Lecture #11 - Planning in the Real World

Outline
- Time, schedules and resources
- Hierarchical task network planning
- Non-deterministic domains
  - Conditional planning
  - Execution monitoring and replanning
  - Continuous planning
- Multi-agent planning
- Ch. 11

Time, Schedules and Resources
- Classical planning: simplified problems
- Real-world:
  - Actions occur at specific moments in time
  - Actions have a beginning and an end
  - Actions take a certain amount of time
- Job-shop scheduling:
  - Complete a set of jobs, each of which consists of a sequence of actions,
  - Where each action has a given duration and might require resources.
  - Determine a schedule that minimizes the total time required to complete all jobs (respecting resource constraints).

Car Construction Example

Init(Chassis(C1) ∧ Chassis(C2) ∧ Engine(E1,C1,30) ∧ Engine(E1,C2,60) ∧ Wheels(W1,C1,30) ∧ Wheels(W2,C2,15))

Goal(Done(C1) ∧ Done(C2))

Action(AddEngine(e,c,m)
  PRECOND: Engine(e,c,d) ∧ Chassis(c) ∧ ¬EngineIn(c)
  EFFECT: EngineIn(c) ∧ Duration(d))

Action(AddWheels(w,c)
  PRECOND: Wheels(w,c,d) ∧ Chassis(c)
  EFFECT: WheelsOn(c) ∧ Duration(d))

Action(Inspect(c)
  PRECOND: EngineIn(c) ∧ WheelsOn(c) ∧ Chassis(c)
  EFFECT: Done(c) ∧ Duration(10))

Temporal Constraints for Problem

Critical path method is used to determine start and end times:
- Path = linear sequence from start to end
- Critical path = path with longest total duration
- Determines the duration of the entire plan
- Critical path should be executed without delay

Planning vs. Scheduling
- How does the problem differ from a standard planning problem?
- When does an action start and when does it end?
  - So next to order (planning), duration is also considered
  - Duration(d)
- Critical path method is used to determine start and end times:
  - Path = linear sequence from start to end
  - Critical path = path with longest total duration
  - Determines the duration of the entire plan
  - Critical path should be executed without delay
Artificial Intelligence

ES and LS
- Earliest possible (ES) start time
- Latest possible (LS) start time
- \( \text{LS} - \text{ES} = \text{slack of an action} \)
- For all actions slack determines the schedule for the entire problem.
  - \( \text{ES}(\text{Start}) = 0 \)
  - \( \text{ES}(A) = \max A < B \text{ ES}(A) + \text{Duration}(A) \)
  - \( \text{LS}(\text{Finish}) = \text{ES}(\text{Finish}) \)
  - \( \text{LS}(A) = \min A < B \text{ LS}(B) - \text{Duration}(A) \)
- Complexity is \( O(Nb) \) (given a PO)

Temporal Constraints for Problem

Car Example with Resources

Resource Constraints
- Resource constraints = required material or objects to perform task
- Reusable resources
  - A resource that is occupied during an action but becomes available when the action is finished.
  - Require extension of action syntax:
  - Resource: \( R(k) \)
    - \( k \) units of resource are required by the action.
    - Is a pre-requisite before the action can be performed.
    - Resource can not be used for \( k \) time units by other.

Specific Task Times

Car Example with Resource Constraints
Aggregation

- **Aggregation** = group individual objects into quantities when the objects are undistinguishable with respect to their purpose.
  - Reduces complexity
- Resource constraints make scheduling problems more complicated.
  - Additional interactions among actions
- **Heuristic: minimum slack algorithm**
  - Select an action with all predecessors scheduled and with the least slack for the earliest possible start.

Hierarchical Task Network Planning

- Reduce complexity ⇒ hierarchical decomposition
  - At each level of the hierarchy a computational task is reduced to a small number of activities at the next lower level.
  - The computational cost of arranging these activities is low.
- Hierarchical task network (HTN) planning uses a refinement of actions through decomposition.
  - e.g. building a house = getting a permit + hiring a contractor + doing the construction + paying the contractor.
  - Refined until only primitive actions remain.
- Pure and hybrid HTN planning.

Representation Decomposition

- General descriptions are stored in plan library.
  - Each method = Decompos(a,d); a= action and d= PO plan.
- See buildhouse example
- Start action supplies all preconditions of actions not supplied by other actions.
  - =external preconditions
- Finish action has all effects of actions not present in other actions
  - =external effects
    - Primary effects (used to achieve goal) vs. secondary effects

Buildhouse example

Buildhouse Example Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>PRECOND</th>
<th>EFFECT</th>
<th>Precond of Action</th>
<th>Effect of Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuyLand</td>
<td>Money</td>
<td>Land</td>
<td>¬BuyLand</td>
<td>BuyLand</td>
</tr>
<tr>
<td>GetLoan</td>
<td>GoodCredit</td>
<td>Money</td>
<td>¬GetLoan</td>
<td>GetLoan</td>
</tr>
<tr>
<td>BuildHouse</td>
<td>Land</td>
<td>House</td>
<td>¬BuildHouse</td>
<td>BuildHouse</td>
</tr>
<tr>
<td>GetPermit</td>
<td>Land</td>
<td>Permit</td>
<td>¬GetPermit</td>
<td>GetPermit</td>
</tr>
<tr>
<td>HireBuilder</td>
<td>Permit</td>
<td>House</td>
<td>¬HireBuilder</td>
<td>HireBuilder</td>
</tr>
<tr>
<td>Construction</td>
<td>Permit, Contract</td>
<td>HouseBuilt</td>
<td>¬Construction</td>
<td>Construction</td>
</tr>
<tr>
<td>PayBuilder</td>
<td>Money, HouseBuilt</td>
<td>¬Money</td>
<td>PayBuilder</td>
<td></td>
</tr>
</tbody>
</table>

Decompose(BuildHouse, Plan was:\ STEPS/3: GetPermit, S2: HireBuilder, S3: Construction, S4: PayBuilder)

ORDERINGS: (Start < S1 < S3 < S4 < Finish, Start=S2=S3)

LINKS

Properties of Decomposition

- Should be correct implementation of action a
  - Correct if plan d is complete and consistent PO plan for the problem of achieving the effects of a given the preconditions of a.
- A decomposition is not necessarily unique.
- Performs information hiding:
  - STRIPS action description of higher-level action hides some preconditions and effects
  - Ignore all internal effects of decomposition
  - Does not specify the intervals inside the activity during which preconditions and effects must hold.
- Information hiding is essential to HTN planning.
Recapitulation of POP (1)
- Assume propositional planning problems:
  - The initial plan contains Start and Finish, the ordering constraint Start < Finish, no causal links, all the preconditions in Finish are open.
  - Successor function:
    - picks one open precondition p on an action B and
    - generates a successor plan for every possible consistent way of choosing action A that achieves p.
  - Test goal

Recapitulation of POP (2)
- When generating successor plan:
  - The causal link A--p->B and the ordering constraining A < B is added to the plan.
    - If A is new also add start < A and A < B to the plan
  - Resolve conflicts between new causal link and all existing actions
  - Resolve conflicts between action A (if new) and all existing causal links.

Adapting POP to HTN planning
- Remember POP?
  - Modify the successor function:
    apply decomposition to current plan
- NEW Successor function:
  - Select non-primitive action a' in P
  - For any Decompose(a',d') method in library where a and a' unify with substitution θ
    - Replace a' with d' = subst(θ)

POP+HTN example
- How to Hook up d in a’?
  - Remove action a’ from P and replace with dθ
    - For each step s in d’ select an action that will play the role of s (either new s or existing s’ from P)
    - Possibility of subtask sharing
  - Connect ordering steps for a’ to the steps in d’
    - Put all constraints so that constraints of the form B < a’ are maintained.
    - Watch out for too strict orderings!
  - Connect the causal links
    - If B -p-> a’ is a causal link in P, replace it by a set of causal links from B to all steps in d’ with preconditions p that were supplied by the start step
    - Do for a’ -p-> C
What about HTN?
- Additional modification to POP are necessary
- BAD news: pure HTN planning is undecidable due to recursive decomposition actions.
  - Walk=make one step and walk
- Resolve problems by
  - Rule out recursion.
  - Bound the length of relevant solutions,
  - Hybridize HTN with POP
- Yet HTN can be efficient (see motivations in book)

Non-deterministic domains
- So far: fully observable, static and deterministic domains.
  - Agent can plan first and then execute plan with eyes closed
- Uncertain environments:
  - Incomplete (partially observable and/or nondeterministic) and
  - Incorrect (differences between world and model) information
  - Use percepts
  - Adapt plan when necessary
- Degree of uncertainty defined by indeterminacy
  - Bounded: actions can have unpredictable effects, yet can be listed in action description axioms.
  - Unbounded: preconditions and effects unknown or too large to enumerate.

Handling Indeterminacy
- Sensorless planning (conformant planning)
  - Find plan that achieves goal in all possible circumstances (regardless of initial state and action effects).
- Conditional planning (Contingency planning)
  - Construct conditional plan with different branches for possible contingencies.
- Execution monitoring and replanning
  - While constructing plan, judge whether plan requires revision.
- Continuous planning
  - Planning active for a life time: adapt to changes in circumstances and reformulate goals if necessary.

Abstract Example - Painting
- Initial state = <chair, table, cans of paint, unknown colors>, goal state = <color(table) = color(chair)>
- Sensorless planning (conformant planning)
  - Open any can of paint and apply it to both chair and table.
- Conditional planning (Contingency planning)
  - Sense color of table and chair, if they are the same then finish else sense labels paint if color(label) = color(Furniture) then apply color to other piece else apply color to both
- Execution monitoring and replanning
  - Same as conditional and can fix errors (missed spots)
- Continuous planning
  - Can revise goal when we want to first eat before painting the table and the chair.
**Conditional Planning**

- Deal with uncertainty by checking the environment to see what is really happening.
- Used in fully observable and nondeterministic environments:
  - The outcome of an action is unknown.
- Conditional steps will check the state of the environment.
- How to construct a conditional plan?

**Example, the vacuum-world**

- Actions: left, right, suck
- Propositions to define states: AtL, AtR, CleanL, CleanR
- How to include nondeterminism?
  - Actions can have more than one effect
  - E.g. moving left sometimes fails
    - Action(Left, PRECOND: AtR, EFFECT: AtL)
    - Becomes: Action(Left, PRECOND: AtR, EFFECT: AtL ∨ AtR)
  - Actions can have conditional effects
    - Action(Left, PRECOND: AtR, EFFECT: AtL ∨ (AtL ∧ when cleanL: ¬cleanL))
    - Both disjunctive and conditional

**Conditional Planning**

- Conditional plans require conditional steps:
  - If <test> then plan_A else plan_B
  - If AtL ∧ CleanL then Right else Suck
- Plans become trees
- Games against nature:
  - Find conditional plans that work regardless of which action outcomes actually occur.
  - Assume vacuum-world
  - Initial state = AtR ∧ CleanL ∧ CleanR
  - Double murphy: possibility of desposit dirt when moving to other square and possibility of despositing dirt when action is Suck.

**State (Game)Tree**

- Solution of Games against “Nature”
  - Solution is a subtree that
    - Has a goal node at every leaf
    - Specifies one action at each of its state nodes
    - Includes every outcome branch at each of the chance nodes.
  - In previous example:
    - [Left, if AtL ∧ CleanL ∧ CleanR then [] else Suck]
  - For exact solutions: use minimax algorithm with 2 modifications:
    - Max and Min nodes become OR and AND nodes
    - Algorithm returns conditional plan instead of single move
And-Or Search Algorithm

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

function OR-SEARCH(state, problem, path) returns a conditional plan or failure
if GOAL-TEST(problem)(state) then return the empty plan
if state is on path then return failure
for action, state_set in SUCCESSORS(problem)(state) do
    plan ← AND-SEARCH(state_set, problem, [state | plan])
    if plan = failure then return failure
return [ if s1 then plan1 else if s2 then plan2 else … if sn then plann-1 else plann ]

function AND-SEARCH(state_set, problem, path) returns a conditional plan or failure
for each si in state_set do
    plan[i] ← OR-SEARCH(si, problem, path)
    if plan[i] = failure then return failure
return [ if s1 then plan1 else if s2 then plan2 else … if sn then plann-1 else plann ]

How to deal with cycles?
- When a state that already is on the path appears, return failure
- No non-cyclic solution
- Ensures algorithm termination
- The algorithm does not check whether some state is already on some other path from the root.

Sometimes Only a Cyclic Solution Exists
- e.g. triple murphy: sometimes the move is not performed
  [Left, if CleanL then [] else Suck] is not a solution
- Use label to repeat parts of plan (but infinite loops)
  [U: Left, if AtR then 1.1 else if CleanL then [] else Suck]

Conditional Planning and Partially Observable Environments
- Fully observable: conditional tests can ask any question and get an answer
- Partially observable???
  - The agent has limited information about the environment.
  - Modeled by a state-set = belief states
  - E.g. assume vacuum agent which can not sense presence or absence of dirt in squares other than the one it is in
  - + alternative murphy: dirt can be left behind when moving to other square.
  - Solution in fully observable world: keep moving left and right, sucking dirt whenever it appears until both squares are clean and I’m in square left.

Belief States
- Representation?
  - Sets of full state descriptions
    \{(AtR ∧ CleanR ∧ CleanL) ∨ (AtR ∧ CleanR ∧ ¬CleanL)\}
  - Logical sentences that capture the set of possible worlds in the belief state (OWA)
    AtR ∧ CleanR
  - Knowledge propositions describing the agent’s knowledge (CWA)
    K(AtR) ∧ K(CleanR)
Belief States
- Choices 2 and 3 are equivalent (let’s continue with 3)
- Symbols can appear in three ways:
  - positive,
  - negative or
  - unknown:
- $3^n$ possible belief states for $n$ proposition symbols.
- YET, set of belief sets is a power set of the physical states which is much larger than $3^n$
- Hence 3 is restricted as representation
- Any scheme capable of representing every possible belief state will require $O(2^n)$ bit to represent each one in the worst case.
- The current scheme only requires $O(n)$

Sensing in Conditional Planning
- How does it work?
- Automatic sensing
  - At every time step the agent gets all available percepts
- Active sensing
  - Percepts are obtained through the execution of specific sensory actions.
    - checkDirt and checkLocation
- Given the representation and the sensing, action descriptions can now be formulated

Monitoring and Replanning
- Execution monitoring: check whether everything is going as planned
  - Unbounded nondeterminacy: some unanticipated circumstances will arise
  - A necessity in realistic environments
- Kinds of monitoring:
  - Action monitoring: verify whether the next action will work
  - Plan monitoring: verify the entire remaining plan

Monitoring and Replanning
- When something unexpected happens: **replan!**
- To avoid too much time on planning try to **repair the old plan**.
- Applicable in both fully and partially observable environments, and in a variety of planning representations

Replanning Agent
```plaintext
function REPLANNING-AGENT(percept) returns an action
static; KB, a knowledge base (+ action descriptions)
plan, a plan initially []
whole_plan, a plan initially []
goal, a goal

TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
current ← STATE-DESCRIPTION(KB, t)
if plan = [] then return the empty plan
whole_plan ← PLANNER(current, goal, KB)
if PRECONDITIONS(FIRST(plan)) not currently true in KB then
  candidates ← SORT(whole_plan, ordered by distance to current)
  find state s in candidates such that
    failure ≠ repair ← PLANNER(current, s, KB)
    continuation ← the tail of whole_plan starting at s
    whole_plan ← plan ← APPEND(repair, continuation)
return POP(plan)
```

Repair Example
Repair Example: Painting

Init(\text{Color}(Chair, Blue) \land \text{Color}(Table, Green) \land 
\text{ContainsColor}(BC, Blue) \land \text{PaintCan}(BC) \land 
\text{ContainsColor}(RC, Red) \land \text{PaintCan}(RC))

Goal(\text{Color}(Chair,x) \land \text{Color}(Table,x))

Action(\text{Paint}(\text{object}, \text{color}))
PRECOND: \text{HavePaint(\text{color})}
EFFECT: \text{Color(\text{object}, \text{color})}
Action(\text{Open(\text{can})})
PRECOND: \text{PaintCan(\text{can})} \land 
\text{ContainsColor(\text{can},\text{color})}
EFFECT: \text{HavePaint(\text{color})}

[Start; Open(BC); Paint(Table,Blue), Finish]

Plan Monitoring

- Check the preconditions for success of the entire plan.
  - Except those which are achieved by another step in the plan.
  - Execution of doomed plan is cut off earlier.
- Limitation of replanning agent:
  - It can not formulate new goals or accept new goals in addition to the current one

Continuous Planning.

- Agent persists indefinitely in an environment
  - Phases of goal formulation, planning and acting
- Execution monitoring + planner as one continuous process
- Example: Blocks world
  - Assume a fully observable environment
  - Assume partially ordered plan

Block World Example

Initial state (a)
Action(\text{Move(\text{C},\text{D})})
PRECOND: \text{Clear(x)} \land \text{Clear(y)} \land \text{On(x,z)}
EFFECT: \text{On(x,y)} \land \text{Clear(z)} \land \neg \text{On(x,z)} \land \neg \text{Clear(y)}
- The agent first needs to formulate a goal: \text{On(C,D)} \land \text{On(D,B)}
- Plan is created incrementally, return \text{NoOp} and check percepts

Assume that percepts don’t change and this plan is constructed
Ordering constraint between \text{Move(D,B)} and \text{Move(C,D)}
Start is label of current state during planning.
Before the agent can execute the plan, nature intervenes:
D is moved onto B
Example (cont)

- Start contains now On(D,B)
- Agent perceives: Clear(B) and On(D,G) are no longer true
  - Update model of current state (start)
- Causal links from Start to Move(D,B) (Clear(B) and On(D,G)) no longer valid.
- Remove causal relations and two PRECOND of Move(D,B) are open
- Replace action and causal links to Finish by connecting Start to Finish.

Example (cont)

- Extending: whenever a causal link can be supplied by a previous step
- All redundant steps (Move(D,B) and their causal links) are removed from the plan
- Execute new plan, perform action Move(C,D)
  - This removes the step from the plan

Example (cont)

- Execute new plan, perform action Move(C,D)
  - Assume agent is clumsy and drops C on A
- No plan but still an open PRECOND
- Determine new plan for open condition
- Again Move(C,D)

Example (cont)

- Similar to POP
- On each iteration find plan-flaw and fix it
- Possible flaws: Missing goal, Open precondition, Causal conflict, Unsupported link, Redundant action, Unexecuted action, unnecessary historical goal

Multi-Agent Planning

- So far we only discussed single-agent environments.
- Other agents can simply be added to the model of the world:
  - Poor performance since agents are not indifferent of other agents’ intentions
- In general two types of multiagent environments:
  - Cooperative
  - Competitive

Cooperation: Joint Goals and Plans

- Multi-planning problem: assume double tennis example where agents want to return ball.

```
Agents(A,B)
Init(At(A,[Left, Baseline]) ∧ At(B,[Right, Net]) ∧ Approaching(Ball,[Right, Baseline]) ∧ Partner(A,B) ∧ Partner(B,A))
Goal(Returned(Ball) ∧ At(agent,[x,Net]))
Action(Hit(agent, Ball))
PRECED: Approaching(Ball,[x,y]) ∧ At(agent,[x,y]) ∧ Partner(agent, partner) ∧ ¬At(partner,[x,y])
EFFECT: Returned(Ball))
Action(Go(agent,[x,y]))
PRECED: At(agent,[a,b])
EFFECT: At(agent,[x,y]) ∧ ¬At(agent,[a,b]))
```
Cooperation: Joint Goals and Plans (II)

- A solution is a joint-plan consisting of actions for both agents.
- Example:
  A: \([\text{Go}(A,[\text{Right}, \text{Baseline}]), \text{Hit}(A,\text{Ball})]\)
  B: \([\text{NoOp}(B), \text{NoOp}(B)]\)

  Or
  A: \([\text{Go}(A,[\text{Left}, \text{Net}]), \text{NoOp}(A)]\)
  B: \([\text{Go}(B,[\text{Right}, \text{Baseline}]), \text{Hit}(B, \text{Ball})]\)

- Coordination is required to reach same joint plan.

Multi-Body Planning

- Planning problem faced by a single centralized agent that can dictate action to each of several physical entities.
- Hence not truly multiagent
- Important: synchronization of actions
  - Assume for simplicity that every action takes one time step and at each point in the joint plan the actions are performed simultaneously
  - Planning can be performed using POP applied to the set of all possible joint actions.
    - Size of this set???

Multi-Body Planning

- Alternative to set of all joint actions: add extra concurrency lines to action description
  - Concurrent action
    \(\text{Action}([\text{Hit}(A, \text{Ball})])\)
    \(\text{PRECOND}: \neg \text{Hit}(B, \text{Ball})\)
    \(\text{EFFECT}: \text{Returned}(\text{Ball})\)
  - Required actions (carrying object by two agents)
    \(\text{Action}([\text{Carry}(A, \text{cooler}, \text{here}, \text{there})])\)
    \(\text{CONCURRENT}: \text{Carry}(B, \text{cooler}, \text{here}, \text{there})\)
    \(\text{PRECOND}: \ldots\)

- Planner similar to POP with some small changes in possible ordering relations

Coordination Mechanisms

- To ensure agreement on joint plan: use convention.
  - Convention = a constraint on the selection of joint plans (beyond the constraint that the joint plan must work if the agents adopt it).
  - e.g. stick to your court or one player stays at the net.
  - Conventions which are widely adopted = social laws e.g. language.
  - Can be domain-specific or independent.
  - Could arise through evolutionary process (flocking behavior).

Flocking Example

- Three rules:
  - Separation:
    Steer away from neighbors when you get too close
  - Cohesion:
    Steer toward the average position of neighbors
  - Alignment:
    Steer toward average orientation (heading) of neighbors
- Flock exhibits emergent behavior of flying as a pseudo-rigid body

Coordination Mechanisms

- In the absence of conventions: Communication
  e.g. Mine! Or Yours! in tennis example
- The burden of arriving at a succesfull joint plan can be placed on
  - Agent designer (agents are reactive, no explicit models of other agents)
  - Agent (agents are deliberative, model of other agents required)
### Competitive Environments
- Agents can have conflicting utilities
  - e.g. zero-sum games like chess
- The agent must:
  - Recognise that there are other agents
  - Compute some of the other agents plans
  - Compute how the other agents interact with its own plan
  - Decide on the best action in view of these interactions.
- Model of other agent is required
- YET, no commitment to joint action plan.

### Summary
- Time, schedules and resources
- Hierarchical task network planning
- Non-deterministic domains
  - Conditional planning
  - Execution monitoring and replanning
  - Continuous planning
- Multi-agent planning
- Ch. 11