Fundamentals of Artificial Intelligence
Chapter 11: **Planning in the Real World**

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
http://disi.unitn.it/rseba/DIDATTICA/fai_2022/

Teaching assistant: Mauro Dragoni – dragoni@fbk.eu
http://www.maurodragoni.com/teaching/fai/

M.S. Course "Artificial Intelligence Systems", academic year 2022-2023

Last update: Friday 25^{th} November, 2022, 11:49

Outline

1. Time, Schedules & Resources
2. Planning & Acting in Non-Determistic Domains
   - Generalities
   - Sensorless Planning (aka Conformant Planning)
   - Conditional Planning (aka Contingent Planning)

Outline

Planning with Time, Schedules and Resources

- Planning so far: choice of actions
- Real world: Planning with time/schedules
  - actions occur at certain moments in time
  - actions have a beginning and an end
  - actions have a duration

  $\Longrightarrow$ Scheduling
- Real world: Planning with resources
  - actions may require resources
  - ex: limited number of staff, planes, hoists, ...
- Preconditions and effects can include
  - logical inferences
  - numeric computations
  - interactions with other software packages
- Approach “plan first, schedule later”:
  - planning phase: build a (partial) plan, regardless action durations
  - scheduling phase: add temporal info to the plan, s.t. to meet resource and deadline constrains

Planning with Time, Schedules and Resources

- Planning so far: choice of actions
- Real world: Planning with time/schedules
  - actions occur at certain moments in time
  - actions have a beginning and an end
  - actions have a duration

  $\Longrightarrow$ Scheduling
- Real world: Planning with resources
  - actions may require resources
  - ex: limited number of staff, planes, hoists, ...
- Preconditions and effects can include
  - logical inferences
  - numeric computations
  - interactions with other software packages
- Approach "plan first, schedule later":
  - planning phase: build a (partial) plan, regardless action durations
  - scheduling phase: add temporal info to the plan, s.t. to meet resource and deadline constrains

Planning with Time, Schedules and Resources

- Planning so far: choice of actions
- Real world: Planning with time/schedules
  - actions occur at certain moments in time
  - actions have a beginning and an end
  - actions have a duration

  $\Longrightarrow$ Scheduling
- Real world: Planning with resources
  - actions may require resources
  - ex: limited number of staff, planes, hoists, ...
- Preconditions and effects can include
  - logical inferences
  - numeric computations
  - interactions with other software packages
- Approach “plan first, schedule later”:
  - planning phase: build a (partial) plan, regardless action durations
  - scheduling phase: add temporal info to the plan, s.t. to meet resource and deadline constrains

Planning with Time, Schedules and Resources

- Planning so far: choice of actions
- Real world: Planning with time/schedules
  - actions occur at certain moments in time
  - actions have a beginning and an end
  - actions have a duration
  - $\Longrightarrow$ Scheduling
- Real world: Planning with resources
  - actions may require resources
  - ex: limited number of staff, planes, hoists, ...
- Preconditions and effects can include
  - logical inferences
  - numeric computations
  - interactions with other software packages
- Approach "plan first, schedule later":
  - planning phase: build a (partial) plan, regardless action durations
  - scheduling phase: add temporal info to the plan, s.t. to meet resource and deadline constrains

Planning with Time, Schedules and Resources

- Planning so far: choice of actions
- Real world: Planning with time/schedules
  - actions occur at certain moments in time
  - actions have a beginning and an end
  - actions have a duration

  $\Longrightarrow$ Scheduling
- Real world: Planning with resources
  - actions may require resources
  - ex: limited number of staff, planes, hoists, ...
- Preconditions and effects can include
  - logical inferences
  - numeric computations
  - interactions with other software packages
- Approach “plan first, schedule later”:
  - planning phase: build a (partial) plan, regardless action durations
  - scheduling phase: add temporal info to the plan, s.t. to meet resource and deadline constrains

Planning with Time & Resources: Example

Planning Phase

$$
\begin{array}{l}
\mathit{Init}(\mathit{Chassis}(C1) \land \mathit{Chassis}(C2) \land \mathit{Engine}(E1, C1, 30) \land \\
\quad \mathit{Engine}(E1, C2, 60) \land \mathit{Wheels}(W1, C1, 30) \land \mathit{Wheels}(W2, C2, 15)) \\
\mathit{Goal}(\mathit{Done}(C1) \land \mathit{Done}(C2)) \\
\mathit{Action}(\mathit{AddEngine}(e, c, d) \\
\quad \mathit{PRECOND} : \mathit{Engine}(e, c, d) \land \mathit{Chassis}(c) \land \neg \mathit{EngineIn}(c) \\
\quad \mathit{EFFECT} : \mathit{EngineIn}(c) \land \mathit{Duration}(d) \\
\quad \mathit{Consume} : \mathit{LugNuts}(20), \mathit{Use} : \mathit{EngineHoists}(1)) \\
\mathit{Action}(\mathit{AddWheels}(w, c, d) \\
\quad \mathit{PRECOND} : \mathit{Wheels}(w, c, d) \land \mathit{Chassis}(c) \\
\quad \mathit{EFFECT} : \mathit{WheelsOn}(c) \land \mathit{Duration}(d) \\
\quad \mathit{Consume} : \mathit{LugNuts}(20), \mathit{Use} : \mathit{WheelStations}(1)) \\
\mathit{Action}(\mathit{Inspect}(c, 10) \\
\quad \mathit{PRECOND} : \mathit{EngineIn}(c) \land \mathit{WheelsOn}(c) \land \mathit{Chassis}(c) \\
\quad \mathit{EFFECT} : \mathit{Done}(c) \land \mathit{Duration}(10) \\
\quad \mathit{Use} : \mathit{Inspectors}(1))
\end{array}
$$

Solution (partial plan):

$$
\left\{
\begin{array}{l}
\mathit{AddEngine}(E1, C1, 30) \prec \mathit{AddWheels}(W1, C1, 30) \prec \mathit{Inspect}(C1, 10); \\
\mathit{AddEngine}(E2, C2, 60) \prec \mathit{AddWheels}(W2, C2, 15) \prec \mathit{Inspect}(C2, 10)
\end{array}
\right\}
$$

Planning with Time & Resources: Example

Planning Phase

$Init(Chassis(C1) \wedge Chassis(C2) \wedge Engine(E1, C1, 30) \wedge$
$\quad Engine(E1, C2, 60) \wedge Wheels(W1, C1, 30) \wedge Wheels(W2, C2, 15))$
$Goal(Done(C1) \wedge Done(C2))$
$Action(AddEngine(e, c, d)$
$\quad PRECOND: Engine(e, c, d) \wedge Chassis(c) \wedge \neg EngineIn(c)$
$\quad EFFECT: EngineIn(c) \wedge Duration(d)$
$\quad Consume: LugNuts(20), Use: EngineHoists(1))$
$Action(AddWheels(w, c, d)$
$\quad PRECOND: Wheels(w, c, d) \wedge Chassis(c)$
$\quad EFFECT: WheelsOn(c) \wedge Duration(d)$
$\quad Consume: LugNuts(20), Use: WheelStations(1))$
$Action(Inspect(c, 10)$
$\quad PRECOND: EngineIn(c) \wedge WheelsOn(c) \wedge Chassis(c)$
$\quad EFFECT: Done(c) \wedge Duration(10)$
$\quad Use: Inspectors(1))$

Solution (partial plan):

$$\left\{ \begin{array}{l} AddEngine(E1, C1, 30) \prec AddWheels(W1, C1, 30) \prec Inspect(C1, 10); \\ AddEngine(E2, C2, 60) \prec AddWheels(W2, C2, 15) \prec Inspect(C2, 10) \end{array} \right\}$$

Planning with Time & Resources: Example

Planning Phase

$$
\begin{aligned}
&Init(Chassis(C1) \land Chassis(C2) \land Engine(E1, C1, 30) \land \\
&\quad Engine(E1, C2, 60) \land Wheels(W1, C1, 30) \land Wheels(W2, C2, 15)) \\
&Goal(Done(C1) \land Done(C2)) \\
&Action(AddEngine(e, c, d) \\
&\quad PRECOND: Engine(e, c, d) \land Chassis(c) \land \neg EngineIn(c) \\
&\quad EFFECT: EngineIn(c) \land Duration(d) \\
&\quad Consume: LugNuts(20), Use: EngineHoists(1)) \\
&Action(AddWheels(w, c, d) \\
&\quad PRECOND: Wheels(w, c, d) \land Chassis(c) \\
&\quad EFFECT: WheelsOn(c) \land Duration(d) \\
&\quad Consume: LugNuts(20), Use: WheelStations(1)) \\
&Action(Inspect(c, 10) \\
&\quad PRECOND: EngineIn(c) \land WheelsOn(c) \land Chassis(c) \\
&\quad EFFECT: Done(c) \land Duration(10) \\
&\quad Use: Inspectors(1))
\end{aligned}
$$

Solution (partial plan):

$$
\left\{ \begin{array}{l} AddEngine(E1, C1, 30) \prec AddWheels(W1, C1, 30) \prec Inspect(C1, 10); \\ AddEngine(E2, C2, 60) \prec AddWheels(W2, C2, 15) \prec Inspect(C2, 10) \end{array} \right\}
$$

Planning with Time & Resources: Example

Planning Phase

$$
\begin{aligned}
&\mathit{Init}(\mathit{Chassis}(C1) \wedge \mathit{Chassis}(C2) \wedge \mathit{Engine}(E1, C1, 30) \wedge \\
&\quad \mathit{Engine}(E1, C2, 60) \wedge \mathit{Wheels}(W1, C1, 30) \wedge \mathit{Wheels}(W2, C2, 15)) \\
&\mathit{Goal}(\mathit{Done}(C1) \wedge \mathit{Done}(C2)) \\
&\mathit{Action}(\mathit{AddEngine}(e, c, d) \\
&\; \mathit{PRECOND} : \mathit{Engine}(e, c, d) \wedge \mathit{Chassis}(c) \wedge \neg \mathit{EngineIn}(c) \\
&\; \mathit{EFFECT} : \mathit{EngineIn}(c) \wedge \mathit{Duration}(d) \\
&\; \mathit{Consume} : \mathit{LugNuts}(20), \mathit{Use} : \mathit{EngineHoists}(1)) \\
&\mathit{Action}(\mathit{AddWheels}(w, c, d) \\
&\; \mathit{PRECOND} : \mathit{Wheels}(w, c, d) \wedge \mathit{Chassis}(c) \\
&\; \mathit{EFFECT} : \mathit{WheelsOn}(c) \wedge \mathit{Duration}(d) \\
&\; \mathit{Consume} : \mathit{LugNuts}(20), \mathit{Use} : \mathit{WheelStations}(1)) \\
&\mathit{Action}(\mathit{Inspect}(c, 10) \\
&\; \mathit{PRECOND} : \mathit{EngineIn}(c) \wedge \mathit{WheelsOn}(c) \wedge \mathit{Chassis}(c) \\
&\; \mathit{EFFECT} : \mathit{Done}(c) \wedge \mathit{Duration}(10) \\
&\; \mathit{Use} : \mathit{Inspectors}(1))
\end{aligned}
$$

Solution (partial plan):

$$
\left\{
\begin{array}{l}
\mathit{AddEngine}(E1, C1, 30) \prec \mathit{AddWheels}(W1, C1, 30) \prec \mathit{Inspect}(C1, 10); \\
\mathit{AddEngine}(E2, C2, 60) \prec \mathit{AddWheels}(W2, C2, 15) \prec \mathit{Inspect}(C2, 10)
\end{array}
\right\}
$$

Job-Shop Scheduling

- Problem:
  - complete a set of jobs,
  - a job consists of a collection of actions with ordering constraints
  - an action has a duration and is subject to resource constraints
  - resource constraints specify
    - the type of resource (e.g., bolts, wrenches, or pilots),
    - the number of that resource required
    - if the resource is consumable (e.g., bolts) or reusable (e.g. pilot)
    - resources can be produced by actions with negative consumption
- Solution (aka Schedule):
  - specify the start times for each action
  - must satisfy all the temporal ordering constraints and resource constraints
- Cost function
  - may be very complicate (e.g. non-linear constraints)
  - we assume is the total duration of the plan (makespan)

$\Longrightarrow$ Determine a schedule that minimizes the makespan, respecting all temporal and resource constraints

Job-Shop Scheduling

- Problem:
  - complete a set of jobs,
  - a job consists of a collection of actions with ordering constraints
  - an action has a duration and is subject to resource constraints
  - resource constraints specify
    - the type of resource (e.g., bolts, wrenches, or pilots),
    - the number of that resource required
    - if the resource is consumable (e.g., bolts) or reusable (e.g. pilot)
    - resources can be produced by actions with negative consumption
- Solution (aka Schedule):
  - specify the start times for each action
  - must satisfy all the temporal ordering constraints and resource constraints
- Cost function
  - may be very complicate (e.g. non-linear constraints)
  - we assume is the total duration of the plan (makespan)

$\Longrightarrow$ Determine a schedule that minimizes the makespan, respecting all temporal and resource constraints

Job-Shop Scheduling

- Problem:
  - complete a set of jobs,
  - a job consists of a collection of actions with ordering constraints
  - an action has a duration and is subject to resource constraints
  - resource constraints specify
    - the type of resource (e.g., bolts, wrenches, or pilots),
    - the number of that resource required
    - if the resource is consumable (e.g., bolts) or reusable (e.g. pilot)
    - resources can be produced by actions with negative consumption
- Solution (aka Schedule):
  - specify the start times for each action
  - must satisfy all the temporal ordering constraints and resource constraints
- Cost function
  - may be very complicate (e.g. non-linear constraints)
  - we assume is the total duration of the plan (makespan)

$\Longrightarrow$ Determine a schedule that minimizes the makespan, respecting all temporal and resource constraints

Job-Shop Scheduling

- Problem:
  - complete a set of jobs,
  - a job consists of a collection of actions with ordering constraints
  - an action has a duration and is subject to resource constraints
  - resource constraints specify
    - the type of resource (e.g., bolts, wrenches, or pilots),
    - the number of that resource required
    - if the resource is consumable (e.g., bolts) or reusable (e.g. pilot)
    - resources can be produced by actions with negative consumption
- Solution (aka Schedule):
  - specify the start times for each action
  - must satisfy all the temporal ordering constraints and resource constraints
- Cost function
  - may be very complicate (e.g. non-linear constraints)
  - we assume is the total duration of the plan (makespan)

$\Longrightarrow$ Determine a schedule that minimizes the makespan, respecting all temporal and resource constraints

Solving Scheduling Problems

Critical-Path Method

- A path is a ordered sequence of actions from Start to Finish
- The critical path is the path with maximum total duration
  - delaying the start of any action on it slows down the whole plan

  $\Longrightarrow$ determines the duration of the entire plan
  - shortening other paths does not shorten the plan as a whole
- Actions have a window of time in which they can be executed: $[ES, LS]$
  - ES: earliest possible start time
  - LS: latest possible start time
  - LS-ES: slack of the action
- LS & ES for all actions can be computed recursively:

$$
\begin{aligned}
ES(Start) &= 0\\
ES(B) &= max_{\{A \mid A \prec B\}}(ES(A) + Duration(A))\\
LS(Finish) &= ES(Finish)\\
LS(A) &= min_{\{B \mid A \prec B\}}(LS(B) - Duration(A))
\end{aligned}
$$

- Action A_i in the critical path are s.t. $ES(A_i) = LS(A_i)$
- Complexity: $O(Nb)$, N: #actions, b: max branching factor

Solving Scheduling Problems

Critical-Path Method

- A path is a ordered sequence of actions from Start to Finish
- The critical path is the path with maximum total duration
  - delaying the start of any action on it slows down the whole plan

  $\Longrightarrow$ determines the duration of the entire plan
  - shortening other paths does not shorten the plan as a whole
- Actions have a window of time in which they can be executed: $[ES, LS]$
  - ES: earliest possible start time
  - LS: latest possible start time
  - LS-ES: slack of the action
- LS & ES for all actions can be computed recursively:

$$
\begin{aligned}
ES(Start) &= 0 \\
ES(B) &= max_{\{A \mid A \prec B\}}(ES(A) + Duration(A)) \\
LS(Finish) &= ES(Finish) \\
LS(A) &= min_{\{B \mid A \prec B\}}(LS(B) - Duration(A))
\end{aligned}
$$

- Action A_i in the critical path are s.t. $ES(A_i) = LS(A_i)$
- Complexity: $O(Nb)$, N: #actions, b: max branching factor

Solving Scheduling Problems

Critical-Path Method

- A path is a ordered sequence of actions from Start to Finish
- The critical path is the path with maximum total duration
  - delaying the start of any action on it slows down the whole plan

  $\Longrightarrow$ determines the duration of the entire plan
  - shortening other paths does not shorten the plan as a whole
- Actions have a window of time in which they can be executed: $[ES, LS]$
  - ES: earliest possible start time
  - LS: latest possible start time
  - LS-ES: slack of the action
- LS & ES for all actions can be computed recursively:

$$\begin{aligned}
ES(Start) &= 0 \\
ES(B) &= max_{\{A \mid A \prec B\}}(ES(A) + Duration(A)) \\
LS(Finish) &= ES(Finish) \\
LS(A) &= min_{\{B \mid A \prec B\}}(LS(B) - Duration(A))
\end{aligned}$$

- Action A_i in the critical path are s.t. $ES(A_i) = LS(A_i)$
- Complexity: $O(Nb)$, N: #actions, b: max branching factor

Solving Scheduling Problems

Critical-Path Method

- A path is a ordered sequence of actions from Start to Finish
- The critical path is the path with maximum total duration
  - delaying the start of any action on it slows down the whole plan

  $\Longrightarrow$ determines the duration of the entire plan
  - shortening other paths does not shorten the plan as a whole
- Actions have a window of time in which they can be executed: $[ES, LS]$
  - ES: earliest possible start time
  - LS: latest possible start time
  - LS-ES: slack of the action
- LS & ES for all actions can be computed recursively:

$$\begin{aligned} ES(Start) &= 0 \\ ES(B) &= max_{\{A \mid A \prec B\}}(ES(A) + Duration(A)) \\ LS(Finish) &= ES(Finish) \\ LS(A) &= min_{\{B \mid A \prec B\}}(LS(B) - Duration(A)) \end{aligned}$$

- Action A_i in the critical path are s.t. $ES(A_i) = LS(A_i)$
- Complexity: $O(Nb)$, N: #actions, b: max branching factor

Solving Scheduling Problems

Critical-Path Method

- A path is a ordered sequence of actions from Start to Finish
- The critical path is the path with maximum total duration
  - delaying the start of any action on it slows down the whole plan

  $\Longrightarrow$ determines the duration of the entire plan
  - shortening other paths does not shorten the plan as a whole
- Actions have a window of time in which they can be executed: $[ES, LS]$
  - ES: earliest possible start time
  - LS: latest possible start time
  - LS-ES: slack of the action
- LS & ES for all actions can be computed recursively:

$$\begin{aligned} ES(Start) &= 0 \\ ES(B) &= max_{\{A \mid A \prec B\}}(ES(A) + Duration(A)) \\ LS(Finish) &= ES(Finish) \\ LS(A) &= min_{\{B \mid A \prec B\}}(LS(B) - Duration(A)) \end{aligned}$$

- Action A_i in the critical path are s.t. $ES(A_i) = LS(A_i)$
- Complexity: $O(Nb)$, N: #actions, b: max branching factor

Solving Scheduling Problems

Critical-Path Method

- A path is a ordered sequence of actions from Start to Finish
- The critical path is the path with maximum total duration
  - delaying the start of any action on it slows down the whole plan

  $\Longrightarrow$ determines the duration of the entire plan
  - shortening other paths does not shorten the plan as a whole
- Actions have a window of time in which they can be executed: $[ES, LS]$
  - ES: earliest possible start time
  - LS: latest possible start time
  - LS-ES: slack of the action
- LS & ES for all actions can be computed recursively:

$$
\begin{aligned}
ES(Start) &= 0 \\
ES(B) &= max_{\{A \mid A \prec B\}}(ES(A) + Duration(A)) \\
LS(Finish) &= ES(Finish) \\
LS(A) &= min_{\{B \mid A \prec B\}}(LS(B) - Duration(A))
\end{aligned}
$$

- Action A_i in the critical path are s.t. $ES(A_i) = LS(A_i)$
- Complexity: $O(Nb)$, N: #actions, b: max branching factor

Solving Scheduling Problems

Critical-Path Method

- A path is a ordered sequence of actions from Start to Finish
- The critical path is the path with maximum total duration
  - delaying the start of any action on it slows down the whole plan

  $\Longrightarrow$ determines the duration of the entire plan
  - shortening other paths does not shorten the plan as a whole
- Actions have a window of time in which they can be executed: $[ES, LS]$
  - ES: earliest possible start time
  - LS: latest possible start time
  - LS-ES: slack of the action
- LS & ES for all actions can be computed recursively:

$$\begin{aligned}
ES(Start) &= 0 \\
ES(B) &= max_{\{A \mid A \prec B\}}(ES(A) + Duration(A)) \\
LS(Finish) &= ES(Finish) \\
LS(A) &= min_{\{B \mid A \prec B\}}(LS(B) - Duration(A))
\end{aligned}$$

- Action A_i in the critical path are s.t. $ES(A_i) = LS(A_i)$
- Complexity: $O(Nb)$, N: #actions, b: max branching factor

Planning with Time & Resources: Example [cont.]

Scheduling Phase

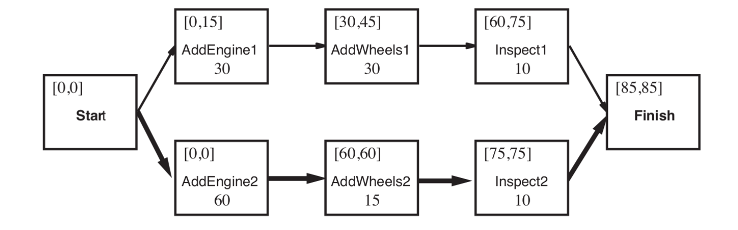
$$
\begin{aligned}
&\mathit{Jobs}(\{\mathit{AddEngine1} \prec \mathit{AddWheels1} \prec \mathit{Inspect1}\},\\
&\qquad \{\mathit{AddEngine2} \prec \mathit{AddWheels2} \prec \mathit{Inspect2}\})
\end{aligned}
$$

$$
\mathit{Resources}(\mathit{EngineHoists}(1), \mathit{WheelStations}(1), \mathit{Inspectors}(2), \mathit{LugNuts}(500))
$$

$$
\begin{aligned}
&\mathit{Action}(\mathit{AddEngine1}, \textsc{Duration}{:}30,\\
&\qquad \textsc{Use}{:}\mathit{EngineHoists}(1))\\
&\mathit{Action}(\mathit{AddEngine2}, \textsc{Duration}{:}60,\\
&\qquad \textsc{Use}{:}\mathit{EngineHoists}(1))\\
&\mathit{Action}(\mathit{AddWheels1}, \textsc{Duration}{:}30,\\
&\qquad \textsc{Consume}{:}\mathit{LugNuts}(20), \textsc{Use}{:}\ \mathit{WheelStations}(1))\\
&\mathit{Action}(\mathit{AddWheels2}, \textsc{Duration}{:}15,\\
&\qquad \textsc{Consume}{:}\mathit{LugNuts}(20), \textsc{Use}{:}\ \mathit{WheelStations}(1))\\
&\mathit{Action}(\mathit{Inspect}_i, \textsc{Duration}{:}10,\\
&\qquad \textsc{Use}{:}\mathit{Inspectors}(1))
\end{aligned}
$$

Planning with Time & Resources: Example [cont.]

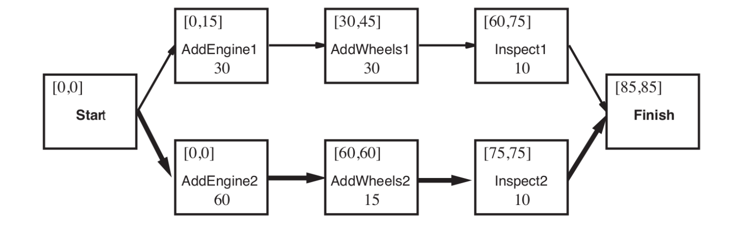
Scheduling Phase

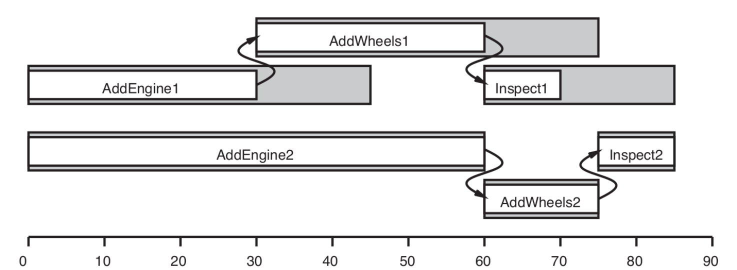


(© S. Russell & P. Norwig, AIMA)

Planning with Time & Resources: Example [cont.]

Scheduling Phase





(© S. Russell & P. Norwig, AIMA)

Adding Resources

- Critical-path problems (without resources) computationally easy:
  - conjunction of linear inequalities on the start and end times:
    ex: $(ES_2 \geq ES_1 + duration_1) \wedge (ES_3 \geq ES_2 + duration_2) \wedge ...$

  $\Longrightarrow$ Polynomial: $O(Nb)$, N: number of actions; b: maximum branching factor in/out of an action
- Reusable resources: R(k) (ex: Use: EngineHoists(1))
  - k units of resource are required by the action.
  - availability is a pre-requisite before the action can be performed.
- Adding resources makes problems much harder
  - "cannot overlap" constraint is disjunction of linear inequalities
    ex: $((ES_2 \geq ES_1 + duration_1) \vee (ES_1 \geq ES_2 + duration_2)) \wedge ...$

  $\Longrightarrow$ NP-hard
- Various techniques:
  - branch-and-bound, simulated annealing, tabu search, ...
  - reduction to constraint optimization problems
  - reduction to optimization modulo theories (combined SAT+LP)
- Integrate planning and scheduling

Adding Resources

- Critical-path problems (without resources) computationally easy:
  - conjunction of linear inequalities on the start and end times:
    ex: $(ES_2 \geq ES_1 + duration_1) \wedge (ES_3 \geq ES_2 + duration_2) \wedge \ldots$

  $\Longrightarrow$ Polynomial: $O(Nb)$, N: number of actions; b: maximum branching factor in/out of an action
- Reusable resources: R(k) (ex: Use: EngineHoists(1))
  - k units of resource are required by the action.
  - availability is a pre-requisite before the action can be performed.
- Adding resources makes problems much harder
  - "cannot overlap" constraint is disjunction of linear inequalities
    ex: $((ES_2 \geq ES_1 + duration_1) \vee (ES_1 \geq ES_2 + duration_2)) \wedge \ldots$

  $\Longrightarrow$ NP-hard
- Various techniques:
  - branch-and-bound, simulated annealing, tabu search, ...
  - reduction to constraint optimization problems
  - reduction to optimization modulo theories (combined SAT+LP)
- Integrate planning and scheduling

Adding Resources

- Critical-path problems (without resources) computationally easy:
  - conjunction of linear inequalities on the start and end times:
    ex: $(ES_2 \geq ES_1 + duration_1) \wedge (ES_3 \geq ES_2 + duration_2) \wedge \ldots$

  $\Longrightarrow$ Polynomial: $O(Nb)$, N: number of actions; b: maximum branching factor in/out of an action
- Reusable resources: R(k) (ex: Use: EngineHoists(1))
  - k units of resource are required by the action.
  - availability is a pre-requisite before the action can be performed.
- Adding resources makes problems much harder
  - "cannot overlap" constraint is disjunction of linear inequalities
    ex: $((ES_2 \geq ES_1 + duration_1) \vee (ES_1 \geq ES_2 + duration_2)) \wedge \ldots$

  $\Longrightarrow$ NP-hard
- Various techniques:
  - branch-and-bound, simulated annealing, tabu search, ...
  - reduction to constraint optimization problems
  - reduction to optimization modulo theories (combined SAT+LP)
- Integrate planning and scheduling

Adding Resources

- Critical-path problems (without resources) computationally easy:
  - conjunction of linear inequalities on the start and end times:
    ex: $(ES_2 \geq ES_1 + duration_1) \wedge (ES_3 \geq ES_2 + duration_2) \wedge \ldots$

  $\Longrightarrow$ Polynomial: $O(Nb)$, N: number of actions; b: maximum branching factor in/out of an action
- Reusable resources: R(k) (ex: Use: EngineHoists(1))
  - k units of resource are required by the action.
  - availability is a pre-requisite before the action can be performed.
- Adding resources makes problems much harder
  - "cannot overlap" constraint is disjunction of linear inequalities
    ex: $((ES_2 \geq ES_1 + duration_1) \vee (ES_1 \geq ES_2 + duration_2)) \wedge \ldots$

  $\Longrightarrow$ NP-hard
- Various techniques:
  - branch-and-bound, simulated annealing, tabu search, ...
  - reduction to constraint optimization problems
  - reduction to optimization modulo theories (combined SAT+LP)
- Integrate planning and scheduling

Adding Resources

- Critical-path problems (without resources) computationally easy:
  - conjunction of linear inequalities on the start and end times:
    ex: $(ES_2 \geq ES_1 + duration_1) \wedge (ES_3 \geq ES_2 + duration_2) \wedge ...$

  $\Longrightarrow$ Polynomial: $O(Nb)$, N: number of actions; b: maximum branching factor in/out of an action
- Reusable resources: R(k) (ex: Use: EngineHoists(1))
  - k units of resource are required by the action.
  - availability is a pre-requisite before the action can be performed.
- Adding resources makes problems much harder
  - "cannot overlap" constraint is disjunction of linear inequalities
    ex: $((ES_2 \geq ES_1 + duration_1) \vee (ES_1 \geq ES_2 + duration_2)) \wedge ...$

  $\Longrightarrow$ NP-hard
- Various techniques:
  - branch-and-bound, simulated annealing, tabu search, ...
  - reduction to constraint optimization problems
  - reduction to optimization modulo theories (combined SAT+LP)
- Integrate planning and scheduling

Adding Resources

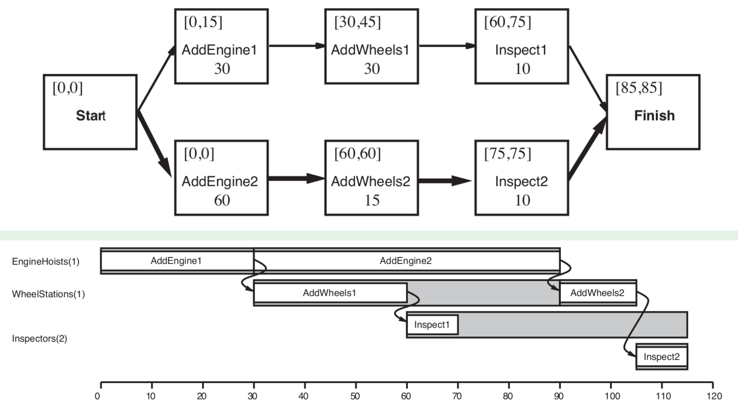
- Critical-path problems (without resources) computationally easy:
  - conjunction of linear inequalities on the start and end times:
    ex: $(ES_2 \geq ES_1 + duration_1) \wedge (ES_3 \geq ES_2 + duration_2) \wedge ...$

  $\Longrightarrow$ Polynomial: $O(Nb)$, N: number of actions; b: maximum branching factor in/out of an action
- Reusable resources: R(k) (ex: Use: EngineHoists(1))
  - k units of resource are required by the action.
  - availability is a pre-requisite before the action can be performed.
- Adding resources makes problems much harder
  - “cannot overlap” constraint is disjunction of linear inequalities
    ex: $((ES_2 \geq ES_1 + duration_1) \vee (ES_1 \geq ES_2 + duration_2)) \wedge ...$

  $\Longrightarrow$ NP-hard
- Various techniques:
  - branch-and-bound, simulated annealing, tabu search, ...
  - reduction to constraint optimization problems
  - reduction to optimization modulo theories (combined SAT+LP)
- Integrate planning and scheduling

Planning with Time & Resources: Example [cont.]

Scheduling Phase



(© S. Russell & P. Norwig, AIMA)

- left-hand margin lists the three reusable resources
- two possible schedules: which assembly uses the hoist first
- shortest-duration solution, which takes 115 minutes

Exercise

- Consider the previous example
  - find another solution
  - draw the diagram
  - check its length and compare it with that in the previous slide

Outline

Outline

Generalities [also recall Ch.04]

- Assumptions so far:
  - the environment is deterministic
  - the environment is fully observable
  - the environment is static
  - the agent knows the effects of each action

$\Longrightarrow$ The agent does not need perception:
  - can calculate which state results from any sequence of actions
  - always knows which state it is in

- In the real world, the environment may be uncertain
  - partially observable and/or nondeterministic environment
  - incorrect information (differences between world and model)

$\Longrightarrow$ If one of the above assumptions does not hold, use percepts
  - the agent's future actions will depend on future percepts
  - the future percepts cannot be determined in advance

- Use percepts:
  - perceive the changes in the world
  - act accordingly
  - adapt plan when necessary

Generalities [also recall Ch.04]

- Assumptions so far:
  - the environment is deterministic
  - the environment is fully observable
  - the environment is static
  - the agent knows the effects of each action

$\Longrightarrow$ The agent does not need perception:
  - can calculate which state results from any sequence of actions
  - always knows which state it is in

- In the real world, the environment may be uncertain
  - partially observable and/or nondeterministic environment
  - incorrect information (differences between world and model)

$\Longrightarrow$ If one of the above assumptions does not hold, use percepts
  - the agent's future actions will depend on future percepts
  - the future percepts cannot be determined in advance

- Use percepts:
  - perceive the changes in the world
  - act accordingly
  - adapt plan when necessary

Generalities [also recall Ch.04]

- Assumptions so far:
  - the environment is deterministic
  - the environment is fully observable
  - the environment is static
  - the agent knows the effects of each action

$\Longrightarrow$ The agent does not need perception:
  - can calculate which state results from any sequence of actions
  - always knows which state it is in
- In the real world, the environment may be uncertain
  - partially observable and/or nondeterministic environment
  - incorrect information (differences between world and model)

$\Longrightarrow$ If one of the above assumptions does not hold, use percepts
  - the agent's future actions will depend on future percepts
  - the future percepts cannot be determined in advance
- Use percepts:
  - perceive the changes in the world
  - act accordingly
  - adapt plan when necessary

Generalities [also recall Ch.04]

- Assumptions so far:
  - the environment is deterministic
  - the environment is fully observable
  - the environment is static
  - the agent knows the effects of each action

$\Longrightarrow$ The agent does not need perception:
  - can calculate which state results from any sequence of actions
  - always knows which state it is in

- In the real world, the environment may be uncertain
  - partially observable and/or nondeterministic environment
  - incorrect information (differences between world and model)

$\Longrightarrow$ If one of the above assumptions does not hold, use percepts
  - the agent's future actions will depend on future percepts
  - the future percepts cannot be determined in advance

- Use percepts:
  - perceive the changes in the world
  - act accordingly
  - adapt plan when necessary

Generalities [also recall Ch.04]

- Assumptions so far:
  - the environment is deterministic
  - the environment is fully observable
  - the environment is static
  - the agent knows the effects of each action

$\Longrightarrow$ The agent does not need perception:

  - can calculate which state results from any sequence of actions
  - always knows which state it is in
- In the real world, the environment may be uncertain
  - partially observable and/or nondeterministic environment
  - incorrect information (differences between world and model)

$\Longrightarrow$ If one of the above assumptions does not hold, use percepts

  - the agent's future actions will depend on future percepts
  - the future percepts cannot be determined in advance
- Use percepts:
  - perceive the changes in the world
  - act accordingly
  - adapt plan when necessary

Handling Indeterminacy

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances (if any)
  - regardless of initial state and action effects
  - for environments with no observations
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments

Differences wrt. general Search (Ch.04)

- planners deal with factored representations rather than atomic
- different representation of actions and observation
- different representation of belief states

Handling Indeterminacy

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances (if any)
  - regardless of initial state and action effects
  - for environments with no observations
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments

Differences wrt. general Search (Ch.04)

- planners deal with factored representations rather than atomic
- different representation of actions and observation
- different representation of belief states

Handling Indeterminacy

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances (if any)
  - regardless of initial state and action effects
  - for environments with no observations
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments

Differences wrt. general Search (Ch.04)

- planners deal with factored representations rather than atomic
- different representation of actions and observation
- different representation of belief states

Handling Indeterminacy

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances (if any)
  - regardless of initial state and action effects
  - for environments with no observations
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments

Differences wrt. general Search (Ch.04)

- planners deal with factored representations rather than atomic
- different representation of actions and observation
- different representation of belief states

Handling Indeterminacy [cont.]

Open-World vs. Closed-World Assumption

- Classical Planning based on Closed-World Assumption (CWA)
  - states contain only positive fluents
  - we assume that every fluent not mentioned in a state is false
- Sensorless & Partially-observable Planning based on Open-World Assumption (OWA)
  - states contain both positive and negative fluents
  - if a fluent does not appear in the state, its value is unknown

Belief States

- A belief state is represented by a logical formula (not an explicitly-enumerated set of states)

$\Longrightarrow$ The belief state corresponds exactly to the set of possible worlds that satisfy the formula representing it

- The unknown information can be retrieved via sensing actions (aka percept actions) added to the plan

Handling Indeterminacy [cont.]

Open-World vs. Closed-World Assumption

- Classical Planning based on Closed-World Assumption (CWA)
  - states contain only positive fluents
  - we assume that every fluent not mentioned in a state is false
- Sensorless & Partially-observable Planning based on Open-World Assumption (OWA)
  - states contain both positive and negative fluents
  - if a fluent does not appear in the state, its value is unknown

Belief States

- A belief state is represented by a logical formula (not an explicitly-enumerated set of states)

$\Longrightarrow$ The belief state corresponds exactly to the set of possible worlds that satisfy the formula representing it

- The unknown information can be retrieved via sensing actions (aka percept actions) added to the plan

Handling Indeterminacy [cont.]

Open-World vs. Closed-World Assumption

- Classical Planning based on Closed-World Assumption (CWA)
  - states contain only positive fluents
  - we assume that every fluent not mentioned in a state is false
- Sensorless & Partially-observable Planning based on Open-World Assumption (OWA)
  - states contain both positive and negative fluents
  - if a fluent does not appear in the state, its value is unknown

Belief States

- A belief state is represented by a logical formula (not an explicitly-enumerated set of states)

$\Longrightarrow$ The belief state corresponds exactly to the set of possible worlds that satisfy the formula representing it

- The unknown information can be retrieved via sensing actions (aka percept actions) added to the plan

Handling Indeterminacy [cont.]

Open-World vs. Closed-World Assumption

- Classical Planning based on Closed-World Assumption (CWA)
  - states contain only positive fluents
  - we assume that every fluent not mentioned in a state is false
- Sensorless & Partially-observable Planning based on Open-World Assumption (OWA)
  - states contain both positive and negative fluents
  - if a fluent does not appear in the state, its value is unknown

Belief States

- A belief state is represented by a logical formula (not an explicitly-enumerated set of states)

$\Longrightarrow$ The belief state corresponds exactly to the set of possible worlds that satisfy the formula representing it

- The unknown information can be retrieved via sensing actions (aka percept actions) added to the plan

Handling Indeterminacy [cont.]

Open-World vs. Closed-World Assumption

- Classical Planning based on Closed-World Assumption (CWA)
  - states contain only positive fluents
  - we assume that every fluent not mentioned in a state is false
- Sensorless & Partially-observable Planning based on Open-World Assumption (OWA)
  - states contain both positive and negative fluents
  - if a fluent does not appear in the state, its value is unknown

Belief States

- A belief state is represented by a logical formula (not an explicitly-enumerated set of states)

$\Longrightarrow$ The belief state corresponds exactly to the set of possible worlds that satisfy the formula representing it

- The unknown information can be retrieved via sensing actions (aka percept actions) added to the plan

A Case Study

The table & chair painting problem

Given a chair and a table, the goal is to have them of the same color.
Initially we have two cans of paint, but the colors of the paint and of the furniture are unknown.
Only the table is initially in the agent's field of view

A Case Study [cont.]

The table & chair painting problem [cont.]

- Initial state:

$Init(Object(Table) \wedge Object(Chair) \wedge Can(C1) \wedge Can(C2) \wedge InView(Table))$

- Goal: $Goal(Color(Chair, c) \wedge Color(Table, c))$
  - recall: in goal, variable c existentially quantified
- Actions:

  $Action(RemoveLid(can),$
  $\quad Precond : Can(can)$
  $\quad Effect : Open(can))$
  $Action(Paint(x, can),$
  $\quad Precond : Object(x) \wedge Can(can) \wedge Color(can, c) \wedge Open(can)$
  $\quad Effect : Color(x, c))$

  c not part of action's variable list (partially observable only)
- Add an action causing objects to come into view (one at a time):

  $Action(LookAt(x),$
  $\quad Precond : InView(y) \wedge (x \neq y)$
  $\quad Effect : InView(x) \wedge \neg InView(y))$

A Case Study [cont.]

The table & chair painting problem [cont.]

- Initial state:

$Init(Object(Table) \land Object(Chair) \land Can(C1) \land Can(C2) \land InView(Table))$

- Goal: $Goal(Color(Chair, c) \land Color(Table, c))$
  - recall: in goal, variable c existentially quantified
- Actions:

  $Action(RemoveLid(can),$
  $\quad Precond : Can(can)$
  $\quad Effect : Open(can))$
  $Action(Paint(x, can),$
  $\quad Precond : Object(x) \land Can(can) \land Color(can, c) \land Open(can)$
  $\quad Effect : Color(x, c))$

  c not part of action's variable list (partially observable only)
- Add an action causing objects to come into view (one at a time):

  $Action(LookAt(x),$
  $\quad Precond : InView(y) \land (x \neq y)$
  $\quad Effect : InView(x) \land \neg InView(y))$

A Case Study [cont.]

The table & chair painting problem [cont.]

- Initial state:

$Init(Object(Table) \land Object(Chair) \land Can(C1) \land Can(C2) \land InView(Table))$

- Goal: $Goal(Color(Chair, c) \land Color(Table, c))$
  - recall: in goal, variable c existentially quantified
- Actions:

  $Action(RemoveLid(can),$
  $\quad Precond : Can(can)$
  $\quad Effect : Open(can))$
  $Action(Paint(x, can),$
  $\quad Precond : Object(x) \land Can(can) \land Color(can, c) \land Open(can)$
  $\quad Effect : Color(x, c))$

  c not part of action's variable list (partially observable only)
- Add an action causing objects to come into view (one at a time):

  $Action(LookAt(x),$
  $\quad Precond : InView(y) \land (x \neq y)$
  $\quad Effect : InView(x) \land \neg InView(y))$

A Case Study [cont.]

The table & chair painting problem [cont.]

- Initial state:

$Init(Object(Table) \land Object(Chair) \land Can(C1) \land Can(C2) \land InView(Table))$

- Goal: $Goal(Color(Chair, c) \land Color(Table, c))$
  - recall: in goal, variable c existentially quantified
- Actions:

  $Action(RemoveLid(can),$
  $\quad Precond : Can(can)$
  $\quad Effect : Open(can))$
  $Action(Paint(x, can),$
  $\quad Precond : Object(x) \land Can(can) \land Color(can, c) \land Open(can)$
  $\quad Effect : Color(x, c))$

  c not part of action's variable list (partially observable only)

- Add an action causing objects to come into view (one at a time):

  $Action(LookAt(x),$
  $\quad Precond : InView(y) \land (x \neq y)$
  $\quad Effect : InView(x) \land \neg InView(y))$

A Case Study [cont.]

The table & chair painting problem [cont.]

- Initial state:

$Init(Object(Table) \land Object(Chair) \land Can(C1) \land Can(C2) \land InView(Table))$

- Goal: $Goal(Color(Chair, c) \land Color(Table, c))$
  - recall: in goal, variable c existentially quantified
- Actions:

  $Action(RemoveLid(can),$
  $\quad Precond: Can(can)$
  $\quad Effect: Open(can))$
  $Action(Paint(x, can),$
  $\quad Precond: Object(x) \land Can(can) \land Color(can, c) \land Open(can)$
  $\quad Effect: Color(x, c))$

  c not part of action's variable list (partially observable only)
- Add an action causing objects to come into view (one at a time):

  $Action(LookAt(x),$
  $\quad Precond: InView(y) \land (x \neq y)$
  $\quad Effect: InView(x) \land \neg InView(y))$

A Case Study [cont.]

The table & chair painting problem [cont.]

- Partially-Observable Problems:
  need to reason about percepts obtained during action

$\Longrightarrow$ Augment PDDL with percept schemata $Percept(\langle fluent\rangle, Precond : \langle fluents\rangle))$ for each fluent. Ex:

- $Percept(Color(x, c),$
  $Precond : Object(x) \wedge InView(x))$
  "if an object is in view, then the agent will perceive its color"

  $\Longrightarrow$ perception will acquire the truth value of $Color(x, c)$, for every x, c
- $Percept(Color(can, c),$
  $Precond : Can(can) \wedge InView(can) \wedge Open(can))$
  "if an open can is in view, then the agent perceives the color of the paint in the can"

  $\Longrightarrow$ perception will acquire the truth value of $Color(can, c)$, for every can, c

- Fully-Observable Problems:

$\Longrightarrow$ Percept schemata with no preconditions for each fluent. Ex:

- $Percept(Color(x, c))$

- Sensorless Agent: no percept schema

A Case Study [cont.]

The table & chair painting problem [cont.]

- Partially-Observable Problems:
  need to reason about percepts obtained during action

$\Longrightarrow$ Augment PDDL with percept schemata $Percept(\langle fluent\rangle, Precond : \langle fluents\rangle))$ for each fluent. Ex:

  - $Percept(Color(x, c),$
    $Precond : Object(x) \land InView(x))$
    "if an object is in view, then the agent will perceive its color"

  $\Longrightarrow$ perception will acquire the truth value of $Color(x, c)$, for every x, c
  - $Percept(Color(can, c),$
    $Precond : Can(can) \land InView(can) \land Open(can))$
    "if an open can is in view, then the agent perceives the color of the paint in the can"

  $\Longrightarrow$ perception will acquire the truth value of $Color(can, c)$, for every can, c

- Fully-Observable Problems:

$\Longrightarrow$ Percept schemata with no preconditions for each fluent. Ex:

  - $Percept(Color(x, c))$

- Sensorless Agent: no percept schema

A Case Study [cont.]

The table & chair painting problem [cont.]

- Partially-Observable Problems:
  need to reason about percepts obtained during action

$\Longrightarrow$ Augment PDDL with percept schemata $Percept(\langle fluent \rangle, Precond : \langle fluents \rangle))$ for each fluent. Ex:

  - $Percept(Color(x, c),$
    $Precond : Object(x) \wedge InView(x))$
    "if an object is in view, then the agent will perceive its color"

  $\Longrightarrow$ perception will acquire the truth value of $Color(x, c)$, for every x, c

  - $Percept(Color(can, c),$
    $Precond : Can(can) \wedge InView(can) \wedge Open(can))$
    "if an open can is in view, then the agent perceives the color of the paint in the can"

  $\Longrightarrow$ perception will acquire the truth value of $Color(can, c)$, for every can, c

- Fully-Observable Problems:

$\Longrightarrow$ Percept schemata with no preconditions for each fluent. Ex:

  - $Percept(Color(x, c))$

- Sensorless Agent: no percept schema

A Case Study [cont.]

The table & chair painting problem [cont.]

- Partially-Observable Problems:
  need to reason about percepts obtained during action

$\Longrightarrow$ Augment PDDL with percept schemata $Percept(\langle fluent \rangle, Precond : \langle fluents \rangle))$ for each fluent. Ex:

- $Percept(Color(x, c),$
  $Precond : Object(x) \wedge InView(x))$
  "if an object is in view, then the agent will perceive its color"

$\Longrightarrow$ perception will acquire the truth value of $Color(x, c)$, for every x, c

- $Percept(Color(can, c),$
  $Precond : Can(can) \wedge InView(can) \wedge Open(can))$
  "if an open can is in view, then the agent perceives the color of the paint in the can"

$\Longrightarrow$ perception will acquire the truth value of $Color(can, c)$, for every can, c

- Fully-Observable Problems:

$\Longrightarrow$ Percept schemata with no preconditions for each fluent. Ex:

- $Percept(Color(x, c))$

- Sensorless Agent: no percept schema

A Case Study [cont.]

The table & chair painting problem [cont.]

- **Partially-Observable Problems**:
  need to reason about percepts obtained during action

$\Longrightarrow$ Augment PDDL with percept schemata $Percept(\langle fluent \rangle, Precond : \langle fluents \rangle))$ for each fluent. Ex:

  - $Percept(Color(x, c),$
    $Precond : Object(x) \land InView(x))$
    "if an object is in view, then the agent will perceive its color"

  $\Longrightarrow$ perception will acquire the truth value of $Color(x, c)$, for every x, c

  - $Percept(Color(can, c),$
    $Precond : Can(can) \land InView(can) \land Open(can))$
    "if an open can is in view, then the agent perceives the color of the paint in the can"

  $\Longrightarrow$ perception will acquire the truth value of $Color(can, c)$, for every can, c

- **Fully-Observable Problems**:

$\Longrightarrow$ Percept schemata with no preconditions for each fluent. Ex:

  - $Percept(Color(x, c))$

- Sensorless Agent: no percept schema

A Case Study [cont.]

The table & chair painting problem [cont.]

- Partially-Observable Problems:
  need to reason about percepts obtained during action

$\Longrightarrow$ Augment PDDL with percept schemata $Percept(\langle fluent\rangle, Precond : \langle fluents\rangle))$ for each fluent. Ex:

- $Percept(Color(x, c),$
  $Precond : Object(x) \land InView(x))$
  "if an object is in view, then the agent will perceive its color"

$\Longrightarrow$ perception will acquire the truth value of $Color(x, c)$, for every x, c

- $Percept(Color(can, c),$
  $Precond : Can(can) \land InView(can) \land Open(can))$
  "if an open can is in view, then the agent perceives the color of the paint in the can"

$\Longrightarrow$ perception will acquire the truth value of $Color(can, c)$, for every can, c

- Fully-Observable Problems:

$\Longrightarrow$ Percept schemata with no preconditions for each fluent. Ex:

- $Percept(Color(x, c))$

- Sensorless Agent: no percept schema

Handling Indeterminacy [cont.]

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances (if any)
  - regardless of initial state and action effects
  - for environments with no observations
  - ex: "Open any can of paint and apply it to both chair and table"
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
  - ex: "Sense color of table and chair;
    if they are the same, then finish, else sense can paint;
    if color(can) =color(furniture) then apply color to other piece;
    else apply color to both"
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments
  - ex: Same as conditional, and can fix errors

Handling Indeterminacy [cont.]

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances (if any)
  - regardless of initial state and action effects
  - for environments with no observations
  - ex: "Open any can of paint and apply it to both chair and table"
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
  - ex: "Sense color of table and chair;
    if they are the same, then finish, else sense can paint;
    if color(can) =color(furniture) then apply color to other piece;
    else apply color to both"
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments
  - ex: Same as conditional, and can fix errors

Handling Indeterminacy [cont.]

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances (if any)
  - regardless of initial state and action effects
  - for environments with no observations
  - ex: “Open any can of paint and apply it to both chair and table”
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
  - ex: “Sense color of table and chair;
    if they are the same, then finish, else sense can paint;
    if color(can) =color(furniture) then apply color to other piece;
    else apply color to both”
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments
  - ex: Same as conditional, and can fix errors

Handling Indeterminacy [cont.]

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances (if any)
  - regardless of initial state and action effects
  - for environments with no observations
  - ex: "Open any can of paint and apply it to both chair and table"
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
  - ex: "Sense color of table and chair;
    if they are the same, then finish, else sense can paint;
    if color(can) =color(furniture) then apply color to other piece;
    else apply color to both"
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments
  - ex: Same as conditional, and can fix errors

Handling Indeterminacy [cont.]

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances (if any)
  - regardless of initial state and action effects
  - for environments with no observations
  - ex: "Open any can of paint and apply it to both chair and table"
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
  - ex: "Sense color of table and chair;
    if they are the same, then finish, else sense can paint;
    if color(can) =color(furniture) then apply color to other piece;
    else apply color to both"
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments
  - ex: Same as conditional, and can fix errors

Outline

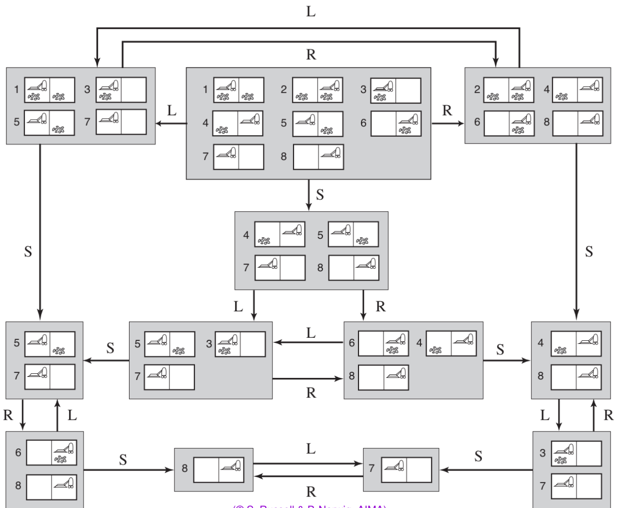
[Recall from Ch.04]: Search with No Observation

Search with No Observation

- aka Sensorless Search or Conformant Search
- Idea: To solve sensorless problems, the agent searches in the space of belief states rather than in that of physical states
  - fully observable, because the agent knows its own belief space
  - solutions are always sequences of actions (no contingency plan), because percepts are always empty and thus predictable
- Main drawback: 2^N candidate states rather than N

Example: Sensorless Vacuum Cleaner: Belief State Space

(note: self-loops are omitted)

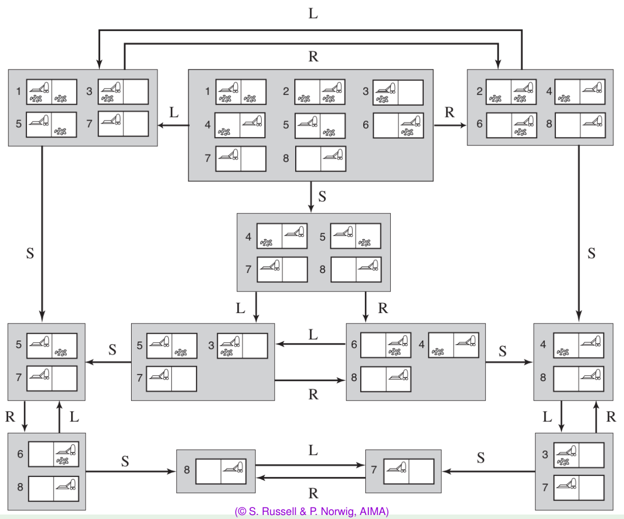


(© S. Russell & P. Norwig, AIMA)

$\Longrightarrow$ [Left,Suck,Right,Suck] contingent plan

Example: Sensorless Vacuum Cleaner: Belief State Space

(note: self-loops are omitted)



(© S. Russell & P. Norwig, AIMA)

$\Longrightarrow$ [Left,Suck,Right,Suck] contingent plan

Sensorless Planning

- Main idea [see ch.04]: see a sensorless planning problem as a belief-state planning problem
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
- Open-World Assumption $\Longrightarrow$ a belief state corresponds to the set of possible worlds that satisfy the formula representing it
- All belief states (implicitly) include unchanging facts (invariants)
  ex: $Object(Table) \land Object(Chair) \land Can(C_1) \land Can(C_2)$
- Initial belief state includes facts that part of the agent's domain knowledge
  - Ex: "objects and cans have colors"
    $\forall x.\exists c.\ Color(x, c) \Longrightarrow$ (Skolemization) $\Longrightarrow b_0 : Color(x, C(x))$

Sensorless Planning

- Main idea [see ch.04]: see a sensorless planning problem as a belief-state planning problem
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
- Open-World Assumption $\Longrightarrow$ a belief state corresponds to the set of possible worlds that satisfy the formula representing it
- All belief states (implicitly) include unchanging facts (invariants)
  ex: $Object(Table) \land Object(Chair) \land Can(C_1) \land Can(C_2)$
- Initial belief state includes facts that part of the agent's domain knowledge
  - Ex: "objects and cans have colors"
    $\forall x.\exists c.\ Color(x, c) \Longrightarrow$ (Skolemization) $\Longrightarrow b_0 : Color(x, C(x))$

Sensorless Planning

- Main idea [see ch.04]: see a sensorless planning problem as a belief-state planning problem
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
- Open-World Assumption $\Longrightarrow$ a belief state corresponds to the set of possible worlds that satisfy the formula representing it
- All belief states (implicitly) include unchanging facts (invariants)
  ex: $Object(Table) \land Object(Chair) \land Can(C_1) \land Can(C_2)$
- Initial belief state includes facts that part of the agent's domain knowledge
  - Ex: "objects and cans have colors"
    $\forall x.\exists c.\ Color(x, c) \Longrightarrow$ (Skolemization) $\Longrightarrow b_0 : Color(x, C(x))$

Sensorless Planning

- Main idea [see ch.04]: see a sensorless planning problem as a belief-state planning problem
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
- Open-World Assumption $\Longrightarrow$ a belief state corresponds to the set of possible worlds that satisfy the formula representing it
- All belief states (implicitly) include unchanging facts (invariants)
  ex: $Object(Table) \land Object(Chair) \land Can(C_1) \land Can(C_2)$
- Initial belief state includes facts that part of the agent's domain knowledge
  - Ex: "objects and cans have colors"
    $\forall x.\exists c.\ Color(x, c) \Longrightarrow$ (Skolemization) $\Longrightarrow b_0 : Color(x, C(x))$

Sensorless Planning [cont.]

- In belief state b, it is possible to apply every action a s.t. $b \models Precond(a)$
  - e.g., $RemoveLid(Can_1)$ applicable in b_0 since $Can(C_1)$ true in b_0
- $Result(b, a)$ is computed:
  - start from b
  - set to false any atom that appears in $Del(a)$ (after unification)
  - set to true any atom that appears in $Add(a)$ (after unification)

  i.e., conjoin Effects(a) to b

Property

If the belief state starts as a conjunction of literals, then any update will yield a conjunction of literals

- with n fluents, any belief state can be compactly represented by a conjunction of size O(n)

$\Longrightarrow$ much simplifies complexity of belief-state reasoning

Sensorless Planning [cont.]

- In belief state b, it is possible to apply every action a s.t. $b \models Precond(a)$
  - e.g., $RemoveLid(Can_1)$ applicable in b_0 since $Can(C_1)$ true in b_0
- $Result(b, a)$ is computed:
  - start from b
  - set to false any atom that appears in $Del(a)$ (after unification)
  - set to true any atom that appears in $Add(a)$ (after unification)

  i.e., conjoin Effects(a) to b

Property

If the belief state starts as a conjunction of literals, then any update will yield a conjunction of literals

- with n fluents, any belief state can be compactly represented by a conjunction of size O(n)

$\Longrightarrow$ much simplifies complexity of belief-state reasoning

Sensorless Planning [cont.]

- In belief state b, it is possible to apply every action a s.t. $b \models Precond(a)$
  - e.g., $RemoveLid(Can_1)$ applicable in b_0 since $Can(C_1)$ true in b_0
- $Result(b, a)$ is computed:
  - start from b
  - set to false any atom that appears in $Del(a)$ (after unification)
  - set to true any atom that appears in $Add(a)$ (after unification)

  i.e., conjoin Effects(a) to b

Property

If the belief state starts as a conjunction of literals, then any update will yield a conjunction of literals

- with n fluents, any belief state can be compactly represented by a conjunction of size O(n)

$\Longrightarrow$ much simplifies complexity of belief-state reasoning

Sensorless Planning: Example

- Start from $b_0 : Color(x, C(x))$
- Apply $RemoveLid(Can_1)$ in b_0 and obtain:
  $b_1 : Color(x, C(x)) \wedge Open(Can_1)$
- Apply $Paint(Chair, Can_1)$ in b_1 using $\{x/Chair, c/C(Can_1)\}$:
  $b_2 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1))$
- Apply $Paint(Table, Can_1)$ in b_2 :
  $b_3 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1)) \wedge Color(Table, C(Can_1))$
- b_3 Satisfies the goal: $b_3 \models Color(Table, c) \wedge Color(Chair, c)$

$\Longrightarrow$ $[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$
valid conformant plan

Sensorless Planning: Example

- Start from $b_0 : Color(x, C(x))$
- Apply $RemoveLid(Can_1)$ in b_0 and obtain:
  $b_1 : Color(x, C(x)) \wedge Open(Can_1)$
- Apply $Paint(Chair, Can_1)$ in b_1 using $\{x/Chair, c/C(Can_1)\}$:
  $b_2 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1))$
- Apply $Paint(Table, Can_1)$ in b_2 :
  $b_3 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1)) \wedge Color(Table, C(Can_1))$
- b_3 Satisfies the goal: $b_3 \models Color(Table, c) \wedge Color(Chair, c)$

$\Longrightarrow$ $[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$
valid conformant plan

Sensorless Planning: Example

- Start from b_0 : $Color(x, C(x))$
- Apply $RemoveLid(Can_1)$ in b_0 and obtain:
  b_1 : $Color(x, C(x)) \land Open(Can_1)$
- Apply $Paint(Chair, Can_1)$ in b_1 using $\{x/Chair, c/C(Can_1)\}$:
  b_2 : $Color(x, C(x)) \land Open(Can_1) \land Color(Chair, C(Can_1))$
- Apply $Paint(Table, Can_1)$ in b_2 :
  b_3 : $Color(x, C(x)) \land Open(Can_1) \land Color(Chair, C(Can_1)) \land Color(Table, C(Can_1))$
- b_3 Satisfies the goal: $b_3 \models Color(Table, c) \land Color(Chair, c)$

$\Longrightarrow$ $[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$
valid conformant plan

Sensorless Planning: Example

- Start from $b_0 : Color(x, C(x))$
- Apply $RemoveLid(Can_1)$ in b_0 and obtain:
  $b_1 : Color(x, C(x)) \wedge Open(Can_1)$
- Apply $Paint(Chair, Can_1)$ in b_1 using $\{x/Chair, c/C(Can_1)\}$:
  $b_2 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1))$
- Apply $Paint(Table, Can_1)$ in b_2 :
  $b_3 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1)) \wedge Color(Table, C(Can_1))$
- b_3 Satisfies the goal: $b_3 \models Color(Table, c) \wedge Color(Chair, c)$

$\Longrightarrow$ $[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$
valid conformant plan

Sensorless Planning: Example

- Start from $b_0 : Color(x, C(x))$
- Apply $RemoveLid(Can_1)$ in b_0 and obtain:
  $b_1 : Color(x, C(x)) \land Open(Can_1)$
- Apply $Paint(Chair, Can_1)$ in b_1 using $\{x/Chair, c/C(Can_1)\}$:
  $b_2 : Color(x, C(x)) \land Open(Can_1) \land Color(Chair, C(Can_1))$
- Apply $Paint(Table, Can_1)$ in b_2 :
  $b_3 : Color(x, C(x)) \land Open(Can_1) \land Color(Chair, C(Can_1)) \land Color(Table, C(Can_1))$
- b_3 Satisfies the goal: $b_3 \models Color(Table, c) \land Color(Chair, c)$

$\Longrightarrow$ $[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$
valid conformant plan

Sensorless Planning: Example

- Start from $b_0 : Color(x, C(x))$
- Apply $RemoveLid(Can_1)$ in b_0 and obtain:
  $b_1 : Color(x, C(x)) \land Open(Can_1)$
- Apply $Paint(Chair, Can_1)$ in b_1 using $\{x/Chair, c/C(Can_1)\}$:
  $b_2 : Color(x, C(x)) \land Open(Can_1) \land Color(Chair, C(Can_1))$
- Apply $Paint(Table, Can_1)$ in b_2 :
  $b_3 : Color(x, C(x)) \land Open(Can_1) \land Color(Chair, C(Can_1)) \land Color(Table, C(Can_1))$
- b_3 Satisfies the goal: $b_3 \models Color(Table, c) \land Color(Chair, c)$

$\Longrightarrow$ $[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$
valid conformant plan

Sensorless Planning: Example

- Start from $b_0 : Color(x, C(x))$
- Apply $RemoveLid(Can_1)$ in b_0 and obtain:
  $b_1 : Color(x, C(x)) \wedge Open(Can_1)$
- Apply $Paint(Chair, Can_1)$ in b_1 using $\{x/Chair, c/C(Can_1)\}$:
  $b_2 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1))$
- Apply $Paint(Table, Can_1)$ in b_2 :
  $b_3 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1)) \wedge Color(Table, C(Can_1))$
- b_3 Satisfies the goal: $b_3 \models Color(Table, c) \wedge Color(Chair, c)$

$\Longrightarrow$ $[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$
valid conformant plan

Exercise

- Provide a novel formalization of the above problem with distinct predicates for the color of an object and for the color the paint in a can
  - find step-by-step a plan with the new formalization

Outline

[Recall from Ch.4]: Searching with Nondeterministic Actions

Generalized notion of transition model

- RESULTS(S,A) returns a set of possible outcomes states
  - Ex: RESULTS(1,SUCK)=$\{5, 7\}$, RESULTS(5,SUCK)=$\{1, 5\}$, ...
- A solution is a contingency plan (aka conditional plan, strategy)
  - contains nested conditions on future percepts (if-then-else, case-switch, ...)
  - Ex: from state 1 we can act the following contingency plan:
    [SUCK, IF STATE = 5 THEN [RIGHT, SUCK] ELSE [ ]]
- Can cause loops (see later)

[Recall from Ch.4]: Searching with Nondeterministic Actions [cont.]

And-Or Search Trees

- In a deterministic environment, branching on agent's choices

  $\Longrightarrow$ OR nodes, hence OR search trees
  - OR nodes correspond to states
- In a nondeterministic environment, branching also on environment's choice of outcome for each action
  - the agent has to handle all such outcomes

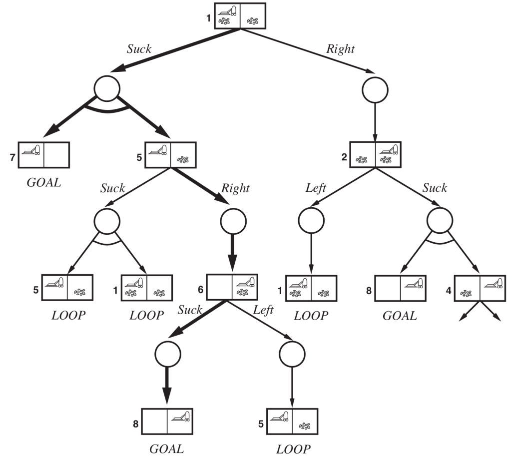
  $\Longrightarrow$ AND nodes, hence AND-OR search trees
  - AND nodes correspond to actions
  - leaf nodes are goal, dead-end or loop OR nodes
- A solution for an AND-OR search problem is a subtree s.t.:
  - has a goal node at every leaf
  - specifies one action at each of its OR nodes
  - includes all outcome branches at each of its AND nodes

OR tree: AND-OR tree with 1 outcome each AND node (determinism)

[Recall from Ch.4]: And-Or Search Trees: Example

(Part of) And-Or Search Tree for Erratic Vacuum Cleaner Example.

Solution for [SUCK, IF STATE = 5 THEN [RIGHT, SUCK] ELSE [ ]]



(© S. Russell & P. Norwig, AIMA)

[Recall from Ch.4]: AND-OR Search

Recursive Depth-First (Tree-based) AND-OR Search

function AND-OR-GRAPH-SEARCH(*problem*) **returns** *a conditional plan, or failure*
  OR-SEARCH(*problem*.INITIAL-STATE, *problem*, [ ])

---

function OR-SEARCH(*state*, *problem*, *path*) **returns** *a conditional plan, or failure*
  if *problem*.GOAL-TEST(*state*) **then return** the empty plan
  if *state* is on *path* **then return** *failure*
  for each *action* **in** *problem*.ACTIONS(*state*) **do**
    plan ← AND-SEARCH(RESULTS(*state*, *action*), *problem*, [*state* | *path*])
    if *plan* ≠ *failure* **then return** [*action* | *plan*]
  return *failure*

---

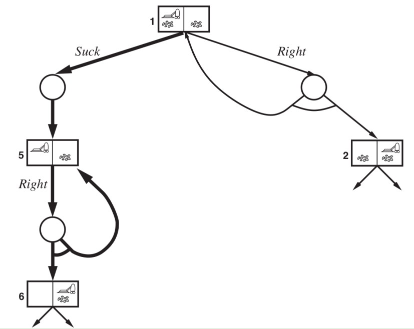
function AND-SEARCH(*states*, *problem*, *path*) **returns** *a conditional plan, or failure*
  for each s_i **in** *states* **do**
    $plan_i \leftarrow$ OR-SEARCH(s_i, *problem*, *path*)
    if $plan_i = failure$ **then return** *failure*
  return [**if** s_1 **then** $plan_1$ **else if** s_2 **then** $plan_2$ **else** ... **if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]

Note: nested if-then-else can be rewritten as case-switch

[Recall from Ch.4]: Cyclic Solution: Example

Example: Slippery Vacuum Cleaner

- Movement actions may fail: e.g., $\mathit{Results}(1, \mathit{Right}) = \{1, 2\}$
- A cyclic solution
- Use labels: [Suck, L1 : Right, if State = 5 then L1 else Suck]
- Use cycles: [Suck, While State = 5 do Right, Suck]



(© S. Russell & P. Norwig, AIMA)

Contingent Planning

- Contingent Planning: generation of plans with conditional branching based on percepts
  - appropriate for partial observability, non-determinism, or both
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
  - sets of belief states represented as disjunctions of logical formulas representing belief states
- When executing a contingent plan, the agent:
  - maintain its belief state as a logical formula
  - evaluate each branch condition:
    - if the belief state entails the condition formula, then proceed with the "then" branch
    - if the belief state entails the negation of the condition formula, then proceed with the "else" branch
- Note: The planning algorithm must guarantee that the agent never ends in a belief state where the condition's truth value is unknown

Contingent Planning

- Contingent Planning: generation of plans with conditional branching based on percepts
  - appropriate for partial observability, non-determinism, or both
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
  - sets of belief states represented as disjunctions of logical formulas representing belief states
- When executing a contingent plan, the agent:
  - maintain its belief state as a logical formula
  - evaluate each branch condition:
    - if the belief state entails the condition formula, then proceed with the “then” branch
    - if the belief state entails the negation of the condition formula, then proceed with the “else” branch
- Note: The planning algorithm must guarantee that the agent never ends in a belief state where the condition’s truth value is unknown

Contingent Planning

- Contingent Planning: generation of plans with conditional branching based on percepts
  - appropriate for partial observability, non-determinism, or both
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
  - sets of belief states represented as disjunctions of logical formulas representing belief states
- When executing a contingent plan, the agent:
  - maintain its belief state as a logical formula
  - evaluate each branch condition:
    - if the belief state entails the condition formula, then proceed with the “then” branch
    - if the belief state entails the negation of the condition formula, then proceed with the “else” branch
- Note: The planning algorithm must guarantee that the agent never ends in a belief state where the condition’s truth value is unknown

Contingent Planning

- Contingent Planning: generation of plans with conditional branching based on percepts
  - appropriate for partial observability, non-determinism, or both
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
  - sets of belief states represented as disjunctions of logical formulas representing belief states
- When executing a contingent plan, the agent:
  - maintain its belief state as a logical formula
  - evaluate each branch condition:
    - if the belief state entails the condition formula, then proceed with the “then” branch
    - if the belief state entails the negation of the condition formula, then proceed with the “else” branch
- Note: The planning algorithm must guarantee that the agent never ends in a belief state where the condition’s truth value is unknown

Contingent Planning

- Contingent Planning: generation of plans with conditional branching based on percepts
  - appropriate for partial observability, non-determinism, or both
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
  - sets of belief states represented as disjunctions of logical formulas representing belief states
- When executing a contingent plan, the agent:
  - maintain its belief state as a logical formula
  - evaluate each branch condition:
    - if the belief state entails the condition formula, then proceed with the "then" branch
    - if the belief state entails the negation of the condition formula, then proceed with the "else" branch
- Note: The planning algorithm must guarantee that the agent never ends in a belief state where the condition's truth value is unknown

Computing *Result*(a, b) with Conditional Steps

Three steps (aka prediction-observation-update)

1. **Prediction**: (same as for sensorless): $\hat{b} = b \setminus Del(a) \cup Add(a)$ // $\hat{b} = b \wedge Effects(a)$
2. **Observation prediction**: determines the set of percepts that could be observed in the predicted belief state $P \stackrel{\text{def}}{=} PossiblePercepts(\hat{b}) \stackrel{\text{def}}{=} \{p \mid \hat{b} \models Precond(p)\}$
3. **Update**: $Result(b, a) = \hat{b} \wedge \bigwedge_{p \in P} b_p$, s.t.:
   - if p has one percept schema, $Percept(p, Precond : c)$, s.t. $\hat{b} \models c$, then $b_p \stackrel{\text{def}}{=} p \wedge c$
   - if p has k percept schemata, $Percept(p, Precond : c_i)$, s.t. $\hat{b} \models c_i$, $i = 1..k$, then $b_p \stackrel{\text{def}}{=} \bigvee_{i=1}^{k}(p \wedge c_i)$

$\Longrightarrow$ $Result(b, a)$ CNF formula, not simply conjunction of literals (cubes)

$\Longrightarrow$ much harder to deal with

$\Longrightarrow$ often (over)approximations used to guantantee b_i cube

Computing *Result*(a, b) with Conditional Steps

Three steps (aka prediction-observation-update)

1. **Prediction**: (same as for sensorless): $\hat{b} = b \setminus Del(a) \cup Add(a)$ // $\hat{b} = b \wedge Effects(a)$
2. **Observation prediction**: determines the set of percepts that could be observed in the predicted belief state $P \stackrel{\text{def}}{=} PossiblePercepts(\hat{b}) \stackrel{\text{def}}{=} \{p \mid \hat{b} \models Precond(p)\}$
3. **Update**: $Result(b, a) = \hat{b} \wedge \bigwedge_{p \in P} b_p$, s.t.:
   - if p has one percept schema, $Percept(p, Precond : c)$, s.t. $\hat{b} \models c$, then $b_p \stackrel{\text{def}}{=} p \wedge c$
   - if p has k percept schemata, $Percept(p, Precond : c_i)$, s.t. $\hat{b} \models c_i$, $i = 1..k$, then $b_p \stackrel{\text{def}}{=} \bigvee_{i=1}^{k} (p \wedge c_i)$

$\Longrightarrow$ $Result(b, a)$ CNF formula, not simply conjunction of literals (cubes)

$\Longrightarrow$ much harder to deal with

$\Longrightarrow$ often (over)approximations used to guantantee b_i cube

Computing $Result(a, b)$ with Conditional Steps

Three steps (aka prediction-observation-update)

1. Prediction: (same as for sensorless): $\hat{b} = b \setminus Del(a) \cup Add(a)$ // $\hat{b} = b \wedge Effects(a)$
2. Observation prediction: determines the set of percepts that could be observed in the predicted belief state $P \stackrel{\text{def}}{=} PossiblePercepts(\hat{b}) \stackrel{\text{def}}{=} \{p \mid \hat{b} \models Precond(p)\}$
3. Update: $Result(b, a) = \hat{b} \wedge \bigwedge_{p \in P} b_p$, s.t.:
   - if p has one percept schema, $Percept(p, Precond : c)$, s.t. $\hat{b} \models c$, then $b_p \stackrel{\text{def}}{=} p \wedge c$
   - if p has k percept schemata, $Percept(p, Precond : c_i)$, s.t. $\hat{b} \models c_i$, $i = 1..k$, then $b_p \stackrel{\text{def}}{=} \bigvee_{i=1}^{k} (p \wedge c_i)$

$\Longrightarrow$ $Result(b, a)$ CNF formula, not simply conjunction of literals (cubes)

$\Longrightarrow$ much harder to deal with

$\Longrightarrow$ often (over)approximations used to guantantee b_i cube

Computing $Result(a, b)$ with Conditional Steps

Three steps (aka prediction-observation-update)

1. Prediction: (same as for sensorless): $\hat{b} = b \setminus Del(a) \cup Add(a)$ // $\hat{b} = b \wedge Effects(a)$
2. Observation prediction: determines the set of percepts that could be observed in the predicted belief state $P \stackrel{\text{def}}{=} PossiblePercepts(\hat{b}) \stackrel{\text{def}}{=} \{p \mid \hat{b} \models Precond(p)\}$
3. Update: $Result(b, a) = \hat{b} \wedge \bigwedge_{p \in P} b_p$, s.t.:
   - if p has one percept schema, $Percept(p, Precond : c)$, s.t. $\hat{b} \models c$, then $b_p \stackrel{\text{def}}{=} p \wedge c$
   - if p has k percept schemata, $Percept(p, Precond : c_i)$, s.t. $\hat{b} \models c_i$, $i = 1..k$, then $b_p \stackrel{\text{def}}{=} \bigvee_{i=1}^{k} (p \wedge c_i)$

$\Longrightarrow$ $Result(b, a)$ CNF formula, not simply conjunction of literals (cubes)
$\Longrightarrow$ much harder to deal with
$\Longrightarrow$ often (over)approximations used to guantantee b_i cube

Contingent Planning: Example

- Possible contingent plan for previous problem described below
  - variables in the plan to be considered existentially quantified
  - ex (2^{nd} row): "if there exists some color c that is the color of the table and the chair, then do nothing" (goal reached)
- "Color(Table,c)", "Color(Chair,c)' and "Color(Can,c)" percepts $\Longrightarrow$ must be matched against percept schemata

$$
\begin{array}{l}
[LookAt(Table), LookAt(Chair), \\
\quad \textbf{if } Color(Table, c) \land Color(Chair, c) \textbf{ then } NoOp \\
\qquad \textbf{else } [RemoveLid(Can_1), LookAt(Can_1), RemoveLid(Can_2), LookAt(Can_2), \\
\qquad\qquad \textbf{if } Color(Table, c) \land Color(can, c) \textbf{ then } Paint(Chair, can) \\
\qquad\qquad \textbf{else if } Color(Chair, c) \land Color(can, c) \textbf{ then } Paint(Table, can) \\
\qquad\qquad \textbf{else } [Paint(Chair, Can_1), Paint(Table, Can_1)]]]
\end{array}
$$

Contingent Planning: Example

- Possible contingent plan for previous problem described below
  - variables in the plan to be considered existentially quantified
  - ex (2^{nd} row): "if there exists some color c that is the color of the table and the chair, then do nothing" (goal reached)
- "Color(Table,c)", "Color(Chair,c)' and "Color(Can,c)" percepts
  $\Longrightarrow$ must be matched against percept schemata

$$
\begin{array}{l}
[LookAt(Table), LookAt(Chair), \\
\quad \textbf{if } Color(Table, c) \land Color(Chair, c) \textbf{ then } NoOp \\
\qquad \textbf{else } [RemoveLid(Can_1), LookAt(Can_1), RemoveLid(Can_2), LookAt(Can_2), \\
\qquad\qquad \textbf{if } Color(Table, c) \land Color(can, c) \textbf{ then } Paint(Chair, can) \\
\qquad\qquad \textbf{else if } Color(Chair, c) \land Color(can, c) \textbf{ then } Paint(Table, can) \\
\qquad\qquad \textbf{else } [Paint(Chair, Can_1), Paint(Table, Can_1)]]]
\end{array}
$$

Exercises

- Try to draw an execution of the conditional plan in previous slide against an imaginary physical state of the world of your choice
  - track step by step the belief states, the logical inferences, the actions performed