI at http://www.cs.nott.ac.uk/~rxq/g51iai.htm
Search Tree + Backtracking

- General search strategy
- CSP features could guide for more efficient search
Basic Search Strategies

- Look back
  - BackJumping
  - Back Checking
  - Back Marking

- Look ahead
  - Forward Checking
  - Look Ahead
Look Back Strategies

- Learn information while searching
  - In search tree of a CSP there are similar subtrees
  - Learn from the failure of searching
  - Avoid repeating the same mistakes
  - Go back up more levels
- Gather-information-while-searching
- We assume the order of variables is fixed
Look Back - BackJumping

- Pick a variable and assign a value compatible with previous ones

- Analyse the situation to find the source of inconsistency

- **Jump** (backtrack) to the most recent conflicting variable rather than the immediate previous variable
Assume node $x$ has domain \{red, green\}.

Jump back to the most recent conflicted variable.
Look Back - BackJumping

- BackJumping is similar to backtracking except the level backtracked
- Backtracking is a simple look back
- BackJumping jumps to relevant decisions only
- Identify the cause when failures occur
In some applications, the compatibility check are computational expensive
- Backchecking attempts to reduce the number of compatibility checks

- Remembers inconsistent pairs of assignment
  - Any of these pairs of assignment will not be considered
  - Computations on them are thus avoided
Look Back - BackChecking

\[ x_1, x_2, \ldots x_i, \ldots x_k, \ldots, x_n \]

- \( x_i = a \): \( x_i \) is labelled
- \( x_k = b \) is not compatible with \( x_i = a \)

- As long as \( x_i \) is labelled \( a \)
  - \( b \) will never be tried for labelling \( x_k \)
Look Back - BackChecking

\[ x_1, x_2, \ldots, x_i, \ldots, x_k, \ldots, x_n \]

- When \( x_i = a \), BackChecking doesn’t consider \( b \) while labelling \( x_k \)
  - Intelligent?
Look Back - BackMarking

- Further improvement of backchecking
  - Remember
    - Not only inconsistent labels which are already committed to
    - But also successful assignments
      - avoid repeated compatibility checking
Look Back - BackMarking

\[ x_1, x_2, \ldots x_i, \ldots x_{k-1}, x_k, \ldots, x_n \]

- \( x_k \) is being labelled, but failed
- Backtracking to \( x_i \), reassign value to \( x_i \)
- Label all variables between \( x_i \) and \( x_{k-1} \)

Labelling of \( x_k \)
- not checking values which are incompatible with labels for \( x_1 \) to \( x_{k-1} \) (like backchecking)
- Furthermore, not checking any value against labels of \( x_1, \ldots x_i \): they were successful
Look Back vs. Look Ahead

- Look back
  - Based on backtracking
  - Remember failures
  - Late detection of inconsistency

- Look ahead
  - Check variables not yet labelled
  - Constraint propagation on future variables
  - Prevent future inconsistency on constraints at earlier stage
Look Ahead - Forward Checking

- Forward Checking

  - Check variables not yet labelled which are constrained with the current variable

  - Any values in these variables conflict with the current value is (temporally) removed
Look Ahead - Forward Checking

Forward Checking

- If the domain of any of these variables is empty
  - Try different value to the current variable

- If all values are tried for the current variable
  - Backtrack to previous variable
Look Ahead - Forward Checking
Look Ahead - Forward Checking

- Forward checking performs arc-consistency
  - There exists at least one valid value for the uninitiated variable

- Forward checking allows branches to be cut
  - Branches leading to dead-ends are removed immediately
  - Search tree is smaller
  - Quicker/efficient search
  - Less number of backtracking
Look Ahead - A Small Exercise

- N-queen problem
  - $n$ queens
  - A $n \times n$ chessboard
  - Put $n$ queens on the board so that no two queens attack each other
    - Vertically
    - Horizontally
    - Diagonally
Look Ahead - A Small Exercise

- The 8-queen problem

- You have 2 minutes!
Backchecking vs. Forward Checking

**Backchecking**
- Once $y=b$ is found incompatible with $x=a$
- As long as $x=a$, $b$ is not considered later when $y$ is being labelled

**Forward checking**
- Check if there exists at least one value for the unlabelled variables when labelling $x$ with $a$
- $b$, if incompatible with $x=a$, is temporarily removed from $y$ once $x$ is labelled
Backchecking vs. Forward Checking

In both, values which are incompatible are rejected from the domain

- **Forward checking**
  - When \( <x, a> \), value \( b \) removed from \( y \)
  - Rejected when \( x \) is labelled

- **Backchecking**
  - \( b \) remembered when labelling \( <x, a> \)
  - Rejected when \( y \) is being labelled

Backchecking is inferior to forward checking
Example - Backchecking

- Example: map coloring \{red, green, blue\}

Value blue taken directly
Example – Forward Checking

- Example: map coloring \{red, green, blue\}
Other Basic Search Strategies

- Look back
  - Dependency directed backtracking
  - Learning nogood compound labels

- Look ahead
  - Directional AC-lookahead
  - AC-lookahead
Summary

- CP Techniques
  - Constraint propagation
    - Consistency check
  - Search strategies
    - Look back: backjumping, backchecking, backmarking
    - Look ahead: forward checking
    - Search orders
    - B&B