Search Strategies: Lookahead

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 numnber of variables) $\rightarrow$ not practical
- Approximate Inference is polynomial but we still need to search for a solution
- search: proceed by trial and errors

Search

Search for CP

- State: partial variable assignment
- Goal State: consistent complete allocation
- Move: assign one (or more) variable(s)
- Good Moves: assignments that go closer to the Goal

Backtracking Search

Backtrack

- Decide whether a state is closer to the goal is very hard
- Try promising moves
- Dead ends: backtrack changing previous assignments
- Halt: when a solution is found or all possible solution where searched
- Worst case: exponential in the number of variables

Improving Backtrack

Improvements

- Reducing size of explored search space
- before the search, preprocess the problem
  - variable orderings
  - forcing consistency (e.g., arc or path consistency)
- during the search, search strategies
  - look-ahead, which is the best next move
  - look-back, where to backtrack

State Space

Basic Concept

- A set S of states
  - consistent partial variable instantiations
- A set O of operators, $O : S \to S$
  - extension of partial instantiation to another variable
- An initial state s_0
  - the empty assignment
- A set of goal states $S_g \subseteq S$
  - a complete consistent assignment
- A terminal state is a state from which we can not reach any other state
  - any complete assignment

State Space and Orderings: Example

Example (Dividing Integer Example)

- 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 search space for assigning variable with different orderings:

- $d_1 = \{z, x, y, l\}$
- $d_2 = \{x, y, l, z\}$

Variable Ordering and Search Space

Effect on Search Space Size

- $d_1 \rightarrow$ 20 legal states
- $d_2 \rightarrow$ 48 legal states
- Search space includes all solutions
- The less dead end the better

State Space and Consistency: Example

Example (Dividing Integer Example)

- Force arc consistency
- Draw search space for d_1
- Force path consistency
- Draw search space for d_2

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 + $d_1 \rightarrow$ backtrack free network

Backtracking

Basic Ideas

- Traverses the search space with a DFS
- Two phases:
  - Forward phase: extend partial solutions by assigning a consistent value if one exists
  - Backward phase: if no further extension is possible return to the previous variable assigned

Backtracking Example

Example (Graph colouring)

- Variables: x_1, x_2, x_3
- Domains: $D_1 = D_2 = \{R, B\}$ $D_3 = \{R, B, Y\}$
- Fixed Ordering: $\{x_3, x_1, x_2\}$
- Find one solution
- Find all solutions

Backtracking Procedure

Algorithm

Algorithm 1 Backtracking

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 ← SelectValue
    if x_i is null then
        i ← i − 1
    else
        i ← i + 1
        D'_i ← D_i
    end if
end while
if i is 0 then
    return inconsistent
else
    return instantiated valuse for {x_1, ··· , x_n}
end if
```

Select Values Procedure

Select Value Algorithm

Algorithm 2 SelectValue

Require: A partial assignment $\bar{a}_{i-1}$

Ensure: A value in D'_i consistent with $\bar{a}_{i-1}$ or null

while $D'_i \neq \{\}$ **do**

  v a value in D'_i

  $D'_i \leftarrow D'_i \setminus v$

  if $< \bar{a}_{i-1}, x_i = v >$ is consistent **then**

    return v

  end if

end while

return null

Complexity of Backtracking

Complexity

- Complexity of extending a partial solution:
  - Complexity of consistent $O(elogt)$
  - Complexity of SelectValue $O(eklogt)$
- t bounds tuple, e constraints, k values

Improvements for Backtracking

Improving Backtracking

- Before Search
  - Forcing Consistency
  - Fixing variable ordering
- During Search
  - Look Ahead (Forward phase)
    - Value Ordering
    - Variable Ordering
  - Look Back (Backward phase)
    - Backjumping
    - Constraint Recording

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
- Which Variable:
  - if order not pre-defined
  - instantiate variable that constraint the most the search space
  - e.g., most constrained variable with least possible assignments
- Which Value
  - value that maximises possible future assignments

Look-Ahead Strategies

Strategies

- Forward Checking
  - check unassigned variables separately
- Maintaining arc consistency
  - propagate arc consistency
- Full look ahead
  - one pass of arc consistency

Look-ahead: Discussion

Discussion

- Incur extra cost for assigning values
  - need to propagate constraints
- Can resctrict 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 cost and benefit

Generalised Look-ahead

Algorithm

Algorithm 3 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 4 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: 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 Forward Checking

- $O(ek^2)$ for each node
- 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 variales
- select a value for current variable $x_i = a$
- apply $AC-1$ on all variable $k > i$ 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 $O(eK^3)$

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 with backtracking

Example

- Given a network $\mathcal{R}$
- 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 $D_1 = 2, 3, 4$

Full Look Ahead

Approximation of Arc Consistency Look Ahead

- Same as Arc Consistency Look ahead
- Perform only one pass of AC (no repeat untill)
- More work than forward checking less than Arc Consistency Look-ahead

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
- 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 assignment of cycle-cutset variables and do arc propagation
- Complexity is still exponential but in the size of the cycle-cutset!

Look Ahead for SAT

DPLL

- Backtracking can be applied to SAT for CNF
- DPLL is a specific backtracking algorithm for SAT
- Uses a CNF-specific look-ahead method: unit propagation
- Plus heuristics to choose next variable to expand

Boolean Constraint Propagation: Example

Example (Boolean constraint propagation)

- $\phi = A \vee \neg B, B$, $\mathcal{R}_\phi$ nework representing ϕ
- force arc consistency to $\mathcal{R}_\phi$
- $\psi = A \vee \neg B, B \vee C$, $\mathcal{R}_\psi$ nework representing ψ
- force path consistency to $\mathcal{R}_\psi$

Boolean Constraint Propagation

Constraint Propagation for CNF

- Domain restiction = unit clause
- Arc-consistency = to unit resolution
- Path consistency = to resolution between clauses of length 2

Unit Propagation

Algorithm

Algorithm 5 UnitPropagate(ϕ)

Require: A CNF formula ϕ
Ensure: An equivalent formula with unit clause removed

```
Q ← all unit clauses in φ
while Q ≠ { } do
    T ← one unit clause in Q
    for all clause β in φ containing T or ¬T do
        if T ∈ β then
            delete β
        else
            γ ← Resolve(β, γ)
            if γ is the empty clause then
                return theory unsatisfiable
            else
                add γ to φ and delete β
                if γ is a unit clause then
                    add γ to Q
                end if
            end if
        end if
    end for

end while
```

Unit Propagation: discussion

Characteristics

- force arc consistency
- arc consistency for general constraints (not-only binary)
- perform arc consistency in linear time:
  - each step we either eliminate a clause or eliminate a literal
  - number of unit resolution is at most the length of the CNF formula

DPLL as backtracking for CNF

DPLL algorithm

Algorithm 6 DPLL(ϕ)

Require: A CNF formula ϕ
Ensure: A decision on whether ϕ is satisfiable

```
UnitPropagate(φ)
if empty clause is generated then
    return false
else
    if All variables are assigned then
        return true
    else
        Q ← one unassigned variable
        return (DPLL(φ ∧ Q) ∨ DPLL(φ ∧ ¬Q))
    end if
end if
```

DPLL: discussion

DPLL as backtrack for DCSP

- backtracking with arc consistency
- unit propagation forces arc consistency at each node
- we can force higher level of consistency
  - path consistency by applying resolution to clauses of lenght two
- heuristics to choose next variables
  - choose the one that causes the most unit clauses to appear
  - approximated by the number of 2-literals clauses in which the variable appears

Example: DPLL

Example (CNF with DPLL)

Consider the formula
$\phi = \{(\neg A \vee B), (\neg C \vee A), (A \vee B \vee D), (C)\}$