Assignment

• Remember:
  • Problem Set #6 Propositional Logic, due Today.
  • 16:413 Project Part 1: Sat-based Activity Planner, due Wednesday, November 3rd.
  • Problem Set #7 Diagnosis, Conflict-directed A* and RRTs, due Wednesday, November 10th.

• Reading
Model-based Diagnosis as Conflict-directed Best First Search

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

- Sherlock Holmes. The Sign of the Four.

1. Generate most likely Hypothesis.
2. Test Hypothesis.
3. If Inconsistent, learn reason for inconsistency (a Conflict).
4. Use conflicts to leap over similarly infeasible options to next best hypothesis.

Compare Most Likely Hypothesis to Observations

Flow$_1$ = zero
Pressure$_1$ = nominal

It is most likely that all components are okay.
Isolate Conflicting Information

The red component modes *conflict* with the model and observations.

Leap to the Next Most Likely Hypothesis that Resolves the Conflict

The next hypothesis must remove the conflict.
New Hypothesis Exposes Additional Conflicts

Another conflict, try removing both.

Final Hypothesis Resolves all Conflicts

Implementation: Conflict-directed A* search.
Outline

• Model-based Diagnosis
• Optimal CSPs
• Informed Search
• Conflict-directed A*

Constraint Satisfaction Problem

CSP = <X, D_X, C>
– variables X with domain D_X.
– Constraint C(X): D_X \rightarrow \{True, False\}.

Problem: Find X in D_X s.t. C(X) is True.
Optimal CSP

OCSP= <Y, g, CSP>

– Decision variables Y with domain D_Y.
– Utility function g(Y): D_Y → ℜ.
– CSP over variables <X;Y>.

Find leading arg max g(Y)
Y ∈ D_y
s.t. ∃ X ∈ D_X s.t. C(X,Y) is True.

→ g: multi-attribute utility with preferential independence, value constraint, …
→ CSP: propositional state logic, simple temporal problem, mixed logic-linear program, …

CSPs Are Frequently Encoded in Propositional State Logic

(mode(E1) = ok implies
(thrust(E1) = on if and only if flow(V1) = on and flow(V2) = on)) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)
Multi Attribute Utility Functions

\[ g(Y) = G(g_1(y_1), g_2(y_2), \ldots) \]

where
\[ G(u_1, u_2 \ldots u_n) = G(u_1, G(u_2 \ldots u_n)) \]
\[ G(u_1) = G(u_1, I_G) \]

Example: Diagnosis
\[ g_i(y_i = \text{mode}_{ij}) = P(y_i = \text{mode}_{ij}) \]
\[ G(u_1, u_2) = u_1 \times u_2 \]
\[ I_G = 1 \]

Mutual Preferential Independence (MPI)

Assignment \( \delta_1 \) is preferred over \( \delta_2 \) if \( g(\delta_1) < g(\delta_2) \).

For any set of decision variables \( W \subseteq Y \), our preference between two assignments to \( W \) is independent of the assignment to the remaining variables \( W - Y \).
MPI Example: Diagnosis

If $A_1 = G$ is more likely than $A_1 = U$,

then

\[
\{A_1 = G, A_2 = G, A_3 = U, X_1 = G, X_2 = G\}
\]

is preferred to

\[
\{A_1 = U, A_2 = G, A_3 = U, X_1 = G, X_2 = G\}.
\]

Outline

• Model-based Diagnosis
• Optimal CSPs
• Informed Search
  – $A^*$
  – Branch and Bound
• Conflict-directed $A^*$
Informed Search

Extend search tree nodes to include path length \( g \)

Problem:
- Given graph \( \langle V,E \rangle \) with weight function \( w: E \rightarrow \mathbb{R} \), and vertices \( S, G \) in \( V \),
- Find a path \( S \rightarrow P G \) with the shortest path length \( \delta(S,G) \) where
  * \( w(p) = \sum w(v_{i-1}, v_i) \).
  * \( \delta(u,v) = \min \{ w(p) : u \rightarrow P v \} \)
A* biases uniform cost towards the goal by using $h$.

- $f = g + h$
- $g =$ distance from start.
- $h =$ estimated distance to goal.

A* finds an optimal solution if $h$ never over estimates. Then $h$ is called “admissible”.

Greedy goes for the goal, but forgets its past.

Best-first Search with Uniform Cost spreads evenly from the start.

**A**

- Best-first Search with $Q$ ordered by admissible $f = g + h$.

Heuristic Value $h$ in Red
Edge cost in Green
### A* Search: State of Search

**Problem:** State Space Search Problem.

- **θ** Initial State.
- **Expand(node)** Children of Search Node = adjacent states.
- **Goal-Test(node)** True if search node at a goal-state.
- **Nodes** Search Nodes to be expanded.
- **Expanded** Search Nodes already expanded.
- **Initialize** Search starts at θ, with no expanded nodes.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g(state)</td>
<td>Cost to state</td>
</tr>
<tr>
<td>h(state)</td>
<td>Admissible Heuristic - Optimistic cost to go.</td>
</tr>
</tbody>
</table>

**Search Node:** Node in the search tree.

- **State** State the search is at.
- **Parent** Parent in search tree.

**Nodes[Problem]:**

- **Enqueue(node, f)** Adds node to those to be expanded.
- **Remove-Best(f)** Removes best cost queued node according to f.

### A* Search

**Function** A*(problem, h)

**returns** the best solution or failure. Problem pre-initialized.

\[
f(x) \leftarrow g[\text{problem}](x) + h(x)
\]

**loop do**

- **node** $\leftarrow$ Remove-Best(Nodes[problem], f)

**new-nodes** $\leftarrow$ Expand(node, problem)

**for each** new-node in new-nodes

- then Nodes[problem] $\leftarrow$ Enqueue(Nodes[problem], new-node, f)

**end**
A* Search

Function $A^*(problem, h)$
returns the best solution or failure. Problem pre-initialized.

\[ f(x) \leftarrow g[problem](x) + h(x) \]

loop do
  if Nodes[problem] is empty then return failure
  node ← Remove-Best(Nodes[problem], f)
  new-nodes ← Expand(node, problem)
  for each new-node in new-nodes
    then Nodes[problem] ← Enqueue(Nodes[problem], new-node, f)
  if Goal-Test[problem] applied to State(node) succeeds
    then return node
end

Terminate when . . .

Expand Vertices More Than Once

path length

- The shortest path from S to G is (G D A S).
- D is reached first using path (D S).

Suppose we expanded only the first path that visits each vertex X?
Expand Vertices More Than Once

Suppose we expanded only the first path that visits each vertex X?

• The shortest path from S to G is (G D A S).
• D is reached first using path (D S).
• This prevents path (D A S) from being expanded.

Suppose we expanded only the first path that visits each vertex X?

• The shortest path from S to G is (G D A S).
• D is reached first using path (D S).
• This prevents path (D A S) from being expanded.
Expand Vertices More Than Once

Suppose we expanded only the first path that visits each vertex X?

- Eliminate the Visited List.
- Return solution when taken off queue.

The shortest path from S to G is (G D A S).

D is reached first using path (D S).

This prevents path (D A S) from being expanded.

The suboptimal path (G D S) is returned.

Shortest Paths Contain Only Shortest Paths

- Subpaths of shortest paths are shortest paths.
  - $s \rightarrow^p v = <s, x, u, v>$ Shortest, then ….
  - $s \rightarrow^p u = <s, x, u>$ Shortest
  - $s \rightarrow^p x = <s, x>$ Shortest
Shortest Paths can be Grown From Shortest Paths

The length of shortest path \( s \rightarrow u \rightarrow v \) is \( \delta(s,v) = \delta(s,u) + w(u,v) \) such that \( \forall <u,v> \in E \quad \delta(s,v) \leq \delta(s,u) + w(u,v) \).

Dynamic Programming Principle:

- Given the shortest path to \( U \), don’t extend other paths to \( U \); delete them (expanded list).
- When A* dequeues the first partial path with head node \( U \), this path is guaranteed to be the shortest path from \( S \) to \( U \).

A* Search

Function \( A*(problem, h) \)

returns the best solution or failure. Problem pre-initialized.

\[
f(x) \leftarrow g(problem)(x) + h(x)
\]

loop do

if Nodes[problem] is empty then return failure

node \( \leftarrow \) Remove-Best(Nodes[problem], f)

state \( \leftarrow \) State(node)

remove any \( n \) from Nodes[problem] such that State(n) = state

Expanded[problem] \( \leftarrow \) Expanded[problem] \( \cup \) {state}

new-nodes \( \leftarrow \) Expand(node, problem)

for each new-node in new-nodes

unless State(new-node) is in Expanded[problem]

then Nodes[problem] \( \leftarrow \) Enqueue(Nodes[problem], new-node, f)

if Goal-Test[problem] applied to State(node) succeeds

then return node

end
Outline

• Model-based Diagnosis
• Optimal CSPs
• Informed Search
  – A*
  – Branch and Bound
• Conflict-directed A*

Branch and Bound

• Maintain the best solution found thus far (incumbent).
• Prune all subtrees worse than the incumbent.

Incumbent:
cost $U = \infty$, 8
path $P = ()$, (S A D G)

Heuristic Value $h$ in Red
Edge cost in Green
Branch and Bound

- Maintain the best solution found thus far (incumbent).
- Prune all subtrees worse than the incumbent.
- Any search order allowed (DFS, Reverse-DFS, BFS, Hill w BT…).

Simple Optimal Search Using Branch and Bound

Let \( <V,E> \) be a Graph
Let \( Q \) be a list of simple partial paths in \( <V,E> \)
Let \( S \) be the start vertex in \( <V,E> \) and Let \( G \) be a Goal vertex in \( <V,E> \).
Let \( f = g + h \) be an admissible heuristic function.

\( U \) and \( P \) are the cost and path of the best solution thus far (Incumbent).

1. Initialize \( Q \) with partial path \( (S) \); Incumbent \( U = \infty \), \( P = () \);
2. If \( Q \) is empty, return Incumbent \( U \) and \( P \),
   Else, remove a partial path \( N \) from \( Q \);
3. If \( f(N) \geq U \), Go to Step 2.
4. If head(\( N \)) = \( G \), then \( U = f(N) \) and \( P = N \) (a better path to the goal)
5. (Else) Find all children of head(\( N \)) (its neighbors in \( <V,E> \)) and
   create one-step extensions from \( N \) to each child.
6. Add extended paths to \( Q \).
7. Go to Step 2.
Outline

• Model-based Diagnosis
• Optimal CSPs
• Informed Search
• Conflict-directed A*
Conflict-directed A*

Increasing Cost

Infeasible

Feasible
Increasing Cost

Conflict-directed A*

Conflict 1

Infeasible

Feasible

Conflict-directed A*

Conflict 1

Infeasible

Conflict 2

Feasible
Increasing Cost

Conflict-directed A*

Conflict 1
Infeasible
Conflict 2

Conflict-directed A*

Conflict 1
Infeasible
Conflict 2

Conflict 3
Feasible
Increasing Cost

Conflict-directed A*

Conflict 1
Infeasible

Conflict 2
Feasible

Conflict 3

Solving Optimal CSPs Through Generate and Test

Leading Candidates Based on Cost

Generate Candidate

Test Candidate

Consistent?

Yes

No

Extract Conflict

(Optional) Update Cost

Done

Below Threshold?

Yes

No
Conflict-directed A*

Function Conflict-directed-A*(OCSP) returns the leading minimal cost solutions.
Conflicts[OCSP] ← {}  
OCSP ← Initialize-Best-Kernels(OCSP)  
Solutions[OCSP] ← {}
loop do  
  decision-state ← Next-Best-State-Resolving-Conflicts(OCSP)
  new-conflicts ← Extract-Conflicts(CSP[OCSP], decision-state)
  Conflicts[OCSP] ← Eliminate-Redundant-Conflicts(Conflicts[OCSP] ∪ new-conflicts)
end

Conflict-guided Expansion

Conflict-directed A*

Function Conflict-directed-A*(OCSP) returns the leading minimal cost solutions.
Conflicts[OCSP] ← {}  
OCSP ← Initialize-Best-Kernels(OCSP)  
Solutions[OCSP] ← {}
loop do  
  decision-state ← Next-Best-State-Resolving-Conflicts(OCSP)
  if no decision-state returned or Terminate?(OCSP) then return Solutions[OCSP]
  if Consistent?(CSP[OCSP], decision-state) then add decision-state to Solutions[OCSP]
  new-conflicts ← Extract-Conflicts(CSP[OCSP], decision-state)
  Conflicts[OCSP] ← Eliminate-Redundant-Conflicts(Conflicts[OCSP] ∪ new-conflicts)
end
Conflict-directed A*

- Each feasible subregion described by a kernel assignment.

Approach: Use conflicts to search for kernel assignment containing the best cost candidate.

Recall: Mapping Conflicts to Kernels

**Conflict C**: A set of decision variable assignments that are inconsistent with constraints $\Phi$.

$$C_i \land \Phi \text{ is inconsistent} \Rightarrow \Phi \text{ entails } \neg C_i$$

**Constituent Kernel**: An assignment $a$ that resolves a conflict $C_i$.

$$a \text{ entails } \neg C_i$$

**Kernel**: A minimal set of decision variable assignments that resolves all known conflicts $C$.

$$A \text{ entails } \neg C_i \text{ for all } C_i \in C$$
Extracting a kernel’s best state

- Select best utility value for unassigned variables (Why?).

\{X1=U\}

\[ A1=? \land A2=U \land A3=? \land X1=? \land X2=? \]

\[ A1=G \land A2=U \land A3=G \land X1=G \land X2=G \]

Next Best State Resolving Conflicts

**function** Next-Best-State-Resolving-Conflicts(OCSP)

\[ \text{best-kernel} \leftarrow \text{Next-Best-Kernel}(OCSP) \]

if best-kernel = failure
then return failure
else return kernel-Best-State[problem](best-kernel)
end

**function** Kernel-Best-State(kernel)

unassigned \leftarrow \text{all variables not assigned in kernel}
return kernel \cup \{\text{Best-Assignment}(v) \mid v \in \text{unassigned}\}

End

**function** Terminate?(OCSP)

return True iff Solutions[OCSP] is non-empty

Algorithm for only finding the first solution, multiple later.
Example: Diagnosis

Assume Independent Failures:
- $P_{G(mi)} \gg P_{U(mi)}$
- $P_{\text{single}} \gg P_{\text{double}}$
- $P_{U(A2)} > P_{U(A1)} > P_{U(A3)} > P_{U(X1)} > P_{U(X2)}$

First Iteration

- Conflicts / Constituent Kernels
  - none
- Best Kernel:
  - {}
- Best Candidate:
  - ?
Extracting the kernel’s best state

- Select best value for unassigned variables

\{ \} \downarrow

A1=? \land A2=? \land A3=? \land X1=? \land X2=?

\downarrow

A1=G \land A2=G \land A3=G \land X1=G \land 2=G

First Iteration

- Conflicts / Constituent Kernels
  - none
- Best Kernel:
  - \{\}
- Best Candidate:
  - A1=G \land A2=G \land A3=G \land X1=G \land X2=G
  - ?
Test: $A_1=G \land A_2=G \land A_3=G \land X_1=G \land X_2=G$

- Extract Conflict and Constituent Kernels:
  $\neg [A_1=G \land A_2=G \land X_1=G]$
  $A_1=U \lor A_2=U \lor X_1=U$
Second Iteration

- $P_{G(mi)} \gg P_{U(mi)}$
- $P_{single} \gg P_{double}$
- $P_{U(A2)} > P_{U(A1)} > P_{U(A3)} > P_{U(X1)} > P_{U(X2)}$

- Conflicts $\iff$ Constituent Kernels
  - $A1=U \lor A2=U \lor X1=U$
- Best Kernel:
  - $A2=U$ (why?)
- Best Candidate:
  - $A1=G \land A2=U \land A3=G \land X1=G \land X2=G$

Test: $A1=G \land A2=U \land A3=G \land X1=G \land X2=G$
Test: \(A1=\text{G} \land A2=\text{U} \land A3=\text{G} \land X1=\text{G} \land X2=\text{G}\)

- Extract Conflict:
  \[\neg [A1=\text{G} \land A3=\text{G} \land X1=\text{G} \land X2=\text{G}]\]

\[A1=\text{U} \lor A3=\text{U} \lor X1=\text{U} \lor X2=\text{U}\]

---

Third Iteration

- \(P_{G(mi)} \gg P_{U(mi)}\)
- \(P_{\text{single}} \gg P_{\text{double}}\)
- \(P_{U(A2)} > P_{U(A1)} > P_{U(A3)} > P_{U(X1)} > P_{U(X2)}\)

- Conflicts ⇒ Constituent Kernels
  - \(A1=\text{U} \lor A2=\text{U} \lor X1=\text{U}\)
  - \(A1=\text{U} \lor A3=\text{U} \lor X1=\text{U} \lor X2=\text{U}\)

- Best Kernel:
  - \(A1=\text{U}\)

- Best Candidate:
  - \(A1=\text{U} \land A2=\text{G} \land A3=\text{G} \land X1=\text{G} \land X2=\text{G}\)
Test: $A1 = U \land A2 = U \land A3 = G \land X1 = G \land X2 = G$

- Consistent!
Outline

- Model-based Diagnosis
- Optimal CSPs
- Conflict-directed A*
  - Generating the Best Kernel
  - Performance Comparison

Generating The Best Kernel of The Known Conflicts

In sight:
- Kernels found by minimal set covering
- Minimal set covering is an instance of breadth first search.
Expanding a Node to Resolve a Conflict

Constituent kernels

\[ A_2 = U \lor A_1 = U \lor X_1 = U \]

To Expand a Node:
- Select an unresolved Conflict.
- Each child adds a constituent kernel of Conflict.
- Prune any node that is
  - Inconsistent, or
  - A superset of a known kernel.

Generating The Best Kernel of The Known Conflicts

Insight:
- Kernels found by minimal set covering
- Minimal set covering is an instance of breadth first search.

\( \Rightarrow \) To find the best kernel, expand tree in best first order.
Admissible \( h(\alpha) \): Cost of best state that extends partial assignment \( \alpha \)
\[ f = g + h \]

\[ A2=U \quad \land A1=? \land A3=? \land X1=? \land X2=? \]

\[ P_{A2=u} \times P_{A1=G} \times P_{A3=G} \times P_{X1=G} \times P_{X2=G} \]

- Select best value of unassigned variables.

**Admissible Heuristic \( h \)**

- Let \( g = <G,g_i,Y> \) describe a multi-attribute utility fn
- Assume the preference for one attribute \( x_i \) is independent of another \( x_k \)
  - **Called Mutual Preferential Independence**:
    For all \( u, v \in Y \)
    If \( g_i(u) \geq g_i(v) \) then for all \( w \)
    \[ G(g_i(u),g_k(w)) \geq G(g_i(v),g_k(w)) \]

**An Admissible \( h \):**

- Given a partial assignment, to \( X \subseteq Y \)
- \( h \) selects the best value of each unassigned variable \( Z = X - Y \)

\[ h(Y) = G(\{g_{z_i,\max} \mid z_i \in Z, \max_{v_i \in D_{z_i}} g_{z_i}(v_i)\}) \]

- A candidate always exists satisfying \( h(Y) \).
Terminate when all conflicts resolved

**Function** Goal-Test-Kernel (node, problem)
- **returns** True IFF node is a complete decision state.
- if forall K in Constituent-Kernels(Conflicts[problem]),
  State[node] contains a kernel in K
  then return True
- else return False

Next Best Kernel of Known Conflicts

**Function** Next-Best-Kernel (OCSP)
- **returns** the next best cost kernel of Conflicts[OCSP].
  \( f(x) \leftarrow G[OCSP](g[OCSP](x), h[OCSP](x)) \)
- loop do
  - if Nodes[OCSP] is empty then return failure
  - node \leftarrow Remove-Best(Nodes[OCSP], f)
  - add State[node] to Visited[OCSP]
  - new-nodes \leftarrow Expand-Conflict(node, OCSP)
  - for each new-node \in new-nodes
    - unless \exists n \in Nodes[OCSP] such that State[new-node] = State[n]
      OR State[new-node] \in Visited[problem]
      then Nodes[OCSP] \leftarrow Enqueue(Nodes[OCSP], new-node, f)
  - if Goal-Test-Kernel[OCSP] applied to State[node] succeeds
    Best-Kernels[OCSP]
    \leftarrow Add-To-Minimal-Sets(Best-Kernels[OCSP], best-kernel)
    if best-kernel \in Best-Kernels[OCSP]
    then return State[node]
  end

An instance of A*
Outline

- Model-based Diagnosis
- Optimal CSPs
- Conflict-directed A*
  - Generating the Best Kernel
  - Performance Comparison

Performance: With and Without Conflicts

<table>
<thead>
<tr>
<th>Problem Parameters</th>
<th>Constraint-based A* (no conflicts)</th>
<th>Conflict-directed A*</th>
<th>Mean CD-CB Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes Expanded</td>
<td>Queue Size</td>
<td>Nodes Expanded</td>
</tr>
<tr>
<td>5 10 10 5</td>
<td>683</td>
<td>1,230</td>
<td>3.3</td>
</tr>
<tr>
<td>5 10 30 5</td>
<td>2,360</td>
<td>3,490</td>
<td>8.1</td>
</tr>
<tr>
<td>5 10 50 5</td>
<td>4,270</td>
<td>6,260</td>
<td>12.0</td>
</tr>
<tr>
<td>10 10 10 6</td>
<td>3,790</td>
<td>13,400</td>
<td>5.7</td>
</tr>
<tr>
<td>10 10 30 6</td>
<td>1,430</td>
<td>5,130</td>
<td>9.7</td>
</tr>
<tr>
<td>10 10 50 6</td>
<td>929</td>
<td>4,060</td>
<td>6.0</td>
</tr>
<tr>
<td>5 20 10 5</td>
<td>109</td>
<td>149</td>
<td>4.2</td>
</tr>
<tr>
<td>5 20 30 5</td>
<td>333</td>
<td>434</td>
<td>6.4</td>
</tr>
<tr>
<td>5 20 50 5</td>
<td>149</td>
<td>197</td>
<td>5.4</td>
</tr>
</tbody>
</table>
Multiple Fault Diagnosis of Systems with Novel Failures

**Consistency-based Diagnosis**: Given symptoms, find diagnoses that are consistent with symptoms.

**Suspending Constraints**: Make no presumption about a component's faulty behavior.

Model-based Diagnosis as Conflict-directed Best First Search

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

- Sherlock Holmes. The Sign of the Four.

1. Generate most likely Hypothesis.
2. Test Hypothesis.
3. If Inconsistent, learn reason for inconsistency (a Conflict).
4. Use conflicts to leap over similarly infeasible options to next best hypothesis.
Outline

- Model-based Diagnosis
- Optimal CSPs
- Conflict-directed A*
  - Generating the Best Kernel
  - Performance Comparison
  - Appendix:
    - Intelligent Tree Expansion
    - Extending to Multiple Solutions
    - Review of A*

Expand Only Best Child & Sibling

Constituent kernels
A2=U ∨ A1=U ∨ X1=U

Order constituents by decreasing utility.

- Traditionally all children expanded.
- Only need child containing best candidate.
⇒ Child with best estimated cost \( f = g + h \).
Expand Only Best Child & Sibling

Constituent kernels
\[ A_2 = U \lor A_1 = U \lor X_1 = U \]

Order constituents by decreasing utility

- Traditionally all children expanded.
- Only need child with best candidate.
- Child with best estimated cost \( f = g + h \).

When Do We Expand The Child’s Next Best Sibling?

Constituent kernels
\[ A_2 = U \lor A_1 = U \lor X_1 = U \]
\[ A_1 = U \lor A_3 = U \lor X_1 = U \lor X_2 = U \]

- When a best child has a subtree or leaf pruned, it may have lost its best candidate.
- One of the child’s siblings might now contain the best candidate.
- Expand child’s next best sibling:
  - when expanding children to resolve another conflict.
Expand Node to Resolve Conflict

**function** Expand-Conflict(node, OCSP)
**return** Expand-Conflict-Best-Child(node, OCSP) ∪ Expand-Next-Best-Sibling (node, OCSP)

**function** Expand-Conflict-Best-Child(node, OCSP)
if for all $K_v$ in Constituent-Kernels($\Gamma[OCSP]$)
   State[node] contains a kernel $\in K_v$
   then return {}
else return Expand-Constituent-Kernel(node,OCSP)

**function** Expand-Constituent-Kernel(node, OCSP)
$K_v \leftarrow \{\text{smallest uncovered set } \in \text{Constituent-Kernels}(\Gamma[OCSP])\}$
$C \leftarrow \{y_i = v_{ij} \mid \{y_i = v_{ij}\} \text{ in } K_v, y_i = v_{ij} \text{ is consistent with State[node]}\}$
Sort C such that for all $i$ from 1 to $|C| - 1,$
   Better-Kernel?($C[i],C[i+1], OCSP$) is True
Child-Assignments[node] $\leftarrow C$
$y_i = v_{ij} \leftarrow C[1], \text{ which is the best kernel in } K_v \text{ consistent with State[node]}$
**return** {Make-Node($\{y_i = v_{ij}\}$, node)}

Expand Node to Resolve Conflict

**function** Expand-Next-Best-Sibling(node, OCSP)
if Root?[node]
   then return {}
else $\{y_i = v_{ij}\} \leftarrow \text{Assignment[node]}
   \{y_k = v_{kl}\} \leftarrow \text{next best assignment in consistent child-assignments[Parent[node]] after } \{y_i = v_{ij}\}$
   if no next assignment $\{y_k = v_{kl}\}$
      or Parent[node] already has a child with $\{y_k = v_{kl}\}$
      then return {}
   else return {Make-Node($\{y_k = v_{kl}\}$, Parent[node])}
Outline

• Model-based Diagnosis
• Optimal CSPs
• Conflict-directed A*
  – Generating the Best Kernel
  – Performance Comparison
  – Appendix:
    • Intelligent Tree Expansion
    • Extending to Multiple Solutions
    • Review of A*

Multiple Solutions: Systematically Exploring Kernels

Constituent Kernels

X1=U, A1=U, A2=U

X1=U, X2=U, A1=U, A3=U

X1=U, A1=U

A1=U

A2=U

X2=U

A3=U

X1=U

A1=U

A1=U ∧ X2=U

A2=U ∧ A3=U
Child Expansion For Finding Multiple Solutions

Conflict
\[ \neg (A_2 = G \land A_1 = G \land X_1 = G) \]

If Unresolved Conflicts:
- Select unresolved conflict.
- Each child adds a constituent kernel.

If All Conflicts Resolved:
- Select unassigned variable \( y_i \).
- Each child adds an assignment from \( D_i \).

Intelligent Expansion Below a Kernel

Select Unassigned Variable.
\[ A_2 = G \lor A_2 = U \]

Order assignments by decreasing utility.

Expand best child.

Continue expanding best descendents.

When leaf visited, expand all next best ancestors. (why?)
Putting It Together:
Expansion Of Any Search Node

Constituent kernels
• $A_2 = U \lor A_1 = U \lor X_1 = U$
• $A_1 = U \lor A_3 = U \lor X_1 = U \lor X_2 = U$

- When a best child loses any candidate, expand child’s next best sibling:
  - If child has unresolved conflicts, expand sibling when child expands its next conflict.
  - If child resolves all conflicts, expand sibling when child expands a leaf.

Conflict-directed A*

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

- Sherlock Holmes. The Sign of the Four.

1. Generate most likely hypothesis.
2. Test hypothesis.
3. If inconsistent, learn reason for inconsistency (a Conflict).
4. Use conflicts to leap over similarly infeasible options to next best hypothesis.
Outline

• Using conflicts in backtrack search
  – Dependency-directed backtracking
  – Conflict learning
  – Conflict-directed backjumping

Using Conflicts to Guide Search: Dependency-directed Search
[Stallman & Sussman, 1978]

Input: Constraint satisfaction problem.
Output: Satisfying assignment.

Repeat while a next candidate assignment exists.
• Generate candidate assignment c.
• Check candidate c against conflicts.
  – If c is a superset of a conflict,
    Then loop to the next candidate.
• Check consistency of c.
  – If inconsistent,
  – Then extract and record a conflict from c.
  – Else return c as a solution.

⇒ Like a Graphplan memo, but generalizes an inconsistent solution.
Procedure Dependency_directed_Backtracking
\(<X,D,C>\)

Input: A constraint network \( R = <X, D, C> \)
Output: A solution, or notification that the network is inconsistent.

\[ i \leftarrow 1; \quad a_i = \{\}; \quad conflicts = \{\} \]
Initialize variable counter, assignments, conflicts

\[ D'_i \leftarrow D_i; \]
Copy domain of first variable.

\[ \text{while } 1 \leq i \leq n \]
while

\[ \text{ instantiate } x_i \leftarrow \text{Select-DDB-Value}(); \]
Add to assignments \( a_i \)

\[ \text{ if } x_i \text{ is null} \]
if

\[ i \leftarrow i - 1; \]
No value was returned,
then backtrack

\[ \text{ else} \]
else

\[ i \leftarrow i + 1; \quad D'_i \leftarrow D_i; \]
else step forward and copy domain of next variable

end while

\[ \text{ if } i = 0 \]
if

\[ \text{ return } \text{“inconsistent”} \]
return “inconsistent”

\[ \text{ else} \]
else

\[ \text{ return } a_i, \text{ the instantiated values of } \{x_i, \ldots, x_n\} \]
return \( a_i \), the instantiated values of \( \{x_i, \ldots, x_n\} \)

end procedure

Procedure Select-DDB-Value()

Output: A value in \( D'_i \) consistent with \( a_{i-1} \), or null, if none.

\[ \text{ while } D'_i \text{ is not empty} \]
while

\[ \text{ select an arbitrary element } a \in D'_i \text{ and remove a from } D'_i; \]
select an arbitrary element \( a \in D'_i \) and remove \( a \) from \( D'_i \);

\[ a_i \leftarrow a_{i-1} \cup \{x_i = a\}; \]
\( a_i \leftarrow a_{i-1} \cup \{x_i = a\} \);

\[ \text{ if for every } c \text{ in conflicts, not } (a_i \text{ superset } c) \]
if for every \( c \) in conflicts, not \( (a_i \supseteq c) \)

\[ \text{ if consistent}(a_{i-1}, x_i = a) \]
if consistent\( (a_{i-1}, x_i = a) \)

\[ \text{ return } a; \]
return \( a \);

\[ \text{ else } \]
else

\[ \text{ conflicts } \leftarrow \text{ conflicts } \cup \]
conflicts \( \leftarrow \) conflicts \( \cup \)

\[ \text{ minimal inconsistent subset of } a_{i-1}; \]
minimal inconsistent subset of \( a_{i-1} \);

end while

return null

end procedure