Planning and Acting

Chapter 12
Outline

diamond Planning with time and resource constraints
diamond Hierarchical planning
diamond The real world
diamond Conditional planning
diamond Monitoring and replanning
diamond Multiagent planning
Planning and Scheduling

Until now, we have considered what actions can do, but not how long they take or when they occur.

Scheduling
- Determine a schedule to complete a set of jobs
- Schedule should minimize total time to complete all jobs
- Schedule should respect resource constraints

Approach
1. Plan
2. Schedule
Example Scheduling Problem

\[
\text{Init(Chassis}(C_1) \land \text{Chassis}(C_2) \\
\land \text{Engine}(E_1,C_1,30) \land \text{Engine}(E_2,C_2,60) \\
\land \text{Wheels}(W_1,C_1,30) \land \text{Wheels}(W_2,C_2,15))
\]

\[
\text{Goal(Done}(C_1) \land \text{Done}(C_2))
\]

Action: AddEngine(e,c,m)
  \text{Precond: Engine}(e,c,d) \land \text{Chassis}(c) \land \text{neg EngineIn}(c)
  \text{Effect: EngineIn}(c) \land \text{Duration}(d)

Action: AddWheels(w,c)
  \text{Precond: Wheels}(w,c,d) \land \text{Chassis}(c)
  \text{Effect: WheelsOn}(c) \land \text{Duration}(d)

Action: Inspect(c)
  \text{Precond: EngineIn}(c) \land \text{WheelsOn}(c) \land \text{Chassis}(c)
  \text{Effect: Done}(c) \land \text{Duration}(10)
Example Scheduling Problem

Scheduling = critical path method (polynomial time)
Resource Constraints

\[
\text{Init}(\text{Chassis}(C_1) \land \text{Chassis}(C_2) \land \text{Engine}(E_1,C_1,30) \land \text{Engine}(E_2,C_2,60) \\
\land \text{Wheels}(W_1,C_1,30) \land \text{Wheels}(W_2,C_2,15) \\
\land \text{EngineHoists}(1) \land \text{WheelStations}(1) \land \text{Inspectors}(2))
\]

\[
\text{Goal}(\text{Done}(C_1) \land \text{Done}(C_2))
\]

\textbf{Action: AddEngine}(e,c,m)

\textbf{Precond:} \text{Engine}(e,c,d) \land \text{Chassis}(c) \land \neg \text{EngineIn}(c)

\textbf{Effect:} \text{EngineIn}(c) \land \text{Duration}(d)

\textbf{Resource:} \text{EngineHoists}(1)

\textbf{Action: AddWheels}(w,c)

\textbf{Precond:} \text{Wheels}(w,c,d) \land \text{Chassis}(c)

\textbf{Effect:} \text{WheelsOn}(c) \land \text{Duration}(d)

\textbf{Resource:} \text{WheelStations}(1)

\textbf{Action: Inspect}(c)

\textbf{Precond:} \text{EngineIn}(c) \land \text{WheelsOn}(c) \land \text{Chassis}(c)

\textbf{Effect:} \text{Done}(c) \land \text{Duration}(10)

\textbf{Resource:} \text{Inspectors}(1)
Scheduling with resource constraints is NP-Hard.
Hierarchical Planning

Hierarchical decomposition

– Each task at one level reduced to more primitive tasks at lower level
– Computational cost of solving simpler sub-problems is small

Hierarchical Task Networks (HTNs)

– Start with high-level plan
– Apply action decompositions
– Continue until only primitive actions in plan

We can modify POP to incorporate HTN planning

– POP successor function allows decomposition of current partial plan
Action Decomposition

Action(BuyLand, Precond: Money, Effect: Land \land \neg Money)
Action(GetLoan, Precond: GoodCredit, Effect: Money \land Mortgage)
Action(BuildHouse, Precond: Land, Effect: House)
Action(GetPermit, Precond: Land, Effect: Permit)
Action(HireBuilder, Effect: Contract)
Action(Construction, Precond: Permit \land Contract, 
    Effect: HouseBuilt \land \neg Permit)
Action(PayBuilder, Precond: Money \land HouseBuilt, 
    Effect: \neg Money \land House \land \neg Contract)
Decompose(BuildHouse, 
    Plan(Steps: \{S_1 : GetPermit, S_2 : HireBuilder, 
    S_3 : Construction, S_4 : PayBuilder\} 
    ORDERINGS: \{Start \prec S_1 \prec S_3 \prec S_4 \prec Finish, Start \prec S_2 \prec S_3\} 
    LINKS: \{ \begin{array}{ll} 
    Start \xrightarrow{\text{Land}} S_1, & Start \xrightarrow{\text{Money}} S_4, \\
    S_1 \xrightarrow{\text{Permit}} S_3, & S_2 \xrightarrow{\text{Contract}} S_3, \ S_3 \xrightarrow{\text{HouseBuilt}} S_4, \\
    S_4 \xrightarrow{\text{House}} Finish, & S_4 \xrightarrow{\neg \text{Money}} Finish \end{array} \})}
HTN Planning

Start → Buy Land → Build House → Finish

Start → Buy Land → Get Loan → GoodCredit
→ Get Permit → Construction → Pay Builder → House → Finish

Money → Land → House

Hire Builder → Money
The real world

START

~Flat(Spare) Intact(Spare) Off(Spare)
On(Tire1) Flat(Tire1)

ON(x) ~Flat(x)

FINISH

On(x)

Remove(x)

Off(x) ClearHub

Puton(x)

Intact(x) Flat(x)

Inflate(x)

Off(x) ClearHub

On(x) ~ClearHub

~Flat(x)
Things go wrong

Incomplete information

Unknown preconditions, e.g., $Intact(Spare)$?

Disjunctive effects, e.g., $Inflate(x)$ causes $Inflated(x) \lor SlowHiss(x) \lor Burst(x) \lor BrokenPump \lor \ldots$

Incorrect information

Current state incorrect, e.g., spare NOT intact

Missing/incorrect postconditions in operators

Qualification problem:

can never finish listing all the required preconditions and possible conditional outcomes of actions
Conformant or sensorless planning
   Devise a plan that works regardless of state or outcome

Such plans may not exist

Conditional planning
   Plan to obtain information (observation actions)
   Subplan for each contingency, e.g.,
   \[ \text{Check(Tire1), if Intact(Tire1) then Inflate(Tire1) else Call AAA} \]

Expensive because it plans for many unlikely cases

Monitoring/Replanning
   Assume normal states, outcomes
   Check progress during execution, replan if necessary

Unanticipated outcomes may lead to failure (e.g., no AAA card)

(Really need a combination; plan for likely/serious eventualities, deal with others when they arise, as they must eventually)
Conformant planning

Search in space of belief states (sets of possible actual states)
Conditional planning

If the world is nondeterministic or partially observable then percepts usually *provide information*, i.e., *split up* the belief state.
Conditional planning contd.

Conditional plans check (any consequence of KB +) percept

[\ldots \textbf{if} C \textbf{then} Plan_A \textbf{else} Plan_B, \ldots]

Execution: check $C$ against current KB, execute “then” or “else”

Need \textit{some} plan for \textit{every} possible percept

(Cf. game playing: \textit{some} response for \textit{every} opponent move)
(Cf. backward chaining: \textit{some} rule such that \textit{every} premise satisfied

AND–OR tree search (very similar to backward chaining algorithm)
Example

Double Murphy: sucking or arriving may dirty a clean square
Example

Triple Murphy: also sometimes stays put instead of moving

$[L_1 : \text{Left, if } AtR \text{ then } L_1 \text{ else } [\text{if CleanL then [] else Suck}]]$

or $[\text{while } AtR \text{ do } [\text{Left}, \text{if CleanL then [] else Suck}]]$

“Infinite loop” but will eventually work unless action always fails
“Failure” = preconditions of *remaining plan* not met

Preconditions of remaining plan

- all preconditions of remaining steps not achieved by remaining steps
- all causal links *crossing* current time point

On failure, resume POP to achieve open conditions from current state

IPEM (Integrated Planning, Execution, and Monitoring):  
keep updating *Start* to match current state
links from actions replaced by links from *Start* when done
Example
Example
Example

- At(Home)
- Go(HWS)
- Buy(Drill)
- At(HWS) Sells(HWS,Drill)
- Go(SM)
- Buy(Ban.)
- Buy(Milk)
- At(SM) Sells(SM,Milk)
- At(SM) Sells(SM,Ban.)
- At(HWS)
- Go(Home)
- Have(Milk)
- At(Home)
- Have(Ban.)
- Have(Drill)
- Finish

At(HWS)
Have(Drill)
Sells(SM,Ban.)
Sells(SM,Milk)
Example

Start

At(Home)
Go(HWS)

At(HWS) Sells(HWS,Drill)
Buy(Drill)

At(HWS)
Go(SM)

At(SM) Sells(SM,Milk) A((SM) Sells(SM,Ban.)
Buy(Milk) Buy(Ban.)

At(SM)
Go(Home)

Have(Milk) At(Home) Have(Ban.) Have(Drill)

Finish

Sells(SM,Ban.) Sells(SM,Milk)
Example

\begin{itemize}
  \item Start
  \item At(Home)
  \item Go(HWS)
  \item At(HWS)
  \item Sells(HWS, Drill)
  \item Buy(Drill)
  \item At(HWS)
  \item Go(SM)
  \item At(SM)
  \item Sells(SM, Milk)
  \item Buy(Milk)
  \item At(SM)
  \item Sells(SM, Ban.)
  \item Buy(Ban.)
  \item At(SM)
  \item Go(Home)
  \item At(Home)
  \item Have(Milk)
  \item At(SM)
  \item Have(Drill)
  \item Have(Ban.)
  \item Have(Milk)
  \item Finish
\end{itemize}
Example

At(Home) Go(HWS) Buy(Drill) Go(SM) Buy(Milk) Have(Ban.) Have(Drill) Have(Milk) Finish

At(Home) Go(HWS) Sells(HWS, Drill) Sells(SM, Milk) Sells(SM, Ban.)
Emergent behavior

PRECONDITIONS

START

Color(Chair,Blue) ~Have(Red)

Get(Red)

Have(Red)

Paint(Red)

Color(Chair,Red)

FINISH

FAILURE RESPONSE

Fetch more red
Emergent behavior

PRECONDITIONS

START

Color(Chair,Blue) ~Have(Red)

Get(Red)

Have(Red)

Paint(Red)

Color(Chair,Red)

FINISH

FAILURE RESPONSE

Extra coat of paint
Emergent behavior

PRECONDITIONS

START

\(\text{Color(Chair,Blue)} \downarrow \sim \text{Have(Red)}\)

Get(Red)

Have(Red)

Paint(Red)

Color(Chair,Red)

FINISH

FAILURE RESPONSE

Extra coat of paint

“Loop until success” behavior emerges from interaction between monitor/replan agent design and uncooperative environment
Multiagent Planning

Multiagent environments can be *cooperative* or *competitive*.

Cooperative multiagent planning
- Joint goals
- Joint plan (may be many, ensure all agents use same plan)
- Synchronization
- Coordination mechanisms

Competitive multiagent planning
- Represent other agents
- Compute other agents’ possible plans
- Determine interaction of other plans on own plan
- Decide on best action
- Approach: Combine minimax with POP
Summary

Planning plus scheduling can address time and resource constraints.

Hierarchical planning can reduce runtime, especially if sub-plans are independent.

Realistic domains
- Incomplete or incorrect information
- Non-deterministic, partially observable environments
- Approach: Execution monitoring with replanning

Multiagent planning