Heuristics for Planning

Outline

I. Planning as boolean satisfiability

II. Heuristics for planning

III. Planning in nondeterministic domains

* Figures are from the textbook site.

I. Planning as Boolean Satisfiability

♣ Translate a PDDL problem description into propositional form.

I. Planning as Boolean Satisfiability

- Translate a PDDL problem description into propositional form.
  - Propositionalize the actions: substituting constants for variables.

    Unload(c, p, a)

I. Planning as Boolean Satisfiability

- Translate a PDDL problem description into propositional form.
  - Propositionalize the actions: substituting constants for variables.

$\boxed{Unload(c, p, a)}$ $\Longrightarrow$ action propositions for every combination of cargo, plane, and airport.

I. Planning as Boolean Satisfiability

- Translate a PDDL problem description into propositional form.
  - Propositionalize the actions: substituting constants for variables.

    $Unload(c, p, a)$ $\Longrightarrow$ action propositions for every combination of cargo, plane, and airport.

  - Add action exclusion axioms so no two actions can occur simultaneously.

$$\neg(FlyP_1SFOJFK^1 \wedge FlyP_1SFOORD^1)$$

I. Planning as Boolean Satisfiability

- Translate a PDDL problem description into propositional form.
  - Propositionalize the actions: substituting constants for variables.

    $\mathit{Unload}(c, p, a)$ $\Longrightarrow$ action propositions for every combination of cargo, plane, and airport.

  - Add action exclusion axioms so no two actions can occur simultaneously.

    $$\neg(\mathit{Fly}P_1\mathit{SFOJFK}^1 \wedge \mathit{Fly}P_1\mathit{SFOORD}^1)$$

  - Add precondition axioms $A^t \Rightarrow \textsc{Pre}(A)^t$

    $$\mathit{Fly}P_1\mathit{SFOJFK}^1 \Rightarrow \mathit{At}(P_1, \mathit{SFO}) \wedge \mathit{Plane}(P_1) \wedge \mathit{Airport}(\mathit{SFO}) \wedge \mathit{Airport}(\mathit{JFK})$$

I. Planning as Boolean Satisfiability

- Translate a PDDL problem description into propositional form.
  - Propositionalize the actions: substituting constants for variables.

    $\mathit{Unload}(c, p, a)$ $\Longrightarrow$ action propositions for every combination of cargo, plane, and airport.

  - Add action exclusion axioms so no two actions can occur simultaneously.

    $$\neg(\mathit{FlyP_1SFOJFK}^1 \wedge \mathit{FlyP_1SFOORD}^1)$$

  - Add precondition axioms $A^t \Rightarrow \textsc{Pre}(A)^t$

    $$\mathit{FlyP_1SFOJFK}^1 \Rightarrow \mathit{At}(P_1, \mathit{SFO}) \wedge \mathit{Plane}(P_1) \wedge \mathit{Airport}(\mathit{SFO}) \wedge \mathit{Airport}(\mathit{JFK})$$

  - Define the initial state S_0: assert F^0 for every fluent F mentioned in S_0 and $\neg F^0$ not mentioned in S_0.

Planning as Boolean Satisfiability (cont'd)

- Propositionalize the *goal*: a disjunction over all ground instances from replacing variables by constants.

Planning as Boolean Satisfiability (cont'd)

- Propositionalize the *goal*: a disjunction over all ground instances from replacing variables by constants.

$$\mathit{On}(A, x) \wedge \mathit{Block}(x)$$

Planning as Boolean Satisfiability (cont'd)

- Propositionalize the *goal*: a disjunction over all ground instances from replacing variables by constants.

$$\boxed{On(A,x) \wedge Block(x)}$$

$$\Big\Downarrow \{A, B, C\}$$

$$\big(On(A,A) \wedge Block(A)\big) \vee \big(On(A,B) \wedge Block(B)\big) \vee \big(On(A,C) \wedge Block(C)\big)$$

Planning as Boolean Satisfiability (cont'd)

- Propositionalize the *goal*: a disjunction over all ground instances from replacing variables by constants.

$$\mathit{On}(A, x) \wedge \mathit{Block}(x)$$

$$\Downarrow \{A, B, C\}$$

$$\big(\mathit{On}(A, A) \wedge \mathit{Block}(A)\big) \vee \big(\mathit{On}(A, B) \wedge \mathit{Block}(B)\big) \vee (\mathit{On}(A, C) \wedge \mathit{Block}(C))$$

- Add successor-state axioms for every fluent F.

$$F^{t+1} \Leftrightarrow \mathit{ActionCausesF}^t \vee (F^t \wedge \neg \mathit{ActionCausesNotF}^t)$$

Planning as Boolean Satisfiability (cont'd)

- Propositionalize the *goal*: a disjunction over all ground instances from replacing variables by constants.

$$\mathit{On}(A,x) \wedge \mathit{Block}(x)$$

$$\Big\Downarrow \quad \{A,B,C\}$$

$$\big(\mathit{On}(A,A) \wedge \mathit{Block}(A)\big) \vee \big(\mathit{On}(A,B) \wedge \mathit{Block}(B)\big) \vee \big(\mathit{On}(A,C) \wedge \mathit{Block}(C)\big)$$

- Add successor-state axioms for every fluent F.

$$F^{t+1} \Leftrightarrow \underbrace{\mathit{ActionCausesF}^t}_{\substack{\text{disjunction of all} \\ \text{actions that add } F}} \vee (F^t \wedge \neg \mathit{ActionCausesNotF}^t)$$

Planning as Boolean Satisfiability (cont'd)

- Propositionalize the *goal*: a disjunction over all ground instances from replacing variables by constants.

$$\mathit{On}(A, x) \wedge \mathit{Block}(x)$$

$$\Downarrow \quad \{A, B, C\}$$

$$\big(\mathit{On}(A, A) \wedge \mathit{Block}(A)\big) \vee \big(\mathit{On}(A, B) \wedge \mathit{Block}(B)\big) \vee \big(\mathit{On}(A, C) \wedge \mathit{Block}(C)\big)$$

- Add successor-state axioms for every fluent F.

$$F^{t+1} \Leftrightarrow \underbrace{\mathit{ActionCausesF}^t}_{\text{disjunction of all actions that add } F} \vee (F^t \wedge \neg \underbrace{\mathit{ActionCausesNotF}^t}_{\text{disjunction of all actions that delete } F})$$

Planning as Boolean Satisfiability (cont'd)

- Propositionalize the *goal*: a disjunction over all ground instances from replacing variables by constants.

$$\mathit{On}(A,x) \wedge \mathit{Block}(x)$$

$$\Downarrow \; \{A, B, C\}$$

$$\big(\mathit{On}(A,A) \wedge \mathit{Block}(A)\big) \vee \big(\mathit{On}(A,B) \wedge \mathit{Block}(B)\big) \vee \big(\mathit{On}(A,C) \wedge \mathit{Block}(C)\big)$$

- Add successor-state axioms for every fluent F.

$$F^{t+1} \Leftrightarrow \underbrace{\mathit{ActionCausesF}^t}_{\text{disjunction of all actions that add } F} \vee \big(F^t \wedge \neg \underbrace{\mathit{ActionCausesNotF}^t}_{\text{disjunction of all actions that delete } F}\big)$$

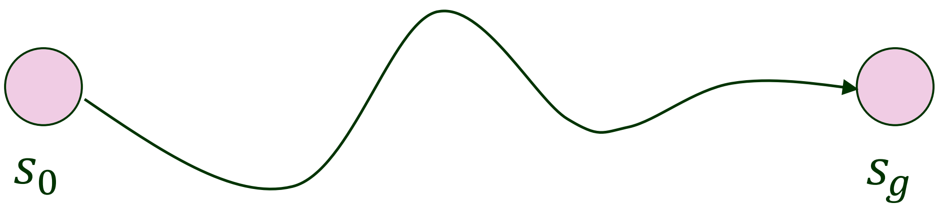
♣ Solve the resulting propositional logic satisfiability problem.

II. Heuristics for Planning

Search problem as conducted in a graph:

nodes $\Longleftrightarrow$ states

edges $\Longleftrightarrow$ actions



- Derive an admissible heuristic by relaxing the problem (which is easier) and using its solution.
  - Add more edges.
  - Group multiple nodes into one.

Ignore-Preconditions Heuristic

- Remove all preconditions and all effects except those that are literals in the goal.

Ignore-Preconditions Heuristic

- Remove all preconditions and all effects except those that are literals in the goal.
- Count the minimum number of actions whose effects have a union that satisfies goal.

Ignore-Preconditions Heuristic

- Remove all preconditions and all effects except those that are literals in the goal.

- Count the minimum number of actions whose effects have a union that satisfies goal.
  - Set covering (NP-hard)
    - S: set of literals in the goal.
    - $S_1, \ldots S_k$: sets of literals added by the k actions, respectively.

Ignore-Preconditions Heuristic

- Remove all preconditions and all effects except those that are literals in the goal.
- Count the minimum number of actions whose effects have a union that satisfies goal.
  - Set covering (NP-hard)
    - S: set of literals in the goal.
    - $S_1, \ldots S_k$: sets of literals added by the k actions, respectively.
  - Greedy solution (no guarantee of admissibility)

Ignore-Preconditions Heuristic

- Remove all preconditions and all effects except those that are literals in the goal.

- Count the minimum number of actions whose effects have a union that satisfies goal.
  - Set covering (NP-hard)
    - S: set of literals in the goal.
    - $S_1, \ldots S_k$: sets of literals added by the k actions, respectively.
  - Greedy solution (no guarantee of admissibility)

- An alternative is to ignore selected preconditions.

Ignore-Preconditions Heuristic

- Remove all preconditions and all effects except those that are literals in the goal.

- Count the minimum number of actions whose effects have a union that satisfies goal.
  - Set covering (NP-hard)
    - S: set of literals in the goal.
    - $S_1, \ldots S_k$: sets of literals added by the k actions, respectively.
  - Greedy solution (no guarantee of admissibility)

- An alternative is to ignore selected preconditions.

8-puzzle:

Action(*Slide*(t, s_1, s_2))

PRECOND: *On*$(t, s_1) \wedge$ *Tile*$(t) \wedge$ *Blank*$(s_2) \wedge$ *Adjacent*(s_1, s_2)

EFFECT: *On*$(t, s_2) \wedge$ *Blank*$(s_1) \wedge \neg$*On*$(t, s_1) \wedge \neg$*Blank*(s_2)

Ignore-Preconditions Heuristic

- Remove all preconditions and all effects except those that are literals in the goal.
- Count the minimum number of actions whose effects have a union that satisfies goal.
  - Set covering (NP-hard)
    - S: set of literals in the goal.
    - $S_1, \ldots S_k$: sets of literals added by the k actions, respectively.
  - Greedy solution (no guarantee of admissibility)
- An alternative is to ignore selected preconditions.

8-puzzle:

Action(*Slide*(t, s_1, s_2))

number of misplaced tiles

PRECOND: *On*$(t, s_1) \wedge$ *Tile*$(t) \wedge \cancel{Blank(s_2) \wedge Adjacent(s_1, s_2)}$

EFFECT: *On*$(t, s_2) \wedge$ *Blank*$(s_1) \wedge \neg$*On*$(t, s_1) \wedge \neg$*Blank*(s_2)

Ignore-Preconditions Heuristic

- Remove all preconditions and all effects except those that are literals in the goal.

- Count the minimum number of actions whose effects have a union that satisfies goal.

  - Set covering (NP-hard)
    - S: set of literals in the goal.
    - $S_1, \ldots S_k$: sets of literals added by the k actions, respectively.
  - Greedy solution (no guarantee of admissibility)

- An alternative is to ignore selected preconditions.

8-puzzle:

$Action(Slide(t, s_1, s_2))$

Manhattan distance

PRECOND: $On(t, s_1) \wedge Tile(t) \wedge \cancel{Blank(s_2)} \wedge Adjacent(s_1, s_2)$

EFFECT: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$

Ignore-Delete-Lists Heuristic

Assume all goals and preconditions contain only positive literals.

Ignore-Delete-Lists Heuristic

Assume all goals and preconditions contain only positive literals.

- Remove the delete lists from all actions,

Ignore-Delete-Lists Heuristic

Assume all goals and preconditions contain only positive literals.

- Remove the delete lists from all actions,
- Make monotonic progress towards the goal.

Ignore-Delete-Lists Heuristic

Assume all goals and preconditions contain only positive literals.

- Remove the delete lists from all actions,
- Make monotonic progress towards the goal.
- Still NP-hard to find the optimal solution in the relaxed problem.

Ignore-Delete-Lists Heuristic

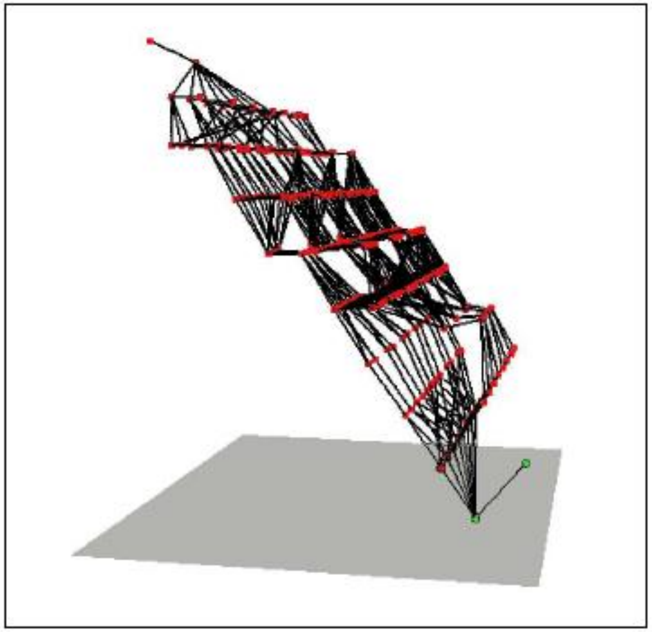
Assume all goals and preconditions contain only positive literals.

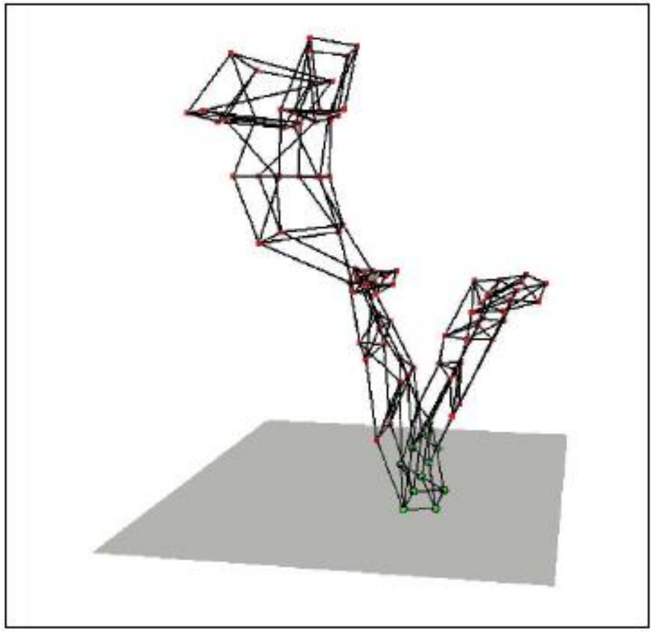
- Remove the delete lists from all actions,
- ♦ Make monotonic progress towards the goal.
- ♠ Still NP-hard to find the optimal solution in the relaxed problem.
- ♦ Can find an approximate solution in polynomial time.

Ignore-Delete-Lists Heuristic

Assume all goals and preconditions contain only positive literals.

- Remove the delete lists from all actions,
- ♦ Make monotonic progress towards the goal.
- ♠ Still NP-hard to find the optimal solution in the relaxed problem.
- ♦ Can find an approximate solution in polynomial time.

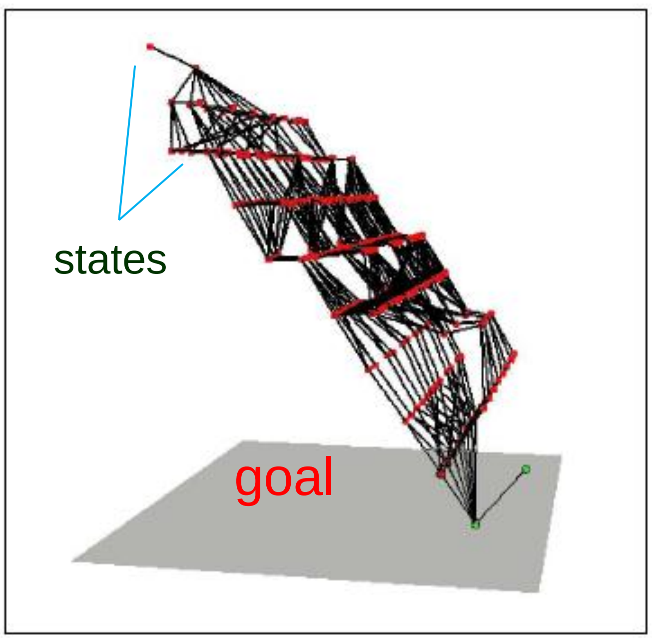


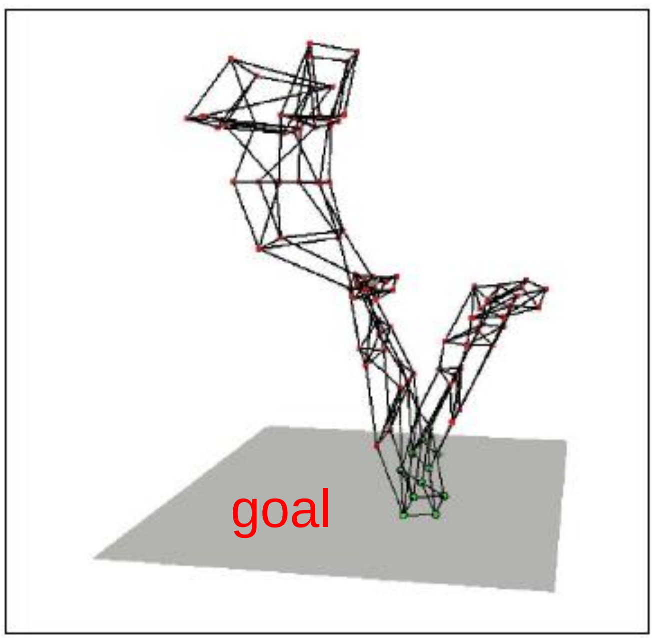


Ignore-Delete-Lists Heuristic

Assume all goals and preconditions contain only positive literals.

- Remove the delete lists from all actions,
- ♦ Make monotonic progress towards the goal.
- ♠ Still NP-hard to find the optimal solution in the relaxed problem.
- ♦ Can find an approximate solution in polynomial time.

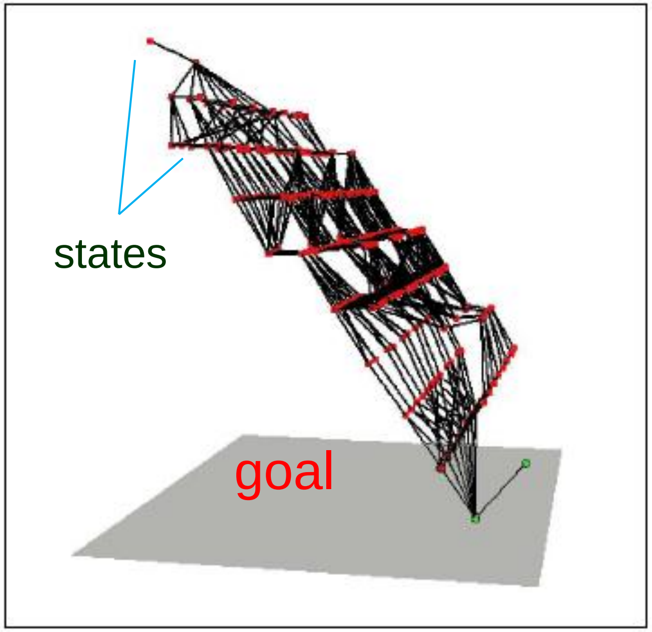


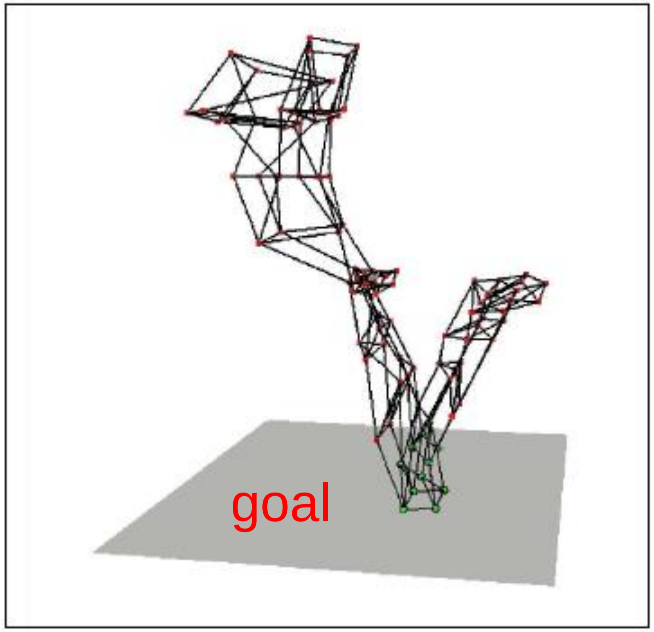


Ignore-Delete-Lists Heuristic

Assume all goals and preconditions contain only positive literals.

- Remove the delete lists from all actions,
- ♦ Make monotonic progress towards the goal.
- ♠ Still NP-hard to find the optimal solution in the relaxed problem.
- ♦ Can find an approximate solution in polynomial time.



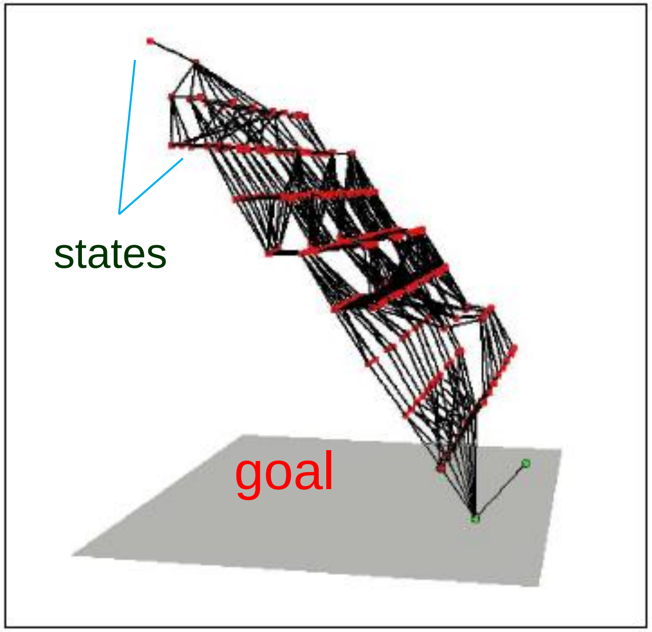


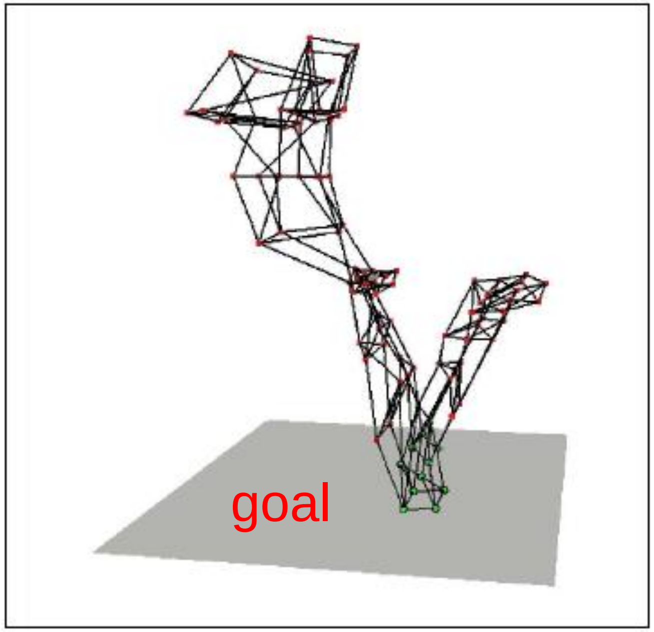
$h(s) =$ height above the plane

Ignore-Delete-Lists Heuristic

Assume all goals and preconditions contain only positive literals.

- Remove the delete lists from all actions,
- ♦ Make monotonic progress towards the goal.
- ♠ Still NP-hard to find the optimal solution in the relaxed problem.
- ♦ Can find an approximate solution in polynomial time.





$h(s)$ = height above the plane

Hill-climbing search works!

State Abstraction

- ♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).
- ♦ Set up a many-to-one mapping from physical states to more abstract ones.

State Abstraction

- ♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).
- ♦ Set up a many-to-one mapping from physical states to more abstract ones.

Heuristics of decomposition

- Divide-and-conquer

State Abstraction

♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).

♦ Set up a many-to-one mapping from physical states to more abstract ones.

Heuristics of decomposition

- Divide-and-conquer
- *Subgoal independence:* the cost of solving a conjunction of subgoals equals the sum of costs of solving them independently.

State Abstraction

♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).

♦ Set up a many-to-one mapping from physical states to more abstract ones.

Heuristics of decomposition

- Divide-and-conquer
- *Subgoal independence:* the cost of solving a conjunction of subgoals equals the sum of costs of solving them independently.

Goal set of fluents: $G = G_1 \cup G_2 \cup \cdots \cup G_n$

State Abstraction

- ♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).

- ♦ Set up a many-to-one mapping from physical states to more abstract ones.

Heuristics of decomposition

- Divide-and-conquer
- *Subgoal independence:* the cost of solving a conjunction of subgoals equals the sum of costs of solving them independently.

Goal set of fluents: $G = G_1 \cup G_2 \cup \cdots \cup G_n$

Optimal plans to
solve subgoals COST:

State Abstraction

♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).

♦ Set up a many-to-one mapping from physical states to more abstract ones.

Heuristics of decomposition

- Divide-and-conquer
- *Subgoal independence:* the cost of solving a conjunction of subgoals equals the sum of costs of solving them independently.

Goal set of fluents: $G = G_1 \cup G_2 \cup \cdots \cup G_n$

Optimal plans to solve subgoals COST: $P_1, P_2, \ldots, P_n$

State Abstraction

♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).

♦ Set up a many-to-one mapping from physical states to more abstract ones.

Heuristics of decomposition

- Divide-and-conquer
- *Subgoal independence:* the cost of solving a conjunction of subgoals equals the sum of costs of solving them independently.

Goal set of fluents: $G = G_1 \cup G_2 \cup \cdots \cup G_n$

Optimal plans to solve subgoals COST: $P_1, P_2, \ldots, P_n$

Choices of heuristic:

State Abstraction

♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).

♦ Set up a many-to-one mapping from physical states to more abstract ones.

Heuristics of decomposition

- Divide-and-conquer
- *Subgoal independence:* the cost of solving a conjunction of subgoals equals the sum of costs of solving them independently.

Goal set of fluents: $G = G_1 \cup G_2 \cup \cdots \cup G_n$

Optimal plans to solve subgoals COST: $P_1, P_2, \ldots, P_n$

Choices of heuristic:

♣ $\max_i$ COST(P_i)?

Admissible but too low.

State Abstraction

♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).

♦ Set up a many-to-one mapping from physical states to more abstract ones.

Heuristics of decomposition

- Divide-and-conquer
- *Subgoal independence:* the cost of solving a conjunction of subgoals equals the sum of costs of solving them independently.

Goal set of fluents: $G = G_1 \cup G_2 \cup \cdots \cup G_n$

Optimal plans to solve subgoals COST: $P_1, P_2, \ldots, P_n$

Choices of heuristic:

♣ $\max_i$ COST(P_i)?

Admissible but too low.

♣ $\sum_i$ COST(P_i)?

Not admissible.

State Abstraction

♠ Relaxing the actions does nothing to reduce #states ($\geq 10^{100}$ for many problems).

♦ Set up a many-to-one mapping from physical states to more abstract ones.

Heuristics of decomposition

- Divide-and-conquer
- *Subgoal independence:* the cost of solving a conjunction of subgoals equals the sum of costs of solving them independently.

Goal set of fluents: $G = G_1 \cup G_2 \cup \cdots \cup G_n$

Optimal plans to solve subgoals COST: $P_1, P_2, \ldots, P_n$

Choices of heuristic:

♣ $\max_i \text{COST}(P_i)$? Admissible but too low.

♣ $\sum_i \text{COST}(P_i)$? Not admissible.

♣ $\text{COST}(P_i) + \text{COST}(P_j)$? when P_i and P_j are independent.

III. Nondeterministic Domains

Task Make a chair and a table have the same color.

$Init(Object(Table) \land Object(Chair) \land Can(C_1) \land Can(C_2) \land InView(Table))$

$Goal(Color(Chair, c) \land Color(Table, c))$

III. Nondeterministic Domains

Task Make a chair and a table have the same color.

$$Init(\overbrace{Object(Table) \land Object(Chair)}^{\text{unknown colors}} \land Can(C_1) \land Can(C_2) \land InView(Table))$$

$$Goal(Color(Chair, c) \land Color(Table, c))$$

III. Nondeterministic Domains

Task Make a chair and a table have the same color.

$$Init(\overbrace{Object(Table) \wedge Object(Chair)}^{\text{unknown colors}} \wedge \overbrace{Can(C_1) \wedge Can(C_2)}^{\text{two cans of paint with unknown colors}} \wedge InView(Table))$$

$$Goal(Color(Chair, c) \wedge Color(Table, c))$$

III. Nondeterministic Domains

Task Make a chair and a table have the same color.

$$\mathit{Init}(\underbrace{\mathit{Object}(\mathit{Table}) \wedge \mathit{Object}(\mathit{Chair})}_{\text{unknown colors}} \wedge \underbrace{\mathit{Can}(C_1) \wedge \mathit{Can}(C_2)}_{\text{two cans of paint with unknown colors}} \wedge \mathit{InView}(\mathit{Table}))$$

$$\mathit{Goal}(\mathit{Color}(\mathit{Chair}, c) \wedge \mathit{Color}(\mathit{Table}, c))$$

Action(*RemoveLid*(*can*),
  PRECOND: *Can*(*can*)
  EFFECT: *Open*(*can*))

III. Nondeterministic Domains

Task Make a chair and a table have the same color.

$$\mathit{Init}(\overbrace{\mathit{Object}(\mathit{Table}) \wedge \mathit{Object}(\mathit{Chair})}^{\text{unknown colors}} \wedge \overbrace{\mathit{Can}(C_1) \wedge \mathit{Can}(C_2)}^{\text{two cans of paint with unknown colors}} \wedge \mathit{InView}(\mathit{Table}))$$

$$\mathit{Goal}(\mathit{Color}(\mathit{Chair}, c) \wedge \mathit{Color}(\mathit{Table}, c))$$

$\mathit{Action}(\mathit{RemoveLid}(\mathit{can})$,
  PRECOND: $\mathit{Can}(\mathit{can})$
  EFFECT: $\mathit{Open}(\mathit{can}))$

$\mathit{Action}(\mathit{Paint}(x, \mathit{can})$, // paint x using the paint from an open can
  PRECOND: $\mathit{Object}(x) \wedge \mathit{Can}(\mathit{can}) \wedge \mathit{Color}(\mathit{can}, c) \wedge \mathit{Open}(\mathit{can})$
  EFFECT: $\mathit{Color}(x, c)$

III. Nondeterministic Domains

Task Make a chair and a table have the same color.

$$\underbrace{Init(Object(Table) \wedge Object(Chair)}_{\text{unknown colors}} \wedge \underbrace{Can(C_1) \wedge Can(C_2)}_{\text{two cans of paint with unknown colors}} \wedge InView(Table))$$

$Goal(Color(Chair, c) \wedge Color(Table, c))$

Action(*RemoveLid*(*can*),
  PRECOND: *Can*(*can*)
  EFFECT: *Open*(*can*))

Action($Paint(x, can)$), // paint x using the paint from an open can
  PRECOND: $Object(x) \wedge Can(can) \wedge Color(can, c) \wedge Open(can)$
  EFFECT: $Color(x, c)$

No color argument c in $Paint(x, can)$.
// it would be $Paint(x, can, c)$ in the fully observable case.

Percept Schema

The agent needs to be able to reason about its percepts as it executes the plan.

Percept Schema

The agent needs to be able to reason about its percepts as it executes the plan.

Percept(*Color*(x, c),
  PRECOND: *Object*(x) $\wedge$ *InView*(x)

// the agent will perceive the color of
// an object in view.

Percept Schema

The agent needs to be able to reason about its percepts as it executes the plan.

Percept$($*Color*(x, c),
  PRECOND: *Object*$(x) \wedge$ *InView*(x)

// the agent will perceive the color of
// an object in view.

Percept$($*Color*$($*can*$, c)$,
  PRECOND: *Can*$($*can*$) \wedge$ *Inview*$($*can*$) \wedge$ *Open*$($*can*$)$

// the agent will perceive the color
// of an open can in view.

Percept Schema

The agent needs to be able to reason about its percepts as it executes the plan.

Percept$(Color(x, c)$,
  PRECOND: *Object*$(x) \wedge InView(x)$

// the agent will perceive the color of
// an object in view.

Percept$(Color(can, c)$,
  PRECOND: *Can*$(can) \wedge Inview(can) \wedge Open(can)$

// the agent will perceive the color
// of an open can in view.

Action$(LookAt(x)$,
  PRECOND: *InView*$(y) \wedge x \neq y$
  EFFECT: *InView*$(x) \wedge \neg InView(y))$

Percept Schema

The agent needs to be able to reason about its percepts as it executes the plan.

Percept$(Color(x, c)$,
    PRECOND: *Object*$(x) \wedge$ *InView*(x)

// the agent will perceive the color of
// an object in view.

Percept$(Color(can, c)$,
    PRECOND: *Can*$(can) \wedge$ *Inview*$(can) \wedge$ *Open*(can)

// the agent will perceive the color
// of an open can in view.

Action$(LookAt(x)$,
    PRECOND: *InView*$(y) \wedge x \neq y$
    EFFECT: *InView*$(x) \wedge \neg$*InView*$(y))$

- Look at the table and chair to get their colors.

Percept Schema

The agent needs to be able to reason about its percepts as it executes the plan.

Percept(*Color*(x, c),
  PRECOND: *Object*(x) $\wedge$ *InView*(x)

// the agent will perceive the color of
// an object in view.

Percept(*Color*(*can*, c),
  PRECOND: *Can*(*can*) $\wedge$ *Inview*(*can*) $\wedge$ *Open*(*can*)

// the agent will perceive the color
// of an open can in view.

Action(*LookAt*(x),
  PRECOND: *InView*(y) $\wedge$ $x \neq y$
  EFFECT: *InView*(x) $\wedge$ $\neg$*InView*(y))

- Look at the table and chair to get their colors.
- If the colors are not the same, open the two paint cans.

Percept Schema

The agent needs to be able to reason about its percepts as it executes the plan.

Percept(*Color*(x, c),
 PRECOND: *Object*(x) $\wedge$ *InView*(x)

// the agent will perceive the color of
// an object in view.

Percept(*Color*(*can*, c),
 PRECOND: *Can*(*can*) $\wedge$ *Inview*(*can*) $\wedge$ *Open*(*can*)

// the agent will perceive the color
// of an open can in view.

Action(*LookAt*(x),
 PRECOND: *InView*(y) $\wedge$ $x \neq y$
 EFFECT: *InView*(x) $\wedge$ $\neg$*InView*(y))

- Look at the table and chair to get their colors.
- If the colors are not the same, open the two paint cans.
  - If the paint in a can has the same color as one furniture piece, apply it to the other piece.

Percept Schema

The agent needs to be able to reason about its percepts as it executes the plan.

Percept($Color(x, c)$),
  PRECOND: *Object*(x) ∧ *InView*(x)

// the agent will perceive the color of
// an object in view.

Percept(*Color*(*can*, c)),
  PRECOND: *Can*(*can*) ∧ *Inview*(*can*) ∧ *Open*(*can*)

// the agent will perceive the color
// of an open can in view.

Action(*LookAt*(x),
  PRECOND: *InView*(y) ∧ $x \neq y$
  EFFECT: *InView*(x) ∧ ¬*InView*(y))

- Look at the table and chair to get their colors.
- If the colors are not the same, open the two paint cans.
  - If the paint in a can has the same color as one furniture piece, apply it to the other piece.
  - Otherwise, paint both pieces with the same color.

Sensorless Planning

The belief state can now be represented as a logical formula instead of a set of enumerated states.

// these facts hold in every belief state.

$Object(Table) \land Object(Chair) \land Can(C_1) \land Can(C_2)$

Sensorless Planning

The belief state can now be represented as a logical formula instead of a set of enumerated states.

// these facts hold in every belief state.

$\mathit{Object}(\mathit{Table}) \wedge \mathit{Object}(\mathit{Chair}) \wedge \mathit{Can}(C_1) \wedge \mathit{Can}(C_2)$

// objects and cans have colors.

$\forall x\ \exists c\ \ \mathit{Color}(x, c)$

Sensorless Planning

The belief state can now be represented as a logical formula instead of a set of enumerated states.

// these facts hold in every belief state.

$Object(Table) \wedge Object(Chair) \wedge Can(C_1) \wedge Can(C_2)$

// objects and cans have colors.

$\forall x \, \exists c \;\; Color(x, c)$

$\Downarrow$

$b_0 = Color(x, C(x))$ Initial belief state

Sensorless Planning

The belief state can now be represented as a logical formula instead of a set of enumerated states.

// these facts hold in every belief state.

$Object(Table) \wedge Object(Chair) \wedge Can(C_1) \wedge Can(C_2)$

// objects and cans have colors.

$\forall x\ \exists c\ \ Color(x, c)$

$\Downarrow$

$b_0 = Color(x, C(x))$ Initial belief state

$\Downarrow$

$[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$ Solution plan

Sensorless Planning

The belief state can now be represented as a logical formula instead of a set of enumerated states.

// these facts hold in every belief state.

$\mathit{Object}(\mathit{Table}) \wedge \mathit{Object}(\mathit{Chair}) \wedge \mathit{Can}(C_1) \wedge \mathit{Can}(C_2)$

// objects and cans have colors.

$\forall x\, \exists c\;\; \mathit{Color}(x, c)$

$\Downarrow$

$b_0 = \mathit{Color}(x, C(x))$ Initial belief state

$\Downarrow$

$[\mathit{RemoveLid}(\mathit{Can}_1), \mathit{Paint}(\mathit{Chair}, \mathit{Can}_1), \mathit{Paint}(\mathit{Table}, \mathit{Can}_1)]$ Solution plan

Open-world assumption:

If a fluent does not appear, its value is unknown (rather than false).

Updating the Belief State

In a belief state b, we consider any action a whose preconditions are satisfied.

$$b' = \text{RESULT}(b, a) = \{s' \mid s' = \text{RESULT}_P\,(s, a) \text{ and } s \in b\}$$

Updating the Belief State

In a belief state b, we consider any action a whose preconditions are satisfied.

$$b' = \textsc{Result}(b, a) = \{s' \mid s' = \textsc{Result}_P\,(s, a) \text{ and } s \in b\}$$

physical transition model

Updating the Belief State

In a belief state b, we consider any action a whose preconditions are satisfied.

$$b' = \text{RESULT}(b, a) = \{s' \mid s' = \text{RESULT}_P(s, a) \text{ and } s \in b\}$$

physical transition model

Consider $b = l_1 \wedge \cdots \wedge l_k$ (1-CNF), where $l_1, \ldots, l_k$ are literals.

Updating the Belief State

In a belief state b, we consider any action a whose preconditions are satisfied.

$$b' = \text{RESULT}(b, a) = \{s' \mid s' = \text{RESULT}_P\,(s, a) \text{ and } s \in b\}$$

physical transition model

Consider $b = l_1 \wedge \cdots \wedge l_k$ (1-CNF), where $l_1, \ldots, l_k$ are literals.

- l_i has known truth value v in b (i.e., in all physical states $s \in b$).

  Compute the truth value in b' from v and the action's add and delete lists.

Updating the Belief State

In a belief state b, we consider any action a whose preconditions are satisfied.

$$b' = \text{RESULT}(b, a) = \{s' \mid s' = \text{RESULT}_P\,(s, a) \text{ and } s \in b\}$$

physical transition model

Consider $b = l_1 \wedge \cdots \wedge l_k$ (1-CNF), where $l_1, \ldots, l_k$ are literals.

- l_i has known truth value v in b (i.e., in all physical states $s \in b$).

  Compute the truth value in b' from v and the action's add and delete lists.

- l_i has unknown truth value in b.

Updating the Belief State

In a belief state b, we consider any action a whose preconditions are satisfied.

$$b' = \textsc{Result}(b, a) = \{s' \mid s' = \textsc{Result}_P\,(s, a) \text{ and } s \in b\}$$

physical transition model

Consider $b = l_1 \wedge \cdots \wedge l_k$ (1-CNF), where $l_1, \ldots, l_k$ are literals.

- l_i has known truth value v in b (i.e., in all physical states $s \in b$).

  Compute the truth value in b' from v and the action's add and delete lists.

- l_i has unknown truth value in b.
  - If a adds l_i , then l_i will be true in b'.

Updating the Belief State

In a belief state b, we consider any action a whose preconditions are satisfied.

$$b' = \text{RESULT}(b, a) = \{s' \mid s' = \text{RESULT}_P\,(s, a) \text{ and } s \in b\}$$

physical transition model

Consider $b = l_1 \wedge \cdots \wedge l_k$ (1-CNF), where $l_1, \ldots, l_k$ are literals.

- l_i has known truth value v in b (i.e., in all physical states $s \in b$).

  Compute the truth value in b' from v and the action's add and delete lists.

- l_i has unknown truth value in b.
  - If a adds l_i , then l_i will be true in b'.
  - If a deletes l_i , then l_i will be false in b'.

Updating the Belief State

In a belief state b, we consider any action a whose preconditions are satisfied.

$$b' = \text{RESULT}(b, a) = \{s' \mid s' = \text{RESULT}_P\,(s, a) \text{ and } s \in b\}$$

physical transition model

Consider $b = l_1 \wedge \cdots \wedge l_k$ (1-CNF), where $l_1, \ldots, l_k$ are literals.

- l_i has known truth value v in b (i.e., in all physical states $s \in b$).

  Compute the truth value in b' from v and the action's add and delete lists.

- l_i has unknown truth value in b.
  - If a adds l_i , then l_i will be true in b'.
  - If a deletes l_i , then l_i will be false in b'.
  - If a does not affect l_i , then l_i will retain its unknown value and not appear in b'.

Updating the Belief State (cont'd)

Calculation of b' is almost the same as in the observable case.

$$b' = \text{RESULT}(b, a) = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

Updating the Belief State (cont'd)

Calculation of b' is almost the same as in the observable case.

$$b' = \text{RESULT}(b, a) = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

$$b_0 = \mathit{Color}(x, C(x))$$

Updating the Belief State (cont'd)

Calculation of b' is almost the same as in the observable case.

$$b' = \text{RESULT}(b, a) = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

$$b_0 = \mathit{Color}(x, C(x))$$

$$\mathit{RemoveLid}(\mathit{Can}_1)$$

Action(*RemoveLid*(*can*),
PRECOND: *Can*(*can*)
EFFECT: *Open*(*can*))

Updating the Belief State (cont'd)

Calculation of b' is almost the same as in the observable case.

$$b' = \text{RESULT}(b, a) = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

$$b_0 = \mathit{Color}(x, C(x))$$

$$\Downarrow \mathit{RemoveLid}(\mathit{Can}_1)$$

$$b_1 = \mathit{Color}(x, C(x)) \land \mathit{Open}(\mathit{Can}_1)$$

Action(*RemoveLid*(*can*),
PRECOND: *Can*(*can*)
EFFECT: *Open*(*can*))

Updating the Belief State (cont'd)

Calculation of b' is almost the same as in the observable case.

$$b' = \textsc{Result}(b, a) = (b - \textsc{Del}(a)) \cup \textsc{Add}(a)$$

$$b_0 = \mathit{Color}(x, C(x))$$

$$\Downarrow \ \mathit{RemoveLid}(\mathit{Can}_1)$$

$$b_1 = \mathit{Color}(x, C(x)) \land \mathit{Open}(\mathit{Can}_1)$$

$\mathit{Paint}(\mathit{Chair}, \mathit{Can}_1)$

$\{x/\ \mathit{Chair}, \text{can}/\ \mathit{Can}_1\}$

Action(*RemoveLid*(*can*),
    PRECOND: *Can*(*can*)
    EFFECT: *Open*(*can*))

Action(*Paint*(x, *can*),
    PRECOND: *Object*(x) ∧ *Can*(*can*) ∧
        Color(*can*, c) ∧ *Open*(*can*)
    EFFECT: *Color*(x, c)

Updating the Belief State (cont'd)

Calculation of b' is almost the same as in the observable case.

$$b' = \text{RESULT}(b, a) = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

$$b_0 = \mathit{Color}(x, C(x))$$

$$\Downarrow \; \mathit{RemoveLid}(\mathit{Can}_1)$$

$$b_1 = \mathit{Color}(x, C(x)) \wedge \mathit{Open}(\mathit{Can}_1)$$

$$\Downarrow \; \mathit{Paint}(\mathit{Chair}, \mathit{Can}_1) \; \{x/\,\mathit{Chair}, \text{can}/\,\mathit{Can}_1\}$$

$$b_2 = \mathit{Color}(x, C(x)) \wedge \mathit{Open}(\mathit{Can}_1) \wedge \mathit{Color}(\mathit{Chair}, C(\mathit{Can}_1))$$

Action(*RemoveLid*(*can*),
PRECOND: *Can*(*can*)
EFFECT: *Open*(*can*))

Action(*Paint*(x, *can*),
PRECOND: *Object*(x) ∧ *Can*(*can*) ∧ *Color*(*can*, c) ∧ *Open*(*can*)
EFFECT: *Color*(x, c)

Updating the Belief State (cont'd)

Calculation of b' is almost the same as in the observable case.

$$b' = \textsc{Result}(b, a) = (b - \textsc{Del}(a)) \cup \textsc{Add}(a)$$

$$b_0 = \mathit{Color}(x, C(x))$$

$$\Downarrow \; \mathit{RemoveLid}(\mathit{Can}_1)$$

$$b_1 = \mathit{Color}(x, C(x)) \wedge \mathit{Open}(\mathit{Can}_1)$$

$$\Downarrow \; \mathit{Paint}(\mathit{Chair}, \mathit{Can}_1) \quad \{x/\,\mathit{Chair},\ \text{can}/\,\mathit{Can}_1\}$$

$$b_2 = \mathit{Color}(x, C(x)) \wedge \mathit{Open}(\mathit{Can}_1) \wedge \mathit{Color}(\mathit{Chair}, C(\mathit{Can}_1))$$

$$\Downarrow \; \mathit{Paint}(\mathit{Table}, \mathit{Can}_1)$$

$$b_3 = \mathit{Color}(x, C(x)) \wedge \mathit{Open}(\mathit{Can}_1) \wedge \mathit{Color}(\mathit{Chair}, C(\mathit{Can}_1)) \wedge \mathit{Color}(\mathit{Table}, C(\mathit{Can}_1))$$

Action(*RemoveLid*(*can*),
PRECOND: *Can*(*can*)
EFFECT: *Open*(*can*))

Action(*Paint*(x, *can*),
PRECOND: *Object*(x) $\wedge$ *Can*(*can*) $\wedge$ *Color*(*can*, c) $\wedge$ *Open*(*can*)
EFFECT: *Color*(x, c))

Updating the Belief State (cont'd)

Calculation of b' is almost the same as in the observable case.

$$b' = \text{RESULT}(b, a) = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

$$b_0 = \mathit{Color}(x, C(x))$$

$$\Downarrow \ \mathit{RemoveLid}(\mathit{Can}_1)$$

Action(*RemoveLid*(*can*),
PRECOND: *Can*(*can*)
EFFECT: *Open*(*can*))

$$b_1 = \mathit{Color}(x, C(x)) \land \mathit{Open}(\mathit{Can}_1)$$

$$\Downarrow \ \mathit{Paint}(\mathit{Chair}, \mathit{Can}_1) \ \{x/\,\mathit{Chair},\ \text{can}/\,\mathit{Can}_1\}$$

Action(*Paint*(x, *can*),
PRECOND: *Object*(x) ∧ *Can*(*can*) ∧
Color(*can*, c) ∧ *Open*(*can*)
EFFECT: *Color*(x, c)

$$b_2 = \mathit{Color}(x, C(x)) \land \mathit{Open}(\mathit{Can}_1) \land \mathit{Color}(\mathit{Chair}, C(\mathit{Can}_1))$$

$$\Downarrow \ \mathit{Paint}(\mathit{Table}, \mathit{Can}_1)$$

$$b_3 = \mathit{Color}(x, C(x)) \land \mathit{Open}(\mathit{Can}_1) \land \mathit{Color}(\mathit{Chair}, C(\mathit{Can}_1)) \land \mathit{Color}(\mathit{Table}, C(\mathit{Can}_1))$$

$$\mathit{Goal}(\mathit{Color}(\mathit{Chair}, c) \land \mathit{Color}(\mathit{Table}, c))$$

Updating the Belief State (cont'd)

Calculation of b' is almost the same as in the observable case.

$$b' = \textsc{Result}(b, a) = (b - \textsc{Del}(a)) \cup \textsc{Add}(a)$$

$b_0 = \mathit{Color}(x, C(x))$

$\Downarrow$ $\mathit{RemoveLid}(\mathit{Can}_1)$

$b_1 = \mathit{Color}(x, C(x)) \wedge \mathit{Open}(\mathit{Can}_1)$

$\Downarrow$ $\mathit{Paint}(\mathit{Chair}, \mathit{Can}_1)$ $\{x/\ \mathit{Chair}, \text{can}/\ \mathit{Can}_1\}$

$b_2 = \mathit{Color}(x, C(x)) \wedge \mathit{Open}(\mathit{Can}_1) \wedge \mathit{Color}(\mathit{Chair}, C(\mathit{Can}_1))$

$\Downarrow$ $\mathit{Paint}(\mathit{Table}, \mathit{Can}_1)$

$b_3 = \mathit{Color}(x, C(x)) \wedge \mathit{Open}(\mathit{Can}_1) \wedge \mathit{Color}(\mathit{Chair}, C(\mathit{Can}_1)) \wedge \mathit{Color}(\mathit{Table}, C(\mathit{Can}_1))$

$\mathit{Action}(\mathit{RemoveLid}(\mathit{can}),$
  $\textsc{Precond}$: $\mathit{Can}(\mathit{can})$
  $\textsc{Effect}$: $\mathit{Open}(\mathit{can}))$

$\mathit{Action}(\mathit{Paint}(x, \mathit{can}),$
  $\textsc{Precond}$: $\mathit{Object}(x) \wedge \mathit{Can}(\mathit{can}) \wedge$
    $\mathit{Color}(\mathit{can}, c) \wedge \mathit{Open}(\mathit{can})$
  $\textsc{Effect}$: $\mathit{Color}(x, c)$

$\mathit{Goal}(\mathit{Color}(\mathit{Chair}, c) \wedge \mathit{Color}(\mathit{Table}, c))$

satisfied under substitution $\{\ c\ /\ C(\mathit{Can}_1)\}$!

Contingent Planning

Generate a plan with conditional branching based on percepts.

A conditional solution:

[*LookAt*(*Table*), *LookAt*(*Chair*),
 if *Color*(*Table*, c) $\wedge$ *Color*(*Chair*, c) then *NoOp*
  else [*RemoveLid* (Can_1), *LookAt* (Can_1), *RemoveLid*(Can_2), *LookAt* (Can_2),
   if *Color*(*Table*, c) $\wedge$ *Color*(*can*, c) then *Paint*(*Chair*, *can*)
   else if *Color*(*Chair*, c) $\wedge$ *Color*(*can*, c) then *Paint*(*Table*, *can*)
   else [*Paint*(*Chair*, Can_1), *Paint*(*Table*, Can_1)]]]

Contingent Planning

Generate a plan with conditional branching based on percepts.

A conditional solution:

[$LookAt(Table)$, $LookAt(Chair)$,
 if $Color(Table, c) \wedge Color(Chair, c)$ then $NoOp$
  else [$RemoveLid\ (Can_1)$, $LookAt\ (Can_1)$, $RemoveLid(Can_2)$, $LookAt\ (Can_2)$,
   if $Color(Table, c) \wedge Color(can, c)$ then $Paint(Chair, can)$
   else if $Color(Chair, c) \wedge Color(can, c)$ then $Paint(Table, can)$
   else [$Paint(Chair, Can_1)$, $Paint(Table, Can_1)$]]]

- ♣ Variables in the solution are existentially quantified.

Contingent Planning

Generate a plan with conditional branching based on percepts.

A conditional solution:

[*LookAt*(*Table*), *LookAt*(*Chair*),
 if *Color*(*Table*, c) $\wedge$ *Color*(*Chair*, c) then *NoOp*
 else [*RemoveLid* (Can_1), *LookAt* (Can_1), *RemoveLid*(Can_2), *LookAt* (Can_2),
 if *Color*(*Table*, c) $\wedge$ *Color*(*can*, c) then *Paint*(*Chair*, *can*)
 else if *Color*(*Chair*, c) $\wedge$ *Color*(*can*, c) then *Paint*(*Table*, *can*)
 else [*Paint*(*Chair*, Can_1), *Paint*(*Table*, Can_1)]]]

- ♣ Variables in the solution are existentially quantified.
- ♣ A conditional formula can be satisfied in multiple ways depending on variable substitutions.

Contingent Planning

Generate a plan with conditional branching based on percepts.

A conditional solution:

[*LookAt*(*Table*), *LookAt*(*Chair*),
if *Color*(*Table*, c) $\wedge$ *Color*(*Chair*, c) then *NoOp*
else [*RemoveLid* (*Can*$_1$), *LookAt* (*Can*$_1$), *RemoveLid*(*Can*$_2$), *LookAt* (*Can*$_2$),
if *Color*(*Table*, c) $\wedge$ *Color*(*can*, c) then *Paint*(*Chair*, *can*)
else if *Color*(*Chair*, c) $\wedge$ *Color*(*can*, c) then *Paint*(*Table*, *can*)
else [*Paint*(*Chair*, *Can*$_1$), *Paint*(*Table*, *Can*$_1$)]]]

- ♣ Variables in the solution are existentially quantified.
- ♣ A conditional formula can be satisfied in multiple ways depending on variable substitutions.
- ♣ The planning algorithm has to avoid a belief state in which the condition's truth value is unknown.

How to Update the Belief State?

Contingent planning carries out the following two steps for each action:

- Calculate the belief state ($b = l_1 \wedge \cdots \wedge l_k$) after the action.

$$\hat{b} = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

How to Update the Belief State?

Contingent planning carries out the following two steps for each action:

- Calculate the belief state ($b = l_1 \wedge \cdots \wedge l_k$) after the action.

$$\hat{b} = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

- Add percept literals $p_1, \ldots, p_k$.

How to Update the Belief State?

Contingent planning carries out the following two steps for each action:

- Calculate the belief state ($b = l_1 \wedge \cdots \wedge l_k$) after the action.

$$\hat{b} = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

- Add percept literals $p_1, \ldots, p_k$.

  - If some p_i, $1 \le i \le k$, has one percept schema, then add all the literals in the preconditions of the schema.

Percept(*Color*(*can*, c),
PRECOND: *Can*(*can*) $\wedge$ *Inview*(*can*)
$\wedge$ *Open*(*can*))

How to Update the Belief State?

Contingent planning carries out the following two steps for each action:

- Calculate the belief state ($b = l_1 \wedge \cdots \wedge l_k$) after the action.

$$\hat{b} = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

- Add percept literals $p_1, \ldots, p_k$.
  - If some p_i, $1 \leq i \leq k$, has one percept schema, then add all the literals in the preconditions of the schema.

Percept(*Color*(*can*, c),
PRECOND: *Can*(*can*) ∧ *Inview*(*can*)
∧ *Open*(*can*)

  - If p_i has more than one percept schema, then add the disjunction of the preconditions of these schemas.
    - $\hat{b}$ will be no longer a 1-CNF.
    - Nevertheless, same complications as conditional effects.

How to Update the Belief State?

Contingent planning carries out the following two steps for each action:

- Calculate the belief state ($b = l_1 \wedge \cdots \wedge l_k$) after the action.

$$\hat{b} = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

- Add percept literals $p_1, \ldots, p_k$.

  - If some p_i, $1 \leq i \leq k$, has one percept schema, then add all the literals in the preconditions of the schema.

Percept(*Color*(*can*, c),
PRECOND: *Can*(*can*) ∧ *Inview*(*can*)
∧ *Open*(*can*))

  - If p_i has more than one percept schema, then add the disjunction of the preconditions of these schemas.
    - $\hat{b}$ will be no longer a 1-CNF.
    - Nevertheless, same complications as conditional effects.