Search Strategies: Lookahead

AIMA 6.3 (6.3.3 excluded),
Constraint Processing, R. Dechter
Sections 5.1, 5.3 (5.3.2 excluded)

Summary

- Introduction and Consistency Levels
- Backtracking
- Look-Ahead

Approximate Inference and Search

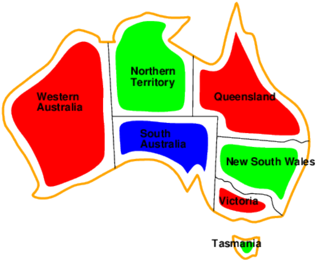
Need to take chances

- Complete inference (e.g., strong n-consistency) ensures no dead-end in extending partial solutions to complete solutions
- However, strong i-consistency is exponential (in the number of variables) $\rightarrow$ not practical
- Approximate Inference is polynomial but we still need to search for a solution
- search: proceed by trial and errors

Running Example: Map-Coloring

Variables WA, NT, Q, NSW, V, SA, T
Domains $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors
e.g., $WA \neq NT$ (if the language allows this), or
$(WA, NT) \in$
$\{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

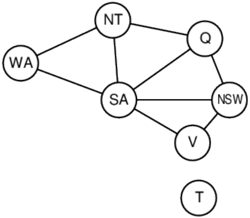
Example: Map-Coloring contd.



Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red,$
$NSW = green, V = red, SA = blue, T = green\}$

Constraint graph

Binary CSP: each constraint relates at most two variables



Standard search formulation for CP

Naive choice

States are defined by the values assigned so far

♢ Initial state: the empty assignment, $\{\}$

♢ Successor function: assign a value to an unassigned variable that does not conflict with current assignment.

$\Longrightarrow$ fail if no legal assignments (not fixable!)

♢ Goal test: the current assignment is complete

1. This is the same for all CSPs!
2. Every solution appears at depth n with n variables
   $\Longrightarrow$ use depth-first search
3. Path is irrelevant, so can also use complete-state formulation
4. $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!!!!

Backtracking search

A better formulation

Variable assignments are commutative, i.e.,
$[WA = red$ then $NT = green]$ same as $[NT = green$ then $WA = red]$
Only need to consider assignments to a single variable at each node
$\Longrightarrow$ $b = d$ and there are d^n leaves

♢ Depth-first search for CSPs with single-variable assignments is called backtracking search

♢ Backtracking search is the basic uninformed algorithm for CSPs. Can solve n-queens for $n \approx 25$

♢ Variable ordering counts for performance

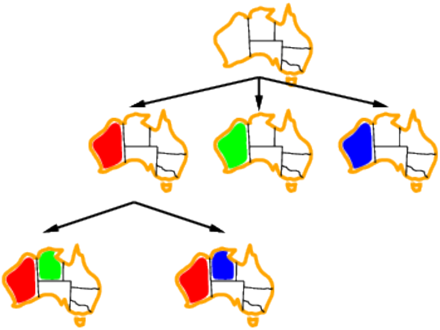
Backtracking search


```
function Backtracking-Search(csp) returns solution or failure
    return Backtrack({ }, csp)
function Backtrack(assignment, csp) returns solution or failure
    if assignment is complete then return assignment
    var ← Select-Unassigned-Variable(csp)
    for each value in Order-Domain-Values(var, assignment, csp) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← Inferences(csp, var, value)
            if inferences ≠ failure then
                add inferences to assignment
                result ← Backtracking(assignment, csp)
                if result ≠ failure then
                    return result
                endif
            endif
        endif
    remove {var = value} and inferences from assignment
    endfor
return failure
```

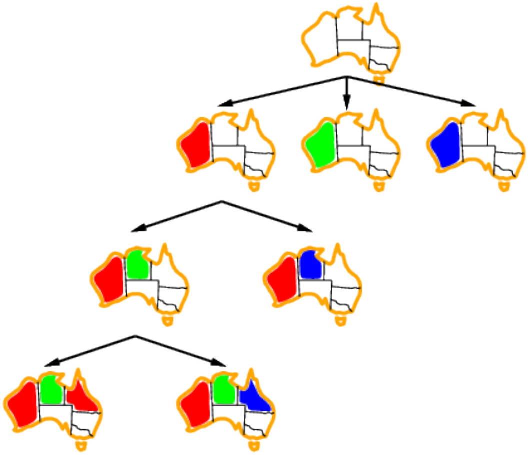
Backtracking example

Backtracking example

Backtracking example



Backtracking example



Improving Backtracking

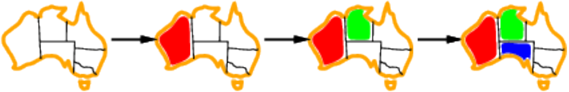
◇ General Goal: Reducing size of explored search space

Ingredients

- ordering variable and variables' values
- local consistency (e.g., arc or path consistency)
- look-ahead, predict future inconsistencies
- (look-back, where to backtrack)
- tree decomposition, exploit problem structure

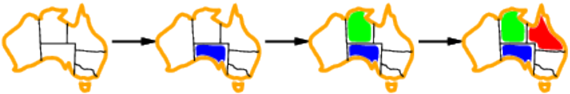
Ordering Variables: Minimum remaining values

Minimum remaining values (MRV):
choose the variable with the fewest legal values



Ordering Variables: Degree heuristic

Tie-breaker among MRV variables
Degree heuristic:
choose the variable with the most constraints on remaining variables



MRV and DH can be applied to any CSP

Variable Orderings: Example

Exercise: Dividing Integer

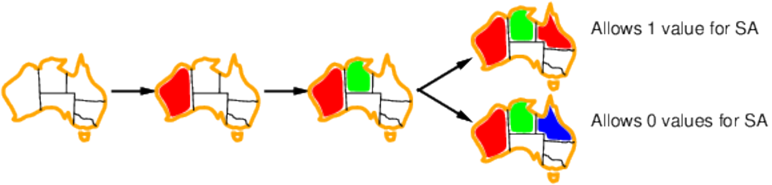
- Consider the following network $\mathcal{R}$
- Variables: $x, y, l, z,$
- Domains:
  $D_x = D_y = \{2, 3, 4\}, D_l = \{2, 5, 6\}, D_z = \{2, 3, 5\}$
- Constraints: z divides evenly x, y, l

Compute number of expanded nodes for assigning variable with different orderings:

- $d_1 = \{x, y, l, z\}$
- Use Minimum Remaining Values and degree heuristic to choose next value

Value ordering: Least constraining value

Given a variable, choose the least constraining value:
the one that rules out the fewest values in the remaining variables



Combining these heuristics makes 1000 queens feasible
Gains depend on the specific problem

Consistency Level and Search Space

Good effects on Search Space Size

- Tighter constraints $\rightarrow$ smaller search space
- Given two equivalent network $\mathcal{R}$ and $\mathcal{R}'$
- if $\mathcal{R}' \subseteq \mathcal{R}$ then any solution path appearing in the search space of $\mathcal{R}'$ also appears in the search space of $\mathcal{R}$, for any ordering d.
- Higher level of consistency reduce the search space

Consistency Level and Search process

Negative effects on Searching

- Adding constraints requires more computation
- Each time a new variable is assigned need to check many more constraints
- If only binary constraints we never have more than $O(n)$ checks
- If r-ary constraints then we could have $O(n^{r-1})$ checks

Backtrack Free Search

Backtrack Free Network

- A network $\mathcal{R}$ is backtrack free if every leaf is a goal state
- A DFS on a backtrack free network ensure a complete consistent assignment
- E.g. $\mathcal{R}$ + arc consistency + $\{z, x, y, l\} \rightarrow$ backtrack free network

Look-Ahead

Look-Ahead Schemes

- Given approximate inference (arc consistency, path-consistency)
- Foresee impact of next move (which variable, which value)
- Impact: how next move restricts future assignment
- Efficient way to update information for choosing next variable/value
  - Can efficiently compute remaining legal values given current assignment

Look-Ahead Strategies

Strategies

- Forward Checking
  - check unassigned variables separately
- Arc consistency look-ahead
  - propagate arc consistency

Look-ahead: Discussion

Discussion

- Incur extra cost for assigning values
  - need to propagate constraints
- Can restrict search space significantly
  - e.g., discover that a value makes a sub-problem inconsistent
  - remove values from future variables' domains
- Usually no changes on worst case performance: trade-off between costs and benefits

Generalised Look-ahead

Algorithm

Algorithm 1 Generalised Look-ahead

Require: A constraint network $\mathcal{R}$
Ensure: A solution or notification that the network is inconsistent

```
i ← 1
D'_i ← D_i
while 1 ≤ i ≤ n do
    x_i ← SelectValueX
    if x_i is null then
        i ← i − 1
        Reset D'_k for each k > i to its value before i was last instantiated
    else
        i ← i + 1
    end if
end while
if i is 0 then
    return inconsistent
else
    return instantiated valuse for {x_1, ··· , x_n}

end if
```

Forward Checking

Forward Checking

- most limited form of constraint propagation
- propagates the effect of a selected value to future variables separately
- if domains of one of future variables becomes empty, try next value for current variable.

Select Value Forward Checking

Algorithm

Algorithm 2 SelectValueForwardChecking

```
a ← D'_i select an arbitrary value
while D'_i ≠ { } do
    for all k, i < k ≤ n do
        for all b, b ∈ D'_k do
            if < ā_{i−1}, x_i = a, x_k = b > is not consistent then
                D'_k ← D'_k \ {b}
            end if
        end for
        if D'_k = { } then
            emptyDomain ← true
        end if
    end for
    if emptyDomain then
        reset each D'_k to its value before assigning a
    else
        return a
    end if
end while

return null
```

Forward checking

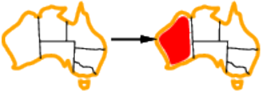
Idea: Keep track of remaining legal values for unassigned variables

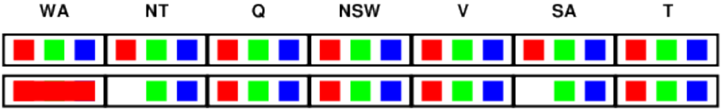
Terminate search when any variable has no legal values

Forward checking

Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values

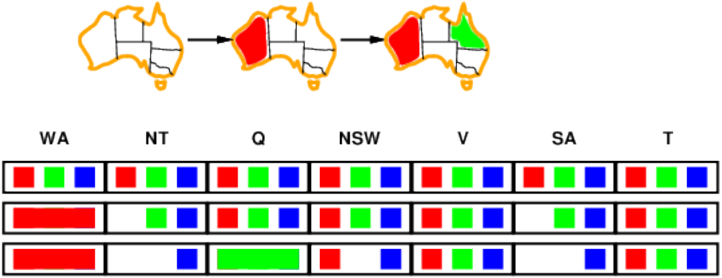




Forward checking

Idea: Keep track of remaining legal values for unassigned variables

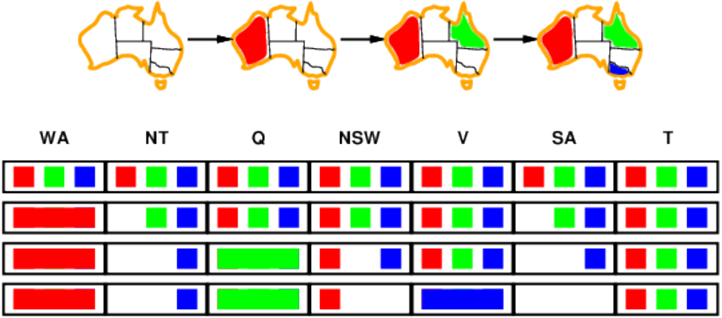
Terminate search when any variable has no legal values



Forward checking

Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values



Forward Checking: Example

Example (Graph Colouring Example)

- Variables: $x_1, x_2, x_3, x_4, x_5, x_6, x_7,$
- Domains: $D_{x_1} = \{R, B, G\}, D_{x_2} = D_{x_5} = \{B, G\}, D_{x_3} = D_{x_4} = D_{x_7} = \{R, B\}, D_{x_6} = \{R, G, Y\}$
- Constraints: $x_1! = x_2, x_1! = x_3, x_1! = x_4, x_1! = x_7, x_2! = x_6, x_3! = x_7, x_4! = x_5, x_4! = x_7, x_5! = x_6, x_5! = x_7$
- $x_1 = \textit{red}$ reduces domains of x_3, x_4, x_7
- $x_2 = \textit{blue}$ no effects
- $x_3 = \textit{blue}$ (only available) makes x_7 empty $\rightarrow$ x_3 dead-end

Complexity of Forward Checking

Complexity of Select Value Forward Checking

- $O(ek^2)$
- e_u consistency check for each value of each future variable x_u
- k value for each future variables $O(e_u k)$
- $\sum_u e_u = e$ then $O(ek)$
- k value for the current variable

Arc Consistency Look-Ahead

Arc Consistency Look ahead

- force full arc consistency on all remaining variables
- select a value for current variable $x_i = a$
- apply $AC - 1$ on all unassigned variables with $x_i = a$
- If a variable domain becomes empty reject current assignment
- can use $AC - 3$ or $AC - 4$ instead

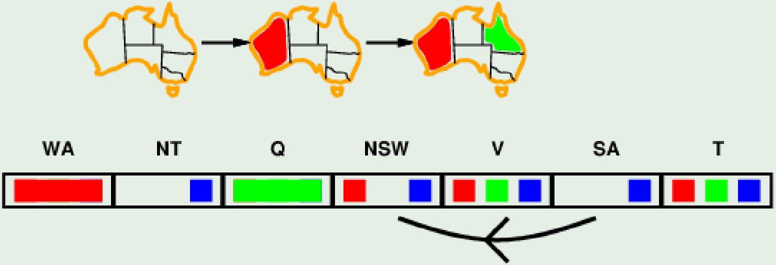
Arc Consistency Look-Ahead Complexity

Arc Consistency Look Ahead

- Best algorithm for AC is $AC - 4$ complexity $O(ek^2)$
- worst case for Select Arc Consistency look-ahead is $O(ek^3)$

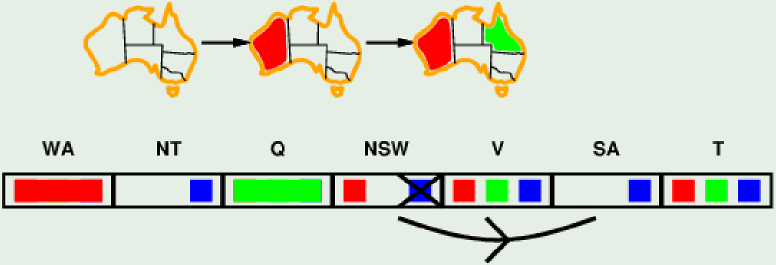
Example of AC Look-Ahead

Example (AC Look-Ahead for Map Colouring)



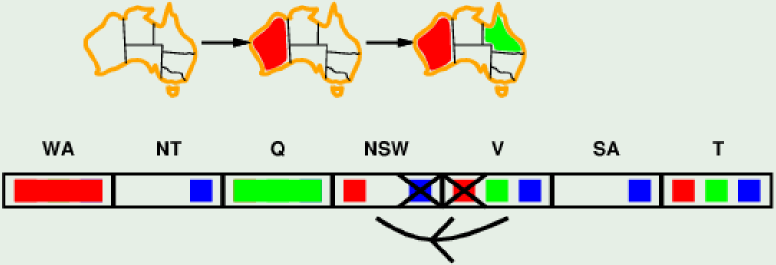
Example of AC Look-Ahead

Example (AC Look-Ahead for Map Colouring)



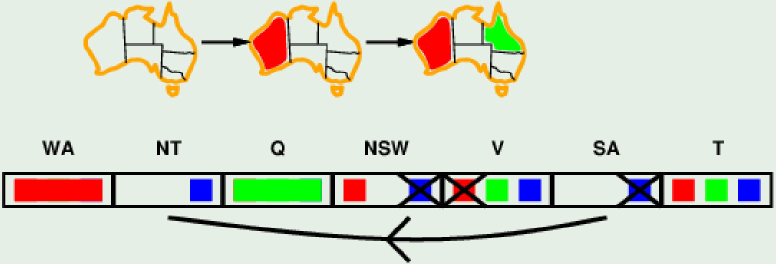
Example of AC Look-Ahead

Example (AC Look-Ahead for Map Colouring)



Example of AC Look-Ahead

Example (AC Look-Ahead for Map Colouring)



Mantaining Arc Consistency

MAC - variant of Arc Consistency Look-Ahead

- Apply Full Arc Consistency each time a value is rejected
- if empty domain $\rightarrow$ no solutions
- otherwise continue backtracking with another variable

Example

- Consider variable x_1 with $D_1 = 1, 2, 3, 4$
- Apply Backtracking with AC look ahead
- Suppose value 1 is rejected: apply full AC with $D_1 = 2, 3, 4$
  - if empty domain $\rightarrow$ stop
  - else value selection with next variable
- search in a binary **virtual** tree $x_1 = 1$ or $x_1 \neq 1$

Exploiting problem structure in Look ahead

Definition (Cycle Cutset)

Given an undirected graph, a subset of nodes in the graph is a cycle cutset iff its removal result in an acyclic graph

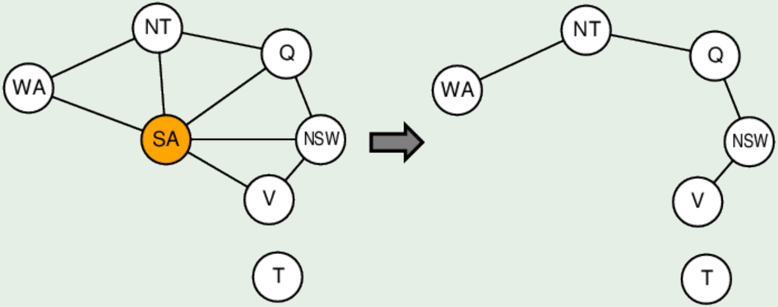
Exploiting problem structure

- Once a variable is assigned it can be removed from the graph (conditioning)
- If we remove a cycle-cutset the rest of the problem is a tree
- Can use arc consistency to solve that sub-problem
- We need to check all possible assignments of cycle-cutset variables and do arc propagation
- Complexity is still exponential but in the size of the cycle-cutset!

Exploiting problem structure: example

Example (Cycle Cut Set for Map Colouring)

Conditioning: instantiate a variable, prune its neighbors' domains



Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size $c \implies$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small c

Exercise: AC Look-Ahead

AC Look-Ahead for Map Colouring

Use the AC-3 Algorithm to show that AC Look-Ahead detects an inconsistency on the partial assignment $\{WA = red, V = blue\}$ for the Australia map colouring problem used above.

Exercise: Cycle cut set

Cycle cut set for Graph Colouring

Use the Cycle cut set algorithm to solve the graph colouring problem defined above.