11. Planning in real world

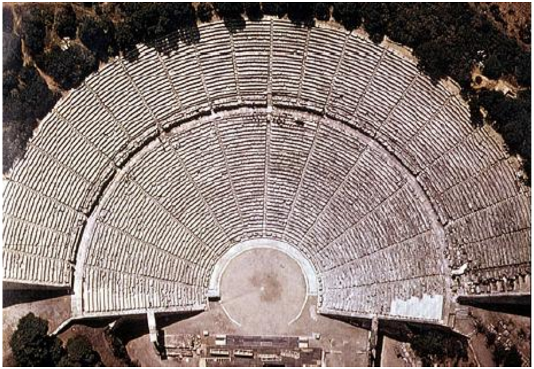
Adrian Groza

In which we see how more expressive representations and more interactive agent architectures lead to planners that are useful in the real world

An experimental lecture - Act XI

Students in the play (tiny roles):

- a puzzle teller
- a painter

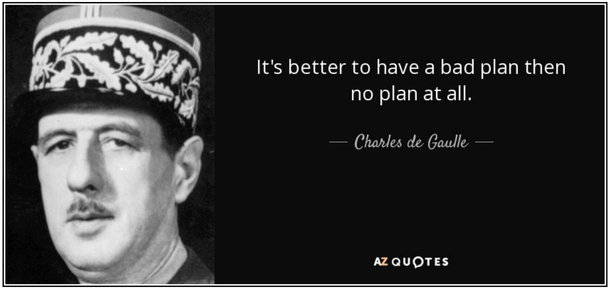


Common Experience: The theatre of Epidaurus (online version)

- Time: Fall, middle point before graduation
- Scene: eager and (still) mysterious students, (un)predictable Teams

Lost in translation

$\forall s_1\ \forall s_2\ Situation(s_1) \land Situation(s_2) \land \exists p_1\ hasPlan(s_1, p_1) \land Plan(p_1) \land Bad(p_1) \land \neg\exists p_2\ Plan(p_2) \land hasPlan(s_2, p_2) \rightarrow better(s_1, s_2)$


It's better to have a bad plan then no plan at all.

— Charles de Gaulle —

AZ QUOTES

In the previous episode

- Planning Domain Definition Language
- Partial Order Planning (open conditions, clobber)
- Planning is non-trivial (relevant actions, Sussman anomaly)
- Heuristic for planning - by relaxing the problem (e.g. igonoring preconditions, ignoring negative effects)

Spoiler alert: Planning/acting in Real world

- So far only looked at classical planning, i.e. environments are fully observable, static, determinist. Also assumed that action descriptions are correct and complete.
- Unrealistic in many real-world applications: Don't know everything - may even hold incorrect information. Actions can go wrong.
- Distinction: bounded vs. unbounded indeterminacy: can possible preconditions and effects be listed at all? Unbounded indeterminacy related to qualification problem
- Real world include temporal and resource constraints:
  - classical planning talks about what to do, and in what order, but cannot talk about time: how long an action takes
  - an airline has a limited number of staff - and staff who are on one flight cannot be on another at the same time.

Outline

Solving scheduling problems

Example (Assembling two cars with resource constraints)

Jobs	$(AddEngine1 \prec AddWheels1 \prec Inspect1,$ $AddEngine2 \prec AddWheels2 \prec Inspect2)$
Resources	$(EngineHoists(1), WheelStations(1), Inspectors(2), LugNuts(500))$
Action	$(AddEngine_1, DURATION : 30, USE : EngineHoists(1))$
Action	$(AddEngine_2, DURATION : 60, USE : EngineHoists(1))$
Action	$(AddWheels_1, DURATION : 30, CONSUME : LugNuts(20), USE : WheelStations(1))$
Action	$(AddWheels_2, DURATION : 15, CONSUME : LugNuts(20), USE : WheelStations(1))$
Action	$(Inspecti, DURATION : 10, USE : Inspectors(1))$

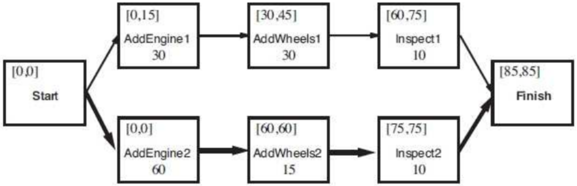
- Which is the optimal solution? (115 minutes)
- Begin by considering just the temporal constraints, ignoring resources

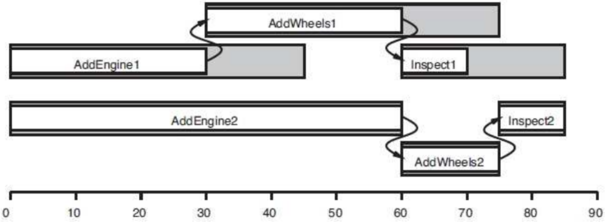
Critical path method

- To determine the possible start and end times of each action
- Critical path is that path whose total duration is longest
- Actions have a window of time [ES,LS]
  (ES - earliest possible start time, LS - latest possible start time)
- Given A, B actions and $A \prec B$:

$$ES(Start) = 0 \qquad ES(B) = max_{A \prec B} ES(A) + Duration(A)$$
$$LS(Finish) = ES(Finish) \qquad LS(A) = min_{B \succ A} LS(B) - Duration(A)$$

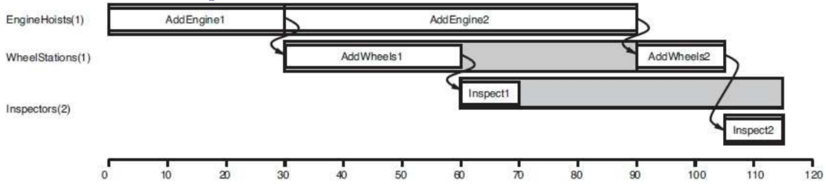
Critical path algorithm





- Actions with zero slack are on the critical path
- Complexity: $O(Nb)$, where N is the number of actions and b is the maximum branching factor into or out of an action (LS and ES computations are done once for each action, and each computation iterates over at most b other actions)

Considering resource constraints



- Complexity introduced by the cannot overlap constraint: disjunction of two linear inequalities, one for each possible ordering: $(AddEngine_1 \prec AddEngine_2) \vee AddEngine_2 \prec AddEngine_1)$

Minimum slack heuristic

1. on each iteration, schedule for the earliest possible start whichever unscheduled action has all its predecessors scheduled and has the least slack;
2. then update the ES and LS times for each affected action and repeat.

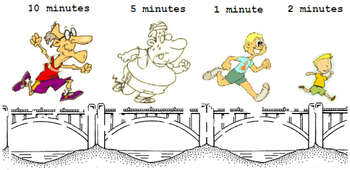
This greedy heuristic resembles the minimum-remaining-values (MRV) heuristic in constraint satisfaction.
How long is the solution with the minimum slack algorithm for the assembly problem?
130 minutes

A puzzle with time and resource constraints

Crossing the bridge in minimum time with one lamp

Four travelers (A, B, C, and D) have to cross a bridge over a deep ravine. It is a very dark night and the travelers only have one oil lamp. The lamp is essential for successfully crossing the ravine because the bridge is very old and has plenty of holes and loose boards. What is worse, its construction is quite weak and it can only support two men at any time. It turns out that each traveler needs a different amount of time to cross the bridge: A:1, B:2, C:5, D:10 minutes. And since each traveler needs the lamp to cross, it is the slower man in a pair who determines the total time required to cross.

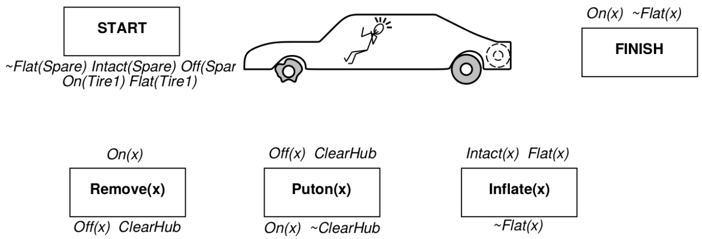
- Which heuristic: exploit the quickest traveler?
- Which heuristic: minimum difference between two travelers?



Step	Persons	Direction	Elapsed Time
1.	A,B	$\rightarrow$	$max(A, B) = 2$
2.	A	$\leftarrow$	$2 + 1 = 3$
3.	C,D	$\rightarrow$	$3 + max(C, D) = 13$
4.	B	$\leftarrow$	$13 + 2 = 15$
5.	A,B	$\rightarrow$	$13 + max(A, B) = 17$

Outline

The real world - things can go wrong



Incomplete information

- Unknown preconditions, e.g. $Intact(Spare)$?
- Disjunctive effects, e.g. $Inflate(x)$ causes $Inflated(x) \lor SlowHiss(x) \lor Burst(x) \lor BrokenPump \lor \dots$

Incorrect information

- Current state incorrect, e.g., spare NOT intact
- Missing/incorrect postconditions in operators
- Qualification problem: can never finish listing all preconditions and outcomes

Solutions

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.,
$[Check(Tire_1), \textit{if } Intact(Tire_1) \textit{ then } Inflate(Tire_1) \textit{ else } CallAAA]$
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)

Running scenario: Painting chairs and tables

Given a chair and a table, the goal is to have them match. In the initial state we have two cans of paint, but the colors of the paint and the furniture are unknown. Only the table is initially in the agents field of view There are two actions: removing the lid from a paint can and painting an object using the paint from an open can. How would each of the following handle this problem?

- Classical planning?
- Conformant (sensorless) plan?
- Contingent (conditional) plan?
- Online (replanning)?

Outline

1. Init: $Object(Table) \land Object(Chair) \land Can(C_1) \land Can(C_2) \land InView(Table)$
2. Goal: $Color(Chair, c) \land Color(Table, c)$
3. Actions:
   Action(RemoveLid(can),
   PRECOND: $Can(can)$
   EFFECT: $Open(can)$)

   Now, we allow preconditions and effects to contain variables that are not part of the action's variable (e.g. Paint(x, can)) - to handle partial observability
   Action(Paint(x, can),
   PRECOND: $Object(x) \land Can(can) \land Color(can, c) \land Open(can)$
   EFFECT: $Color(x, c)$)

   New in town: observation action
   Action(LookAt (x),
   PRECOND: $InView(y) \land (x \neq y)$
   EFFECT: $InView(x) \land \neg InView(y)$)

Model of sensors

To solve a partially observable problem, the agent will have to reason about the percepts it will obtain when it is executing the plan.

Percept schema

models the agents sensors. It tells the agent what it knows, given certain conditions about the state it's in

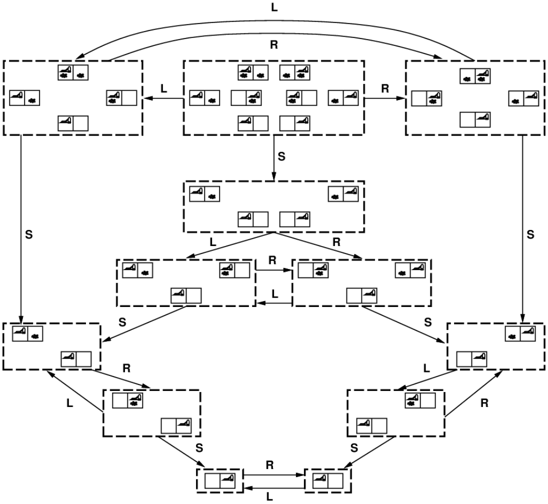
Percept (Color(x, c),
 PRECOND: $Object(x) \wedge InView(x)$)

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

- A fully observable environment has a percept axiom for each fluent with no preconditions!
- A sensorless planner has no percept schemata at all!
- A sensorless planner is a persona non-grata in the IoT world (governed by the Lord of the Things), but still successful due to its robustness (nothing to break).

Conformant planning

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



The update of a belief state b given an action a is the set of all states that result from doing a in each possible state s that satisfies belief state b.

$$b' = Result(b, a) = s' : s' = Result_P(s, a) \land s \in b$$

Conformant planning for the painting scenario

- Initial belief state: $b_0 = Color(x, C(x))$ from $\forall x\ \exists c\ Color(x, c)$
- Open-world assumption: in which states contain both positive and negative fluents, and if a fluent does not appear, its value is unknown

$$RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)$$

1. If action adds l, then l is true in b' regardless of its initial value
2. If action deletes l, then l is false in b' regardless initial value
3. If action says nothing about l, then l will retain its b-value

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

$$b_1 = RESULT(b_0, RemoveLid(C_1)) = Color(x, C(x)) \land Open(Can_1)$$
$$b_2 = Color(x, C(x)) \land Open(Can_1) \land Color(Chair, C(Can_1))$$
$$b_3 = Color(x, C(x)) \land Open(Can_1) \land Color(Chair, C(Can_1)) \land Color(Table, C(Can_1))$$

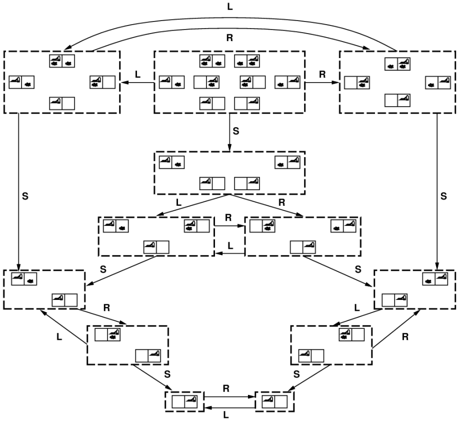
Conditional effects

So far, we have only considered actions that have the same effects on all states where the preconditions are satisfied

$Action(Suck,$
$\quad EFFECT : whenAtL :$
$CleanL \wedge whenAtR : CleanR)$
$b_0 = true$
$b_1 = (AtL \wedge CleanL) \vee (AtR \wedge CleanR)$

$Action(SuckL,$
$\quad PRECOND : AtL; EFFECT : CleanL)$
$Action(SuckR,$
$\quad PRECOND : AtR; EFFECT : CleanR)$

We cannot determine the applicability of *SuckL* and *SuckR* in b_0



For sensorless planning, it is better to have conditional effects than an inapplicable action

$[Right, Suck, Left, Suck] : b_0, b_1, b_2, b_3, b_4$

Augment STRIPS to allow for nondeterminism

- Add Conditional effects (i.e., depends on state in which its executed):
  Form: `when<condition>:<effect>`
  Action(Suck,
  Precond:,
  Effect:(when $AtL : CleanL$) $\wedge$ (when $AtR : CleanR$)
- Add Disjunctive effects (e.g., to model when action sometimes fails):
  Action(Left, Precond: AtR, Effect: $AtL \vee AtR$)

Outline

A contingent planner can do better than a sensorless planning:

1. Look at the table and chair to sense their colours.
2. If they are the same colour, you are done.
3. If not, look at the paint cans.
4. If one of the cans is the same colour as one of the pieces of furniture, then apply that paint to the other piece of furniture.
5. Otherwise, paint both pieces with one of the cans.

Example (Conditional/contingent planning)

```
[LookAt(Table), LookAt(Chair),
    if $Color(Table, c) \wedge Color(Chair, c)$ then $NoOp$
        else [$RemoveLid(Can1), LookAt(Can_1), RemoveLid(Can_2), LookAt(Can_2)$,
            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 [$Paint(Chair, Can_1), Paint(Table, Can_1)$]]]
```

Percept (Color(x,c), PRECOND: $Object(x) \wedge InView(x)$)
Percept (Color(can,c), PRECOND: $Can(can) \wedge InView(can) \wedge Open(can)$)
Action(LookAt(x),
PRECOND: $InView(y) \wedge (x \neq y)$
EFFECT:$InView(x) \wedge \neq InView(y)$)

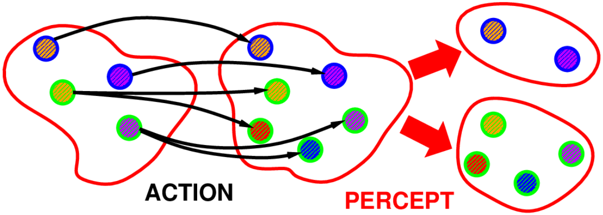
Computing b' an action and subsequent percept is done in two stages:

1. Calculate the belief state after the action, just as for the sensorless agent:
$$b' = (b - DEL(a)) \cup ADD(a)$$
2. Add information for the percept axioms with the conditions satisfied

Outline

Games against nature

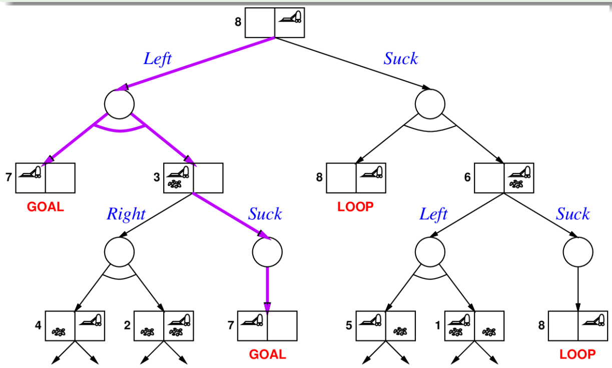
If the world is nondeterministic or partially observable then percepts usually *provide information*, i.e., *split up* the belief state



- Conditional plans check (any consequence of KB +) percept
  $[\ldots, \mathit{if}\ C\ \mathit{then}\ Plan_A\ \mathit{else}\ Plan_B, \ldots]$
- Execution: check C against current KB, execute "then" or "else". Need *some* plan for *every* possible percept
  (Cf. game playing: *some* response for *every* opponent move)
  (Cf. backward chaining: *some* rule such that *every* premise satisfied
- AND–OR tree search (very similar to backward chaining algorithm)

Example (Double Murphy)

sucking or arriving may dirty a clean square

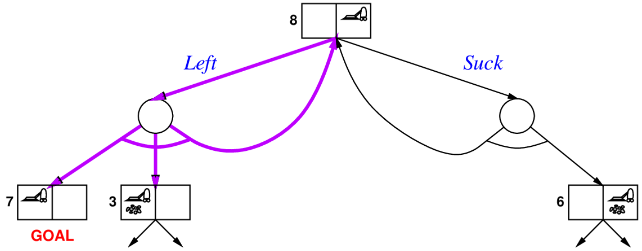


- Robot takes action in state nodes.
- Nature decides outcome at chance nodes.
- Plan needs to take some action at every state it reaches (i.e., Or nodes)
- Plan must handle every outcome for the action it takes (i.e., And nodes)
- Solution is a subtree with (1) goal node at every leaf, (2) one action specified at each state node, and (3) includes every outcome at chance nodes

Plan: [Left, if CleanL then [] else Suck]

Example (Triple Murphy)

also sometimes stays instead of moving



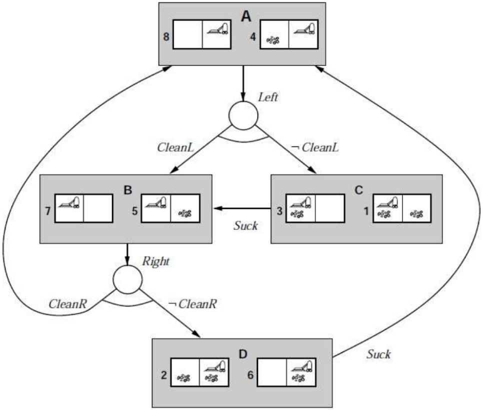
$[L_1:\ Left, if\ AtR\ then\ L_1\ else\ [if\ CleanL\ then\ noplan\ else\ Suck]]$
or $[whileAtR[Left],\ if\ CleanL\ then\ noplan\ else\ Suck]$
“Infinite loop” but will eventually work unless action always fails

Nondeterminism and partially observable env.

Example (Alternate double Murphy)

Vacuum cleaner can sense cleanliness of square its in, but not the other square, and dirt can sometimes be left behind when leaving a clean square.

Plan in fully observable world: Keep moving left and right, sucking up dirt whenever it appears, until both squares are clean and in the left square. But now goal test cannot be performed!



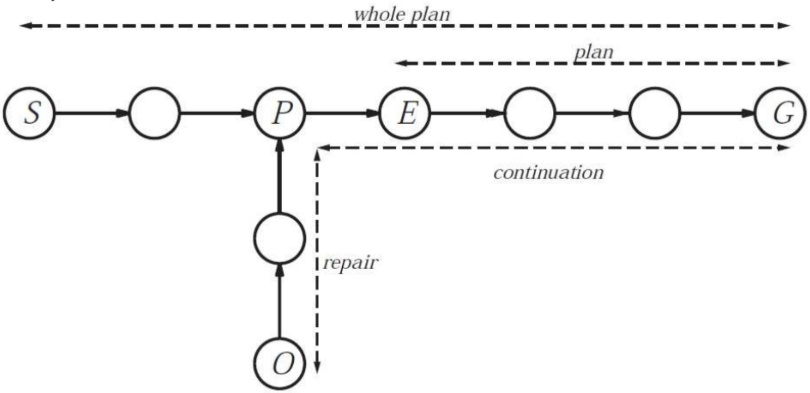
Outline

Execution monitoring and replanning

- Execution monitoring: checking whether things are going according to plan (necessitated by unbounded indeterminacy in realistic environments or agent gets tired of planning for every little contingency) Some branches of a partially constructed contingent plan can simply say Replan
- Action monitoring: checking whether next action is feasible
- Plan monitoring: checking whether remainder of plan is feasible
- Replanning: ability to find new plan when things go wrong (usually repairing the old plan). Replanning is needed if the agents model of the world is incorrect:
  - missing precondition: removing the lid requires a screwdriver
  - missing efect: painting an object may get paint on the floor
  - missing state variable: no notion of the amount of paint in a can
  - exogenous events: someone knocking over the paint can

Repair or replan?

While attempting to get from S to G, a problem is encountered in E, agent discovers actual state is O and plans to get to P and execute the rest of the original plan

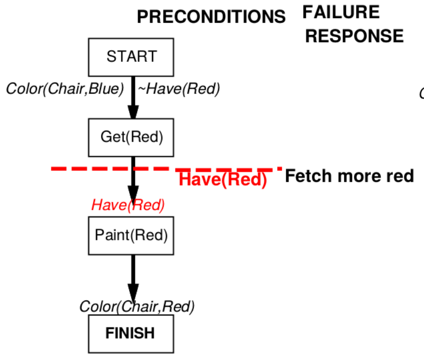


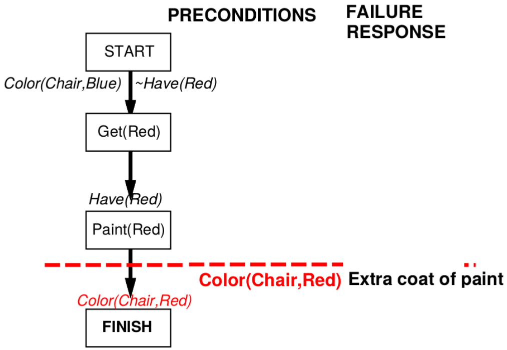
Example (Replanning)

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

- Suppose the agent observes that the table and can of paint are white and the chair is black.
- The current state is identical to the precondition before the $Paint(Chair, Can_1)$ Repair: []; Plan: [Paint]
- Loop created by a process of plan-execute-replan, not by an explicit loop in a plan
- Action monitoring leads to less intelligent behavior than plan monitoring: - red paint enough only for the chair

Emergent behaviour





"Loop until success" behavior *emerges* from interaction between monitor/replan agent design and uncooperative environment

Outline

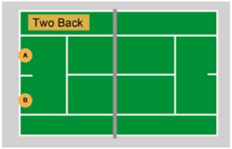
Multi-agent planning

Between the purely single-agent and truly multiagent cases is a wide spectrum of problems that exhibit various degrees of decomposition of the monolithic agent

- Multieffector planning: to manage each effector while handling positive and negative interactions among the effectors
- Multibody planning: effectors are physically decoupled into detached units - as in a fleet of delivery robots in a factory
- Decentralized planning: multiple reconnaissance robots covering a wide area may often be out of radio contact with each other and should share their findings during times when communication is feasible.
- Coordination: multiagents with the same goal (e.g. tenis)

Planning with multiple simultaneous actions

The double tennis problem



Actors(A,B)
Init (At(A,LeftBaseline) $\wedge$ At(B,RightNet) $\wedge$
    Approaching(Ball, RightBaseline)) $\wedge$ Partner (A,B) $\wedge$ Partner (B,A)
Goal (Returned(Ball) $\wedge$ (At(a, RightNet) $\vee$ At(a, LeftNet))
Action(Hit (actor, Ball),
    PRECOND: Approaching(Ball, loc) $\wedge$ At(actor, loc)
    EFFECT: Returned(Ball))
Action(Go(actor, to),
    PRECOND: At(actor, loc) $\wedge$ to $\neq$ loc,
    EFFECT: At(actor, to) $\wedge$ $\neg$ At(actor, loc))

- Assume perfect synchronization: each action takes the same amount of time and actions at each point in the joint plan are simultaneous
- Joint action: $\langle a_1, ..., a_n \rangle$, where a_i is the action taken by the ith actor.
- Loosely coupled: focus on decoupling the actors to the extent possible, so that the complexity of the problem grows linearly with n rather than exponentially.
- Joint plan:
  A: [Go(A, RightBaseline), Hit(A, Ball)]
  B: [NoOp(B), NoOp(B)]

Concurent action list

- Problems arise, however, when a plan has both agents hitting the ball at the same time.
- The difficulty is that preconditions constrain the state in which an action can be executed successfully, but do not constrain other actions that might mess it up.

Action(Hit(a, Ball),
    CONCURRENT: $b \neq a \Rightarrow \neg Hit(b, Ball)$
    PRECOND: $Approaching(Ball, loc) \land At(a, loc)$
    EFFECT: $Returned(Ball)$)

For some actions, the desired effect is achieved only when another action occurs concurrently:

Action(Carry(a, cooler, here, there),
    CONCURRENT: $b \neq a \land Carry(b, cooler, here, there)$
    PRECOND: $At(a, here) \land At(cooler, here) \land Cooler(cooler)$
    EFFECT: $At(a, there) \land At(cooler, there) \land \neg At(a, here) \land \neg At(cooler, here)$)

Cooperation and coordination

PLAN 1: A : [Go(A, RightBaseline), Hit(A, Ball)]
B : [NoOp(B), N oOp(B)]
PLAN 2: A : [Go(A, LeftNet), N oOp(A)]
B : [Go(B, RightBaseline), Hit(B, Ball)]

- Convention: any constraint on the selection of joint plans (e.g., "stick to your side of the court").
- Social laws: widespread conventions
- Communication: "Mine!"
- Plan recognition: If agent A heads for the net, then agent B is obliged to go back to the baseline to hit the ball, because PLAN 2 is the only joint plan that begins with As heading for the net

Outline

Analysis of planning approaches

- Planning combines the two major areas of AI we have covered so far: search and logic.
- Planning is foremost an exercise in controlling combinatorial explosion (e.g. If there are n propositions in a domain, then there are n^2 states); real-world industrial applications with millions of states and thousands of actions
- New techniques will emerge, perhaps providing a synthesis of highly expressive first-order and hierarchical representations with the highly efficient factored and propositional representations that dominate today.
- Portfolio planning system: where a collection of algorithms are available to apply to any given problem. This can be done selectively (the system classifies each new problem to choose the best algorithm for it), or in parallel (all the algorithms run concurrently, each on a different CPU), by interleaving the algorithms according to a schedule.

Summary

- Many actions consume resources. It is convenient to treat these resources as numeric measures in a pool rather than try to reason about each individual resource it is usually cheap and effective to check partial plans for satisfaction of resource constraints before attempting further refinements
- Time is one of the most important resources. It can be handled by specialized scheduling algorithms
- An online planning agent uses execution monitoring and splices in repairs as needed to recover from unexpected situations, which can be due to nondeterministic actions, exogenous events, or incorrect models of the environment.
- Incomplete info: use conditional plans, conformant planning (can use belief states)
- Incorrect info: use execution monitoring and replanning