Optimal CSPs and Conflict-directed A*

Brian C. Williams
16.412J/6.834J
February 22nd, 2005

courtesy of JPL
Mode Estimation:
Select a most likely set of component modes that are consistent with the model and observations

Mode Reconfiguration:
Select a least cost set of commandable component modes that entail the current goal, and are consistent

arg min \( P_t(Y \mid \text{Obs}) \)
\[ \text{s.t. } \Psi(X,Y) \land O(m') \text{ is consistent} \]

arg max \( R_t(Y) \)
\[ \text{s.t. } \Psi(X,Y) \text{ entails } G(X,Y) \]
\[ \text{s.t. } \Psi(X,Y) \text{ is consistent} \]
Outline

- Optimal CSPs
- Application to Model-based Execution
  - Review of A*
  - Conflict-directed A*
  - Generating the Best Kernel
  - Intelligent Tree Expansion
  - Extending to Multiple Solutions
  - Performance Comparison
Constraint Satisfaction Problem

CSP = \langle X, D_X, C \rangle

- variables X with domain D_X
- Constraint C(X): \( D_X \rightarrow \{\text{True}, \text{False}\} \)

Find X in \( D_X \) s.t. C(X) is True

Diagram:
- \( V_1, V_2, V_3 \)
- \( \text{Different-color constraint} \)
- \( \{R, G, B\} \)
Optimal CSP

OCSP = \langle Y, g, \text{CSP} \rangle

- Decision variables \(Y\) with domain \(D_Y\)
- Utility function \(g(Y): D_Y \rightarrow \mathbb{R}\)
- CSP is over variables \(\langle X, Y \rangle\)

Find Leading arg max \(g(Y)\)
\[
Y \in D_y
\]

s.t. \(\exists X \in D_X\) s.t. \(C(X, Y)\) is True

- Frequently we encode \(C\) in propositional state logic
- \(g()\) is a multi-attribute utility function that is preferentially independent.
CSP Frequently in Propositional 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

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 $W - Y$. 
Mutual Preferential Independence

Example: Diagnosis

- If $M_1 = G$ is more likely than $M_1 = U$, 

- Then, 
  \[ \{M_1 = G, M_2 = G, M_3 = U, A_1 = G, A_2 = G\} \]

- Is preferred to 
  \[ \{M_1 = U, M_2 = G, M_3 = U, A_1 = G, A_2 = G\} \]
Reconfiguration via Conflict Learning

Goal: Achieve Thrust

arg max Rt(Y)  
s.t. \( \Psi(X,Y) \) entails G(X,Y)  
s.t. \( \Psi(X,Y) \) is consistent

A conflict is an assignment to a subset of the control variables that entails the negation of the goal.
Approximate PCCA Belief State Update

- Assigns a value to each variable (e.g., 3,000 vars).
- Consistent with all state constraints (e.g., 12,000).

- A set of concurrent transitions, one per automata (e.g., 80).
- Previous & Next states consistent with source & target of transitions
Belief State Propagation

- **Propagation Equation**
  propagates the system dynamics

  \[
  P(s_{j}^{t+1}|o^{<0,t>}, \mu^{<0,t>}) = \sum_{s_{i}^{t} \in S^{t}} \left( P(s_{j}^{t+1}|s_{i}^{t}, \mu^{t}) P(s_{i}^{t}|o^{<0,t>}, \mu^{<0,t-1)}) \right)
  \]

- **Update Equation**
  updates prior distribution with observations

  \[
  P(s_{j}^{t+1}|o^{<0,t+1>}, \mu^{<0,t>}) = \frac{P(s_{j}^{t+1}|o^{<0,t>}, \mu^{<0,t>}) \cdot P(o^{t+1}|s_{j}^{t+1})}{\sum_{s_{i}^{t+1} \in S^{t+1}} P(s_{i}^{t+1}|o^{<0,t>}, \mu^{<0,t>}) \cdot P(o^{t+1}|s_{i}^{t+1})}
  \]
Best-First Belief State Enumeration

- Enumerate next state priors in best first order
- Evaluate likelihood of partial states using optimistic estimate of unassigned variables.

\[
f(n) = \sum_{s_j^t \in S^t} \left( P(s_j^{t+1} | s_i^t, \mu^t) P(s_i^t | o^{<0,t>}, \mu^{<0,t-1>}) \right)
\]

\[
= \sum_{s_i^t \in S^t} \left( \prod_{x_a \in s_j} P(x_a^{t+1} = v' | x_a^t = v, \mu^t) P(s_i^t | o^{<0,t>}, \mu^{<0,t-1>}) \right)
\]

\[
= \sum_{s_i^t \in S^t} \left( \prod_{x_g \in n} P(x_g^{t+1} = v' | x_g^t = v, \mu^t) \prod_{x_h \notin n} \left( \max_{v' \in \mathcal{D}(x_h)} P(x_h^{t+1} = v' | x_h^t = v, \mu^t) \right) P(s_i^t | o^{<0,t>}, \mu^{<0,t-1>}) \right)
\]

\[\text{cost so far, } g \quad \text{optimistic estimate of the cost to go, } h\]
Outline

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

Problem: State Space Search Problem
- $\emptyset$: Initial State
- $\text{Expand}(node)$: Children of Search Node = next states
- $\text{Goal-Test}(node)$: True if search node at a goal-state

$h$: Admissible Heuristic - Optimistic cost to go

Search Node: Node in the search tree
- State: State the search is at
- Parent: Parent in search tree
### A* Search: State of Search

**Problem:** State Space Search Problem

- \( \Theta \) 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 \( \Theta \), with no expanded nodes

**g(state)** Cost to state

**h(state)** Admissible Heuristic-Optimistic cost to go

**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 \( f \)
Function $A^*(\text{problem}, h)$
returns the best solution or failure. Problem pre-initialized.
$f(x) \leftarrow g[\text{problem}](x) + h(x)$
loop do

$node \leftarrow \text{Remove-Best}($Nodes[problem], f$)$

$new-nodes \leftarrow \text{Expand}(node, \text{problem})$
for each new-node in new-nodes

then Nodes[problem] $\leftarrow \text{Enqueue}($Nodes[problem], new-node, f$)$$
end
A* Search

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

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

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

Terminates when . . .
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
end

Dynamic Programming Principle . . .
Outline

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

Increasing Cost

Feasible

Infeasible
Conflict-directed A*
Conflict-directed A*

Increasing Cost

Conflict 1

Infeasible

Feasible
Conflict-directed A*
Conflict-directed A*
Conflict-directed A*
Conflict-directed A*
Solving Optimal CSPs Through Generate and Test

Leading Candidates Based on Cost

Generate Candidate

Test Candidate

Consistent?

Yes

Keep

(Optional) Update Cost

Yes

Below Threshold?

Done

No

Extract Conflict

Conflicts-directed A*

Incremental Sat

No

Yes
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-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**

- Feasible subregions described by kernel assignments.
- Approach: Use conflicts to search for kernel assignment containing the best cost candidate.
**Mapping Conflicts to Kernels**

**Conflict** $C_i$: A set of decision variable assignments that are **inconsistent** with the constraints.

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

A entails $\square \neg C_i$

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

A entails $\square \neg C_i$ for all $C_i$ in $C$

2/22/2005  
copyright Brian Williams, 2002  
33
Mapping conflict to constituent kernels

Conflict: \{M1=G, M2=G, A1=G\}

\neg (M1=G \land M2=G \land A1=G)

M1=U \lor M2=U \lor M3=U

Constituent Kernels: \{M1=U, M2=U, A1=U\}
Composing Constituents
Kernels of Every Conflict

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

$A$ entails $\square \neg C_i$

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

$A$ entails $\square \neg C_i$ for all $C_i$ in $C$

$\Rightarrow$ Constituent kernels map to kernels by minimal set covering
Extracting a kernel’s best state

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

\{M2=U\}

\[\begin{align*}
    \text{M1}=\text{?} & \land \text{M2}=\text{U} & \land \text{M3}=\text{?} & \land \text{A1}=\text{?} & \land \text{A2}=\text{?} \\
    \text{M1}=\text{G} & \land \text{M2}=\text{U} & \land \text{M3}=\text{G} & \land \text{A1}=\text{G} & \land \text{A2}=\text{G}
\end{align*}\]
Next Best State
Resolving Conflicts

function Next-Best-State-Resolving-Conflicts(OCSP)
  best-kernel ← 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 ← all variables not assigned in kernel
  return kernel ∪ \{Best-Assignment(v) | v ∈ 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:

- \( P_{G(m_i)} \gg P_{U(m_i)} \)
- \( P_{\text{single}} \gg P_{\text{double}} \)
- \( P_{U(M2)} > P_{U(M1)} > P_{U(M3)} > P_{U(A1)} > P_{U(A2)} \)
First Iteration

**Conflicts / Constituent Kernels**
- none

**Best Kernel:**
- {}

**Best Candidate:**
- ?
Extracting the kernel’s best state

- Select best value for unassigned variables

\[
\begin{align*}
\{ & \} \\
M1=\? & \land M2=\? & \land M3=\? & \land A1=\? & \land A2=\?
\end{align*}
\]

\[
\begin{align*}
M1=G & \land M2=G & \land M3=G & \land A1=G & \land A2=G
\end{align*}
\]
- Test: \( M1=G \land M2=G \land M3=G \land A1=G \land A2=G \)
Test: $M1=G \land M2=G \land M3=G \land A1=G \land A2=G$
- Test: $M_1 = G \land M_2 = G \land M_3 = G \land A_1 = G \land A_2 = G$
Test: \( M1 = G \land M2 = G \land M3 = G \land A1 = G \land A2 = G \)
Test: $M1=G \land M2=G \land M3=G \land A1=G \land A2=G$
- **Test:** \( M_1 = G \land M_2 = G \land M_3 = G \land A_1 = G \land A_2 = G \)

- **Extract Conflict and Constituent Kernels:**
- **Test:** $M_1 = G \land M_2 = G \land M_3 = G \land A_1 = G \land A_2 = G$

- **Extract Conflict and Constituent Kernels:**
- Test: \( M1=G \land M2=G \land M3=G \land A1=G \land A2=G \)

- Extract Conflict and Constituent Kernels:

  \[ \neg [M1=G \land M2=G \land A1=G] \]

  \( M1=U \lor M2=U \lor A1=U \)
Second Iteration

- \( P_{G(mi)} >> P_{U(mi)} \)
- \( P_{\text{single}} >> P_{\text{double}} \)
- \( P_{U(M2)} > P_{U(M1)} > P_{U(M3)} > P_{U(A1)} > P_{U(A2)} \)

### Conflicts \( \iff \) Constituent Kernels

- \( M1=U \lor M2=U \lor A1=U \)

### Best Kernel:

- \( M2=U \) (why?)

### Best Candidate:

- \( M1=G \land M2=U \land M3=G \land A1=G \land A2=G \)
Test: $M1=G \land M2=U \land M3=G \land A1=G \land A2=G$
Test: $M1 = G \land M2 = U \land M3 = G \land A1 = G \land A2 = G$
Test: $M1=G \land M2=U \land M3=G \land A1=G \land A2=G$
Test: $M_1 = G \land M_2 = U \land M_3 = G \land A_1 = G \land A_2 = G$
Test: $M_1 = G \land M_2 = U \land M_3 = G \land A_1 = G \land A_2 = G$
Test: $M1=G \land M2=U \land M3=G \land A1=G \land A2=G$

Extract Conflict:

$\neg [M1=G \land M3=G \land A1=G \land A2=G]$
- **Test:** $M1 = G \land M2 = U \land M3 = G \land A1 = G \land A2 = G$

- **Extract Conflict:**
  \[\neg [M1 = G \land M3 = G \land A1 = G \land A2 = G]\]
  
  $M1 = U \lor M3 = U \lor A1 = U \lor A2 = U$
Second Iteration

- \( P_{G(mi)} \gg P_{U(mi)} \)
- \( P_{\text{single}} \gg P_{\text{double}} \)
- \( P_{U(M2)} > P_{U(M1)} > P_{U(M3)} > P_{U(A1)} > P_{U(A2)} \)

Conflicts \( \Rightarrow \) Constituent Kernels

- \( M1=U \lor M2=U \lor A1=U \)
- \( M1=U \lor M3=U \lor A1=U \lor A2=U \)

Best Kernel:

- \( M1=U \)

Best Candidate:

- \( M1=U \land M2=G \land M3=G \land A1=G \land A2=G \)
Test: $M1=U \land M2=G \land M3=G \land A1=G \land A2=G$
Test: $M1=U \land M2=G \land M3=G \land A1=G \land A2=G$
Test: \( M_1 = U \land M_2 = G \land M_3 = G \land A_1 = G \land A_2 = G \)
Test: $M1=U \land M2=G \land M3=G \land A1=G \land A2=G$
Test: \( M1=U \land M2=G \land M3=G \land A1=G \land A2=G \)
Test: \( M_1 = U \land M_2 = G \land M_3 = G \land A_1 = G \land A_2 = G \)

Consistent!
Outline

- Optimal CSPs
  - Application to Model-based Execution
  - Review of A*
  - Conflict-directed A*
  - Generating the Best Kernel
  - Intelligent Tree Expansion
  - Extending to Multiple Solutions
  - Performance Comparison
Generating The Best Kernel of The Known Conflicts

Constituent Kernels

A1=U, M1=U, M2=U
A1=U, A2=U, M1=U, M3=U
M2=U ∧ M3=U

Insight:
• 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

\[ \{ \} \]

\[ M_2 = U \lor M_1 = U \lor A_1 = U \]

To Expand a Node:

- Select an unresolved Conflict.
- Each child adds a constituent kernel.
- Prune child if state is
  - Inconsistent, or
  - subsumed by a known kernel (or another node’s state).
Generating The Best Kernel of The Known Conflicts

Constituent Kernels

A1=U, M1=U, M2=U
A1=U, A2=U, M1=U, M3=U
M2=U ∧ M3=U

Insight:
- Kernels found by minimal set covering
- Minimal set covering is an instance of breadth first search.
Generating The Best Kernel of The Known Conflicts

Constituent Kernels

A1=U, M1=U, M2=U

M1=U

A1=U, A2=U, M1=U, M3=U

M1=U

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

⇒ To find the best kernel, expand tree in best first order.
Admissible $h(\alpha)$: Cost of best state extending partial assignment $\alpha$

\[
f = g + h
\]

\[
M2=U \land M1=? \land M3=? \land A1=? \land A2=\
\]

\[
P_{M2=u} \times P_{M1=G} \times P_{M3=G} \times P_{A1=G} \times P_{A2=G}
\]

- Select best value of unassigned variables
Admissible Heuristic $h$

- Let $g = \langle G, g_i, Y \rangle$ 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 $Z = X - Y$

$$h(Y) = G(\{g_{z_i \max} | z_i \in Z, \max_{v_{ij} \in D_{z_i}} g_{z_i}(v_{ij})\})$$

- 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) ← G[OCSP] (g[OCSP](x), h[OCSP](x))
loop do
  if Nodes[OCSP] is empty then return failure
  node ← Remove-Best(Nodes[OCSP], f)
  add State[node] to Visited[OCSP]
  new-nodes ← Expand-Conflict(node, OCSP)
  for each new-node ∈ new-nodes
    unless ∃ n ∈ Nodes[OCSP] such that State[new-node] = State[n]
    OR State[new-node] ∈ Visited[problem]
    then Nodes[OCSP] ← Enqueue(Nodes[OCSP], new-node, f)
  if Goal-Test-Kernel[OCSP] applied to State[node] succeeds
    Best-Kernels[OCSP] ← Add-To-Minimal-Sets(Best-Kernels[OCSP], best-kernel)
    if best-kernel ∈ Best-Kernels[OCSP]
      then return State[node]
end

An instance of A*

2/22/2005  copyright Brian Williams, 2002  72
Outline

• Optimal CSPs
  ▪ Application to Model-based Execution
  ▪ Review of A*
  ▪ Conflict-directed A*
  ▪ Generating the Best Kernel
  ▪ Intelligent Tree Expansion
  ▪ Extending to Multiple Solutions
  ▪ Performance Comparison
Expand Only Best Child & Sibling

Constituent kernels

M2=U \lor M1=U \lor A1=U

Order constituents by decreasing utility

Traditionally all children expanded.

But only need to expand the child with the best candidate, if it can be identified apriori (how?).

This child is the one with the best estimated cost \( f = g + h \).
Expand Only Best Child & Sibling

Constituent kernels

\[ M_2 = U \lor M_1 = U \lor A_1 = U \]

Order constituents by decreasing utility

- Traditionally all children expanded.
- But only need to expand the child with the best candidate, if it can be identified apriori (how?).
  
  \[ \Rightarrow \] This child is the one with the best estimated cost \( f = g + h \).
When Do We Expand The Child’s Next Best Sibling?

Constituent kernels

- $M_2 = U \lor M_1 = U \lor A_1 = U$
- $M_1 = U \lor M_3 = U \lor A_1 = U \lor A_2 = U$

- 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 have the best candidate.
- ➔ Expand child’s next best sibling:
  - when child expanded in order 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 $K_γ$ in Constituent-Kernels(Γ[OCSP])
  State[node] contains a kernel $∈ K_γ$
then return {}
else return Expand-Constituent-Kernel(node, OCSP)

function Expand-Constituent-Kernel(node, OCSP)
$K_γ ← = smallest uncovered set $ ∈ Constituent-Kernels(Γ[OCSP])
$C ← \{y_i = v_{ij} | \{y_i = v_{ij}\} in K_γ, y_i = v_{ij} is consistent with State[node]\}$
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] ← $C$
y_i = v_{ij} ← $C[1]$, which is the best kernel in $K_γ$ consistent with State[node]
return {Make-Node({y_i = v_{ij}}, node)}
function Expand-Next-Best-Sibling(node, OCSP)
if Root?[node]
    then return {}
else \{y_i = v_ij\} ← Assignment[node]
    \{y_k = v_ki\} ← next best assignment in consistent child-assignments[Parent[node]] after \{y_i = v_ij\}
    if no next assignment \{y_k = v_ki\}
        or Parent[node] already has a child with \{y_k = v_ki\}
        then return {}
    else return \{Make-Node(\{y_k = v_ki\}, Parent[node])\}
Outline

- Optimal CSPs
  - Application to Model-based Execution
  - Review of A*
  - Conflict-directed A*
  - Generating the Best Kernel
  - Intelligent Tree Expansion
  - Extending to Multiple Solutions
  - Performance Comparison
Multiple Solutions: Systematically Exploring Kernels

Constituent Kernels

A1=U, M1=U, M2=U

A1=U, A2=U, M1=U, M3=U

A1=U, M1=U, M2=U

A2=U

M3=U

A1=U

M1=U

M1=U ∧ A2=U

M2=U ∧ M3=U
Child Expansion For Finding Multiple Solutions

Conflict
\neg (M2=G \land M1=G \land A1=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

M2=G ∨ M2=U

Order assignments by decreasing utility

Expand best child

Continue expanding best descendants

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

Constituent kernels

\[ M_2 = U \lor M_1 = U \lor A_1 = U \]
\[ M_1 = U \lor M_3 = U \lor A_1 = U \land A_2 = U \]

- 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.
Outline

- Optimal CSPs
  - Application to Model-based Execution
  - Review of A*
  - Conflict-directed A*
  - Generating the Best Kernel
  - Intelligent Tree Expansion
  - Extending to Multiple Solutions
  - 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>Dom Size</td>
<td>Dec Vars</td>
<td>Clauses Length</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>
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. Test Hypothesis
2. If inconsistent, learn reason for inconsistency (a Conflict).
3. Use conflicts to leap over similarly infeasible options to next best hypothesis.
Presentation Notes

- Change Example to Boolean Polycell
- Introduce CDA* before Sherlock-style Mode Estimation.
- Describe Kernels and Conflicts in terms of set/subset lattice.
- More Intuitive and focused introduction to A*
- Add systematicity in each development
- Add pseudo code for multiple solns and CBA*
- Show full search trees for each
- Highlight Important features of performance