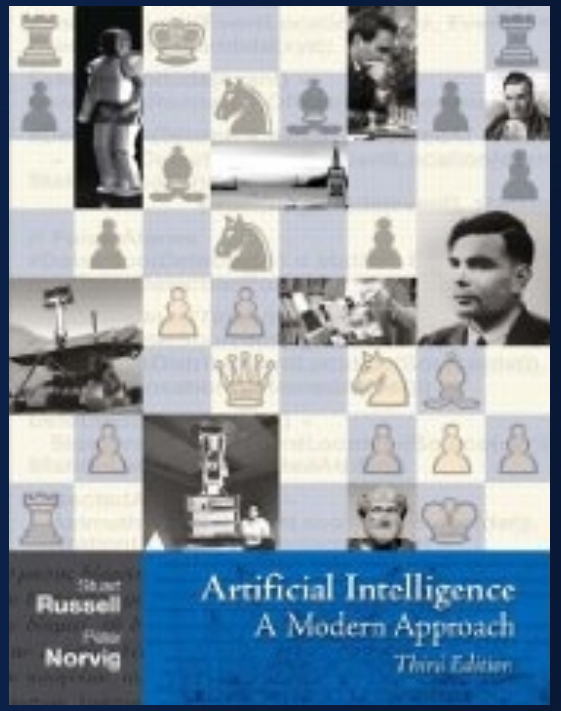


AI Fundamentals: planning

Maria Simi

Planning and acting in the real world

LESSON 3: PLANNING WITH CONSTRAINTS – HIERARCHICAL PLANNING – NON DETERMINISTIC PLANNING – MULTI-AGENT PLANNING

Summary

1. Actions with duration and resource constraints
2. Hierarchical planning
3. Planning and acting in nondeterministic domains
4. Planning in a multi-agent environments

Actions with duration and resource constraints

Combining planning and scheduling

- In classical planning we do not take into account the **duration** of the actions and **resource constraints**.
- These factors are important in **scheduling problems**.
- The most common approach is "*plan first, schedule later*". Divide the problem into:
  1. **A planning phase**: produces a partially ordered plan with a minimum set of ordering constraints on actions. We can use SATPLAN, GRAPHPLAN, POP …
  2. **A scheduling phase**: we take into account the duration of actions and other resources.

The language for scheduling problems

Jobs: a collection of actions with ordering constraints.

Constraints on actions:

- Duration
- Resources needed
  - the type (personnel, materials, tools, machinery …)
  - the quantity
  - consumable/reusable distinction. Resources can also be produced (negative consumption).

A solution must specify the start times for each action and must satisfy all the temporal ordering constraints and resource constraints.

A solution has a cost function: for simplicity we assume it is the **total duration** (**makespan**)

A job-shop scheduling problem

$Jobs(\{AddEngine1 \prec AddWheels1 \prec Inspect1\},$
$\{AddEngine2 \prec AddWheels2 \prec Inspect2\})$

$Resources(EngineHoists(1), WheelStations(1), Inspectors(2), LugNuts(500))$

$Action(AddEngine1, \text{DURATION}:30,$
$\text{USE}:EngineHoists(1))$
$Action(AddEngine2, \text{DURATION}:60,$
$\text{USE}:EngineHoists(1))$
$Action(AddWheels1, \text{DURATION}:30,$
$\text{CONSUME}:LugNuts(20), \text{USE}:WheelStations(1))$
$Action(AddWheels2, \text{DURATION}:15,$
$\text{CONSUME}:LugNuts(20), \text{USE}:WheelStations(1))$
$Action(Inspect_i, \text{DURATION}:10,$
$\text{USE}:Inspectors(1))$

- The assembly of two cars.
- Two jobs.
- Four types of resources and their quantities available a
- Action schemas, with *duration* and resources needed, their *use* or *consumption*, *quantities* (not individual resources ---a useful abstraction, **aggregation**)

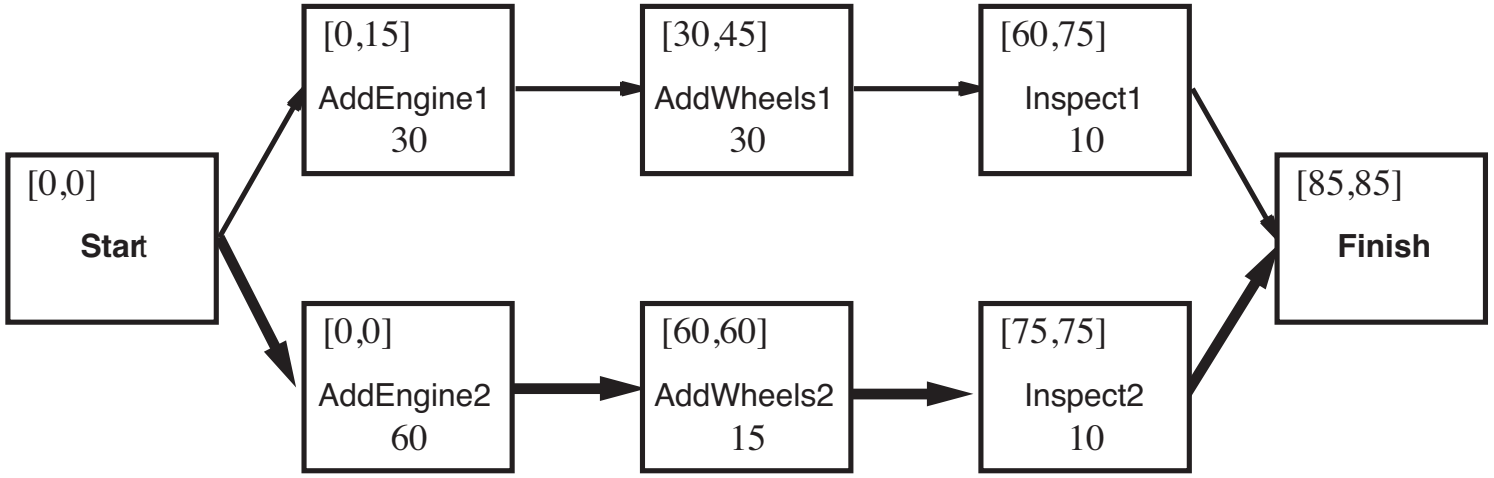
Solving scheduling problems

We start with temporal scheduling.

Ordering constraints can be visualized as a directed graph.

A path from **Start** to **Finish** is a linear sequence of actions. The **critical path** is the longest path (in bold). “Critical” because it conditions the duration of the whole plan.

Actions have a window for their execution, taking into account the earliest possible start time (**ES**) and the latest possible star time (**LS**). The quantity (LS – ES) is called the **slack** of the action.

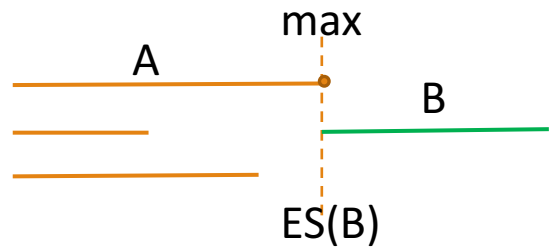


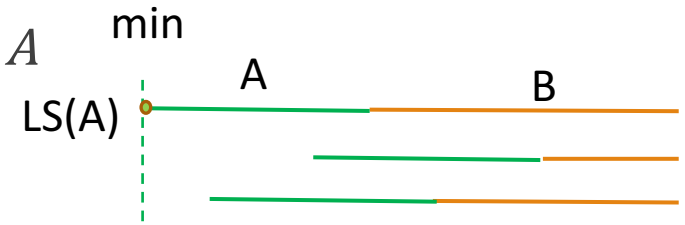
Critical Path Method (CPM)

The ES and LS times for all the actions constitute a **schedule** for the problem.

Definition:

- $\mathrm{ES}(Start) = 0$
- $\mathrm{ES}(B) = \max_{\mathrm{A}\prec\mathrm{B}} (\mathrm{ES}(A) + Duration(\mathrm{A}))$ for any action A preceding B
- $\mathrm{LS}(Finish) = \mathrm{ES}(\mathrm{Finish})$
- $\mathrm{LS}(A) = \min_{\mathrm{B}\succ\mathrm{A}} (\mathrm{LS}(B) - Duration(A))$ for any action B following A



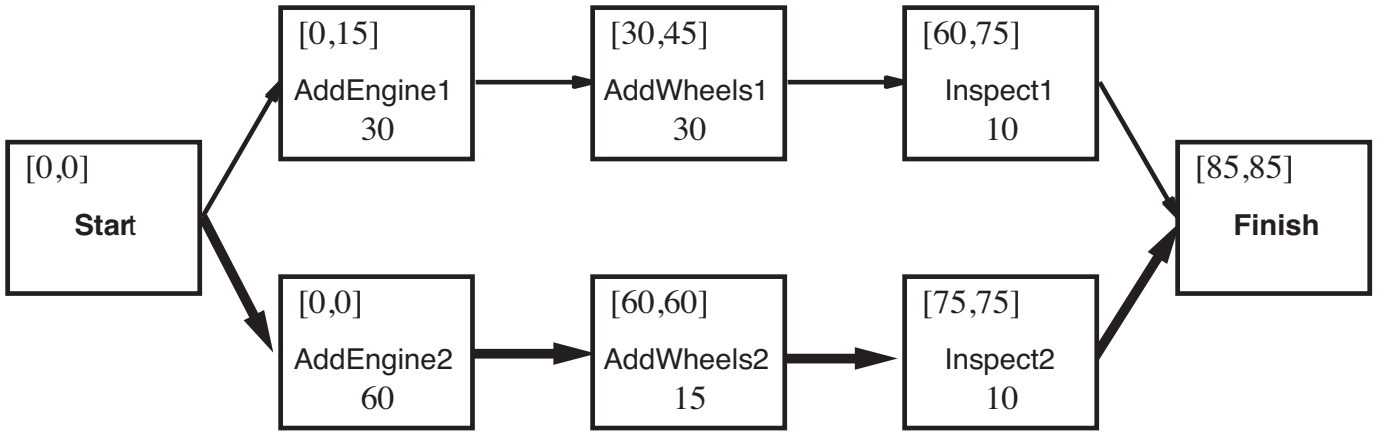


Method, going forward:

1. Start with $\mathrm{ES}(Start) = 0$
2. When we reach an action B such all preceding actions A have their ES, compute $\mathrm{ES}(B)$

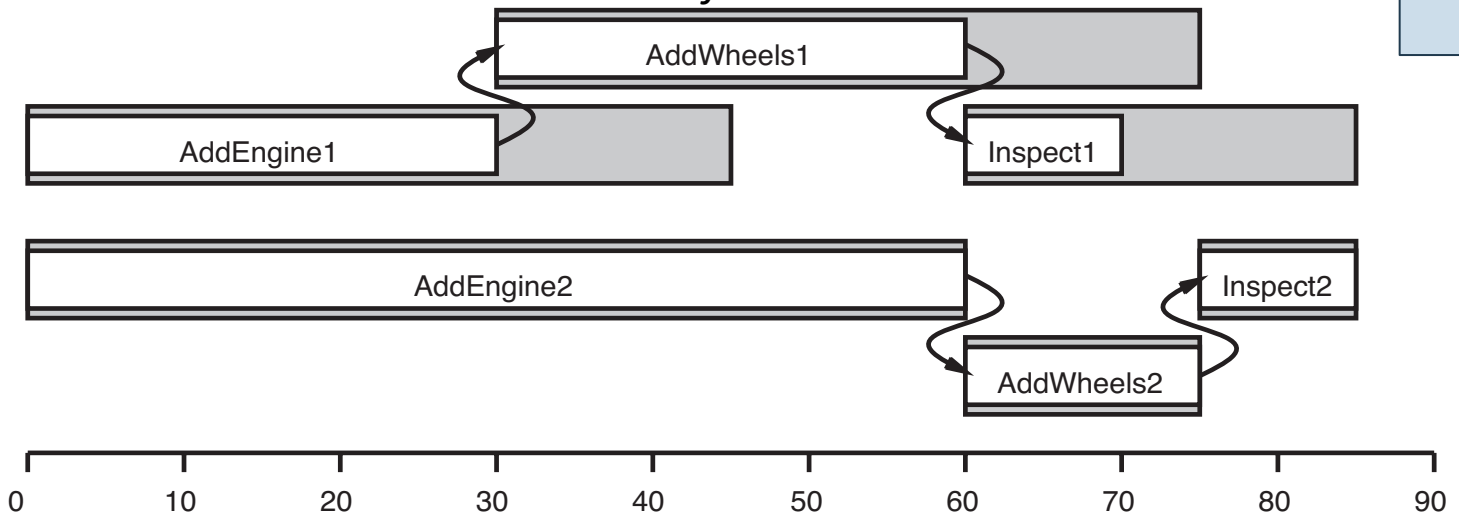
Going backword:

1. $\mathrm{LS}(Finish) = \mathrm{ES}(Finish)$
2. When we reach an action A such as all the following action have their $\mathrm{LS}(B)$, compute $\mathrm{LS}(A)$



Complexity: $O(Nb)$
- N is the number of actions
- b is the maximum branching factor into or out of an action.

Slack of action Addweels1

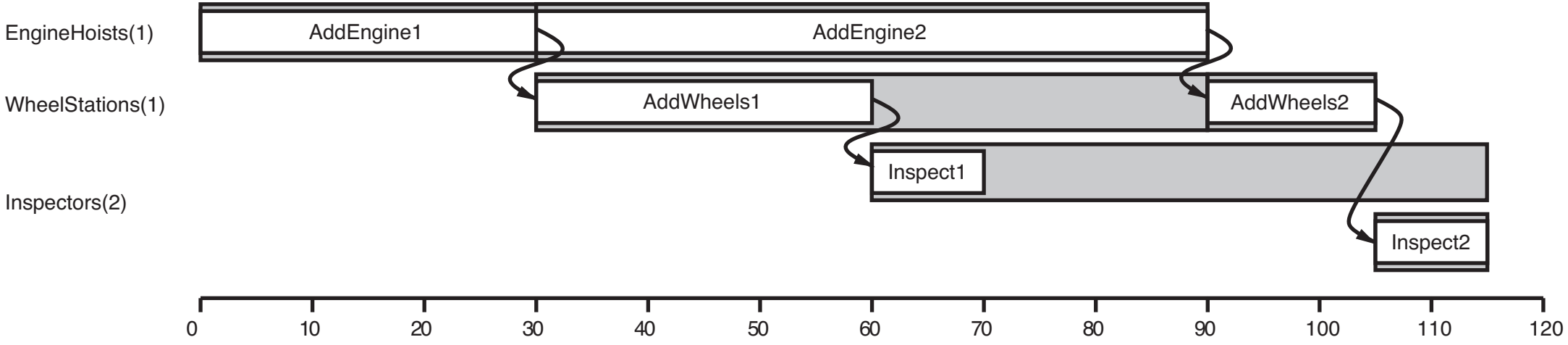


Critical path; no slacks

Introducing resource constraints

Introducing **resource constraints**, the resulting constraints on start and end times become more complicated.

For example *AddEngine1* and *AddEngine2* require the same tool and **cannot overlap**. Below the **shortest duration** plan, taking onto account reusable resources (these are reported on left).



Minimum slack algorithm

The “cannot overlap” constraint is a disjunction of two linear inequalities, one for each possible ordering.

The introduction of disjunctions turns out to make scheduling with resource constraints **NP-hard**.

The **minimum slack algorithm** is one of many.

The heuristic is the following; on each iteration:

- Select for scheduling the action **with the least slack** among unscheduled actions having all predecessors scheduled; schedule this action for the earliest possible start.
- then update the ES and LS times for each affected action and repeat.

The heuristic resembles the minimum-remaining-values (MRV) heuristic in constraint satisfaction.

Not optimal: on the example this heuristic does not find the shortest solution.

Hierarchical planning

Hierarchical decomposition

A strategy to dominate complexity is to work at different level of abstractions.

Humans do not plan at the level of single muscle activation. The complexity would be too high.

We could also plan at a high level and defer computation of more detailed plans until necessary for execution. Especially when dealing with non determinism.

Hierarchical decomposition is a winning organizing principle in complex organizations, software development and many other fields

High level actions

We are in the domain of **Hierarchical Task Networks** planning (HTN planning).

We distinguish actions in:

- **Primitive actions** (directly executable by the agent)
- **High Level Actions** (HLA)

Each HLA has one or more possible **refinements** in terms of a sequence of primitive or high level actions.

Specifying the decomposition humans have the possibility to express knowledge about their strategy

The navigate example (for the vacuum world) shows **recursive refinements**.

Going to SFO

$Refinement(Go(Home, SFO),$
STEPS: $[Drive(Home, SFOLongTermParking),$
$Shuttle(SFOLongTermParking, SFO)]\ )$
$Refinement(Go(Home, SFO),$
STEPS: $[Taxi(Home, SFO)]\ )$

The vacuum world

$Refinement(Navigate([a, b], [x, y]),$
PRECOND: $a = x \ \wedge \ b = y$
STEPS: $[]\ )$
$Refinement(Navigate([a, b], [x, y]),$
PRECOND: $Connected([a, b], [a - 1, b])$
STEPS: $[Left, Navigate([a - 1, b], [x, y])]\ )$
$Refinement(Navigate([a, b], [x, y]),$
PRECOND: $Connected([a, b], [a + 1, b])$
STEPS: $[Right, Navigate([a + 1, b], [x, y])]\ )$
...

High level plans

HLA's with only primitive actions are **implementations**.

Example: the sequences [$Right$, $Right$, $Down$] and [$Down$, $Right$, $Right$], in a grid world, both implement the HLA $Navigate([1, 3], [3, 2])$.

An implementation of a **high-level plan** (a sequence of HLAs) is the concatenation of implementations of each HLA in the sequence.

A high-level plan achieves the goal from a given state if **atleast one** of its implementations achieves the goal from that state.

When a HLA has multiple implementations, there are two options

1. to search for one of the primitive solution that works
2. to reason about correctness of abstract plans, without considering implementations

Searching for primitive solutions

We start with Act is the top level action and the goal is to find an implementation of Act.

Classical planning problems can be recursively defined as follows:

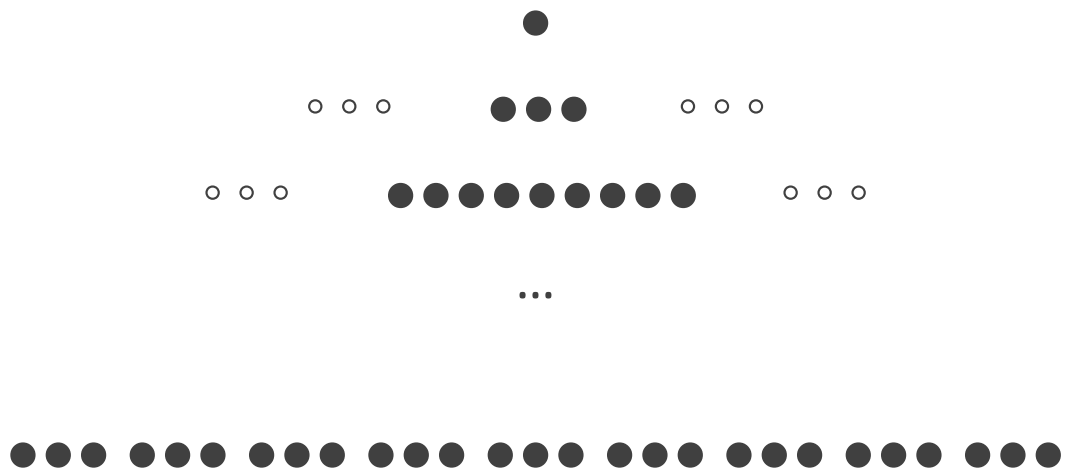
- for each primitive action a_i, provide a refinement of Act with steps $[a_i, Act]$. Proceed with $Act \leftarrow [a_i, Act]$.
- stop when Act, is achieved, i.e. admits a refinement with an empty list of steps.

One possible implementation based on **breadth-first** tree search follows.

Breadth first hierarchical search

```
function HIERARCHICAL-SEARCH(problem, hierarchy) returns a solution, or failure
    frontier ← a FIFO queue with [Act] as the only element                    [Go(Home, SFO)]
    loop do
        if EMPTY?(frontier) then return failure
        plan ← POP(frontier)  /* chooses the shallowest plan in frontier */
        hla ← the first HLA in plan, or null if none                          Go(Home, SFO)
        prefix,suffix ← the action subsequences before and after hla in plan
        outcome ← RESULT(problem.INITIAL-STATE, prefix)                       The state after prefix.
        if hla is null then  /* so plan is primitive and outcome is its result */
            if outcome satisfies problem.GOAL then return plan                You found a good primitive plan
        else for each sequence in REFINEMENTS(hla, outcome, hierarchy) do     Try all refinements
            frontier ← INSERT(APPEND(prefix, sequence, suffix), frontier)     Breadth first
```

Complexity of hierarchical planning



K refinements at each level

$\log_k d$ levels

d primitive actions

Number of nodes expanded, assuming one refinement for each HLA:

$$1 + k + k^2 + \ldots + k^{\log_k d - 1} = (d - 1)/(k - 1)$$ *see next slide for more detail*

Assuming r refinements for each of the k actions, the number of possible refinement trees is: $r^{(d-1)/(k-1)}$ to be compared with $O(b^d)$ of classical planning, where b is the number of actions at each level and d the length of the plan.

Keeping r small and k large can result in huge savings: a library of long primitive plans …

Justification

This computation requires some more detail:

$$1 + k + k^2 + \ldots + k^{\log_k d - 1} = (d-1)/(k-1)$$

Let $\#(n)$ be the number of nodes expanded to reach level *n*.

$$\#(n-1) = 1 + k + k^2 + \ldots + k^{n-1}$$

$$k\,\#(n) = 1 + k(1 + k + k^2 + \ldots + k^{n-1}) = 1 + k\,\#(n-1)$$

$$\#(n-1) = (\#(n) - 1)/k$$

$$\#(n-1) = (d + \#(n-1) - 1)/k \qquad \text{the nodes at the } n^{\text{th}} \text{ levels are } d$$

$$k\,\#(n-1) = d + \#(n-1) - 1$$

$$k\,\#(n-1) - \#(n-1) = d - 1$$

$$\#(n-1) = (d-1)/(k-1)$$

Searching for abstract solutions

[$Drive(Home, SFOLongTermParking)$, $Shuttle(SFOLongTermParking, SFO)$]

The question is: can we prove that this plan is correct *at an abstract level*?

We could add pre and post-conditions to HLA's and see that they are satisfied.

The **downward refinement property** for HLA descriptions guarantees that the high level plan has **at least one** correct implementation.

How to provide such a guarantee in case of multiple implementations? Since a single HLA can be implemented in different way, what is its global effect?

Example: the HLA $Go(Home, SFOLongTermParking)$ can be refined by two actions: $Drive(Home, SFOLongTermParking)$ and $Taxi(Home, SFO)$. The second one may require $Money$ as pre-condition. Can we guarantee the effect of Go?

Angelic semantics for HLA's

We are facing a non deterministic choice which is in control of the agent (**angelic** non-determinism) and not something which is decided by some external entity (**demonic** non-determinism). The agent has a solution and this is enough.

Angelic semantics relies on the notion of **reachable set of an HLA**:

> Given a state s, the reachable set for an HLA h, written as $\text{REACH}(s, h)$, is the set of states reachable by any of the HLA's implementations.

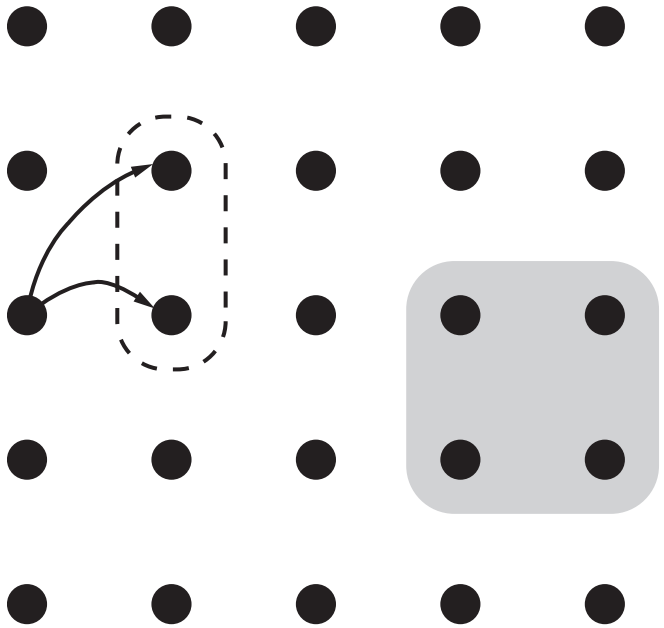
Reachable set of a sequence of HLA's:

> The reachable set of a sequence $[h_1, h_2]$ is the union of all the reachable sets of states obtained by applying h_2 in each state in the reachable set of h_1:
>
> $\text{REACH}(s, [h_1, h_2]) = \bigcup_{s' \in \text{REACH}(s, h_1)} \text{REACH}(s', h_2)$

Definition: A high-level plan—a sequence of HLAs—achieves the goal if its **reachable set intersects the set of goal states.**

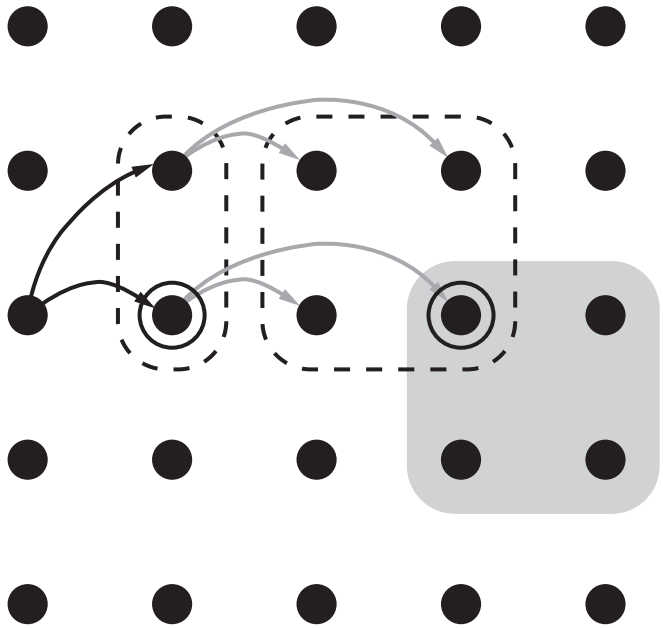
If the reachable set doesn't intersect the goal, then there is no plan.

A visualization of reachability



(a)

The grey area includes goal states. The dotted area is the reachable set of an HLA h_1 in a state s .



(b)

The reachable set for the sequence $[h_1, h_2]$. The sequence achieves the goal.

Angelic Search algorithm

Search among high level plans, looking for one whose **reachable set intersects the goal**; once that happens, the algorithm can commit to that abstract plan and focus on refining the plan further.

Problem: in many cases we can only approximate the effects because an HLA may have infinitely many implementations. This leads, skipping many details, to the definition of two approximate representation of reachable sets for h.

- An optimistic description $\text{REACH}^{+}(s, h)$ of h may overstate the reachable set.
- A pessimistic description $\text{REACH}^{-}(s, h)$ of h may understate the reachable set.

Thus the relation: $\text{REACH}^{-}(s, h) \subseteq \text{REACH}(s, h) \subseteq \text{REACH}^{+}(s, h)$

The algorithm fails when $\text{REACH}^{+}(s, h)$ does not intersect the goal and succeeds when $\text{REACH}^{-}(s, h)$ intersects the goal.

When $\text{REACH}^{+}(s, h)$ intersects and $\text{REACH}^{-}(s, h)$ does not, the algorithm needs to refine the plan and check whether a plan is possible.

Angelic-Search

```
function ANGELIC-SEARCH(problem, hierarchy, initialPlan) returns solution or fail
  frontier ← a FIFO queue with initialPlan as the only element
  loop do
    if EMPTY?(frontier) then return fail
    plan ← POP(frontier)  /* chooses the shallowest node in frontier */
    if REACH+(problem.INITIAL-STATE, plan) intersects problem.GOAL then
      if plan is primitive then return plan  /* REACH+ is exact for primitive plans */
      guaranteed ← REACH−(problem.INITIAL-STATE, plan) ∩ problem.GOAL
      if guaranteed≠{ } and MAKING-PROGRESS(plan, initialPlan) then
        finalState ← any element of guaranteed
        return DECOMPOSE(hierarchy, problem.INITIAL-STATE, plan, finalState)
      hla ← some HLA in plan
      prefix,suffix ← the action subsequences before and after hla in plan
      for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
        frontier ← INSERT(APPEND(prefix, sequence, suffix), frontier)
```

We can commit to the abstract plan

The plan exists

We need to refine the abstract plan

Planning and acting in nondeterministic domains

Planning and acting in nondeterministic domains

Consider partially observable, nondeterministic, and unknown environments.

1. **sensorless planning** (also known as **conformant** planning) for environments with no observations;
2. **contingent planning** for partially observable and nondeterministic environments;
3. **online planning** and **re-planning** for unknown environments.

The belief state in all these cases is **uncertain**. For this reason we need to change the representation of states.

An example

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

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

Goal: $Color(Chair, c) \wedge Color(Table, c)$ c is a variable

There are two actions:

1. removing the lid from a paint can
2. painting an object using the paint from an open can.

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

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

Note: c is not a variable of the schema

Percepts

We need to formalize perceptions, by means of **percept schema**:

$Percept(Color(x, c),$

PRECOND: $Object(x) \wedge InView(x))$

$Percept(Color(can, c),$

PRECOND: $Can(can) \wedge InView(can) \wedge Open(can))$

We also need an action $LookAt$:

$Action(LookAt(x),$

PRECOND: $InView(y) \wedge (x \neq y)$ *one at a time*

EFFECT: $InView(x) \wedge \neg InView(y))$

Sensorless planning

Backround knowledge: $Object(Table) \wedge Object(Chair) \wedge Can(C1) \wedge Can(C2)$

Initial state b_0: $Color(x, C(x))$ *any object has a color*

We cannot assume the CWA as in classical planning. States must also **include negative literals**. A possible plan is:

$$[RemoveLid(Can1), Paint(Chair, Can1), Paint(Table, Can1)]$$

Let's execute the plan:

$RemoveLid(Can1)$ in $b_0 \rightarrow b_1 = Color(x, C(x)) \wedge Open(Can1)$

$Paint(Chair, Can1)$ in $b_1 \rightarrow b_2 = Color(x, C(x)) \wedge Open(Can1) \wedge Color(Chair, C(Can1))$

$Paint(Table, Can1)$ in $b_2 \rightarrow$

$$b_3 = Color(x, C(x)) \wedge Open(Can1) \wedge Color(Chair, C(Can1)) \wedge Color(Table, C(Can1))$$

In this example the belief representation is always a conjunction of literals (1-CNF)

Actions with conditional effects

There may be actions whose effect depends on some aspect of the state.

Consider the Vacuum world with two locations.

Fluents are: AtL, AtR, $CleanL$, $CleanR$

$Action(Suck,$

EFFECT: **when** AtL: $CleanL$ ∧ **when** AtR: $CleanR$)

Not knowing location leads to uncertain beliefs. After $Suck$ the resulting belief state is:

$(AtL \wedge CleanL) \vee (AtR \wedge CleanR)$ no longer in 1-CNF.

Note: having two $Suck$ actions with different pre-conditions (AtL and AtR) does not work; we cannot determine applicability of the actions.

An approximation of the belief state is necessary.

AIMA discusses a few of them. Details are omitted.

Contingent planning

Contingent planning is the generation of plans with **conditional branching** based on percepts. It is appropriate for environments with partial observability, nondeterminism, or both. The action is selected on the basis of perceptions. Different from **conditional planning**.

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

Computation of the new belief state is done in two steps:

1. Compute the belief state as a result of actions (using add and delete lists)
2. Update le belief state according to perceptions

Online replanning

In contingent planning you have to consider all the contingencies that may arise.

An alternative, or addition, to contingent planning is **replanning** during execution.

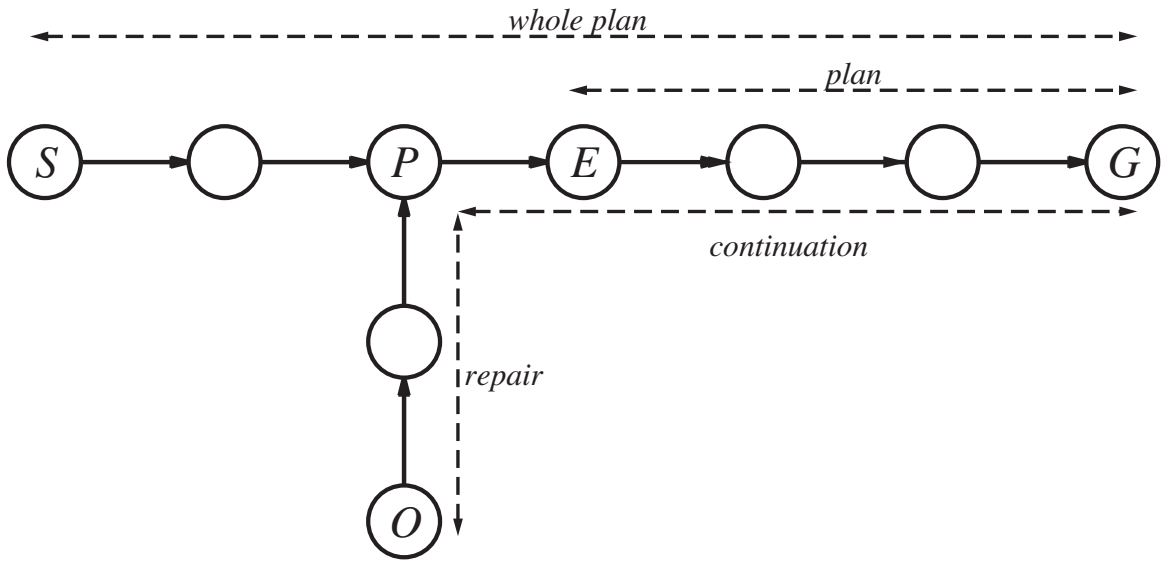
Different reasons for replanning:

- Inaccurate model (missing preconditions, missing effect, missing state variables …)
- exogenous events

It requires some **execution monitoring**:

- Action monitoring (preconds ok?)
- Plan monitoring (rest of plan still ok?)
- Goal monitoring (better goal?)

Plan execution monitoring also allows for **serendipity,** accidental achievement of the goal by other means,



Planning in multi-agent scenarios

Multi-agent planning

In **multi-agent planning** problems an agent tries to achieve its own goals with the help or despite the interference of others.

There are intermediate scenarios in the spectrum from a single agent to a multi-agent scenario made of autonomous agents:

- multi-effectors planning: the centralized plan has to control different effectors
- multi-body planning: overall plan executed by different bodies, decentralized planning
- multi-agent planning:
  - same goal but different plans, coordination is needed;
  - different goals as players in opposite teams in competitive environments , e.g. tennis or soccer.
  - a mixture …

The issues involved in multi-agent planning:

1. representing and planning for multiple simultaneous actions
2. cooperation, coordination, competition in multi-agent planning

Multiple simultaneous actions

Multi-actor planning: where actors is a generic term covering effectors, bodies, and agents themselves.

The different entities share a common goal and collaborate to achieve it. The planning is done centrally. We assume **perfect synchronization**.

Transition model:

The single action a is replaced by a joint action $\langle a_1, \ldots, a_n \rangle$, where a_i is the action taken by the i^{th} actor.

Two big problems, given that b is the number of possible actions:

1. we have to describe the transition model for b^n different joint actions;
2. we have a joint planning problem with a branching factor of b^n.

The major concern is to **decouple** the actors so that each one can work independently to one sub-problem.

Loosely coupled sub-problems

The standard approach to loosely coupled problems is to pretend the problems are **completely decoupled** and then **fix up** the interactions.

Double tennis problem:

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

Controlling interactions

The following joint plan works:

PLAN 1:

A : [$Go(A, RightBaseline)$, $Hit(A, Ball)$]

B : [$NoOp(B)$, $NoOp(B)$]

But in general, to restrict unwanted interactions we need to augment action schemas with a **concurrent action** list stating which actions must or must not be executed concurrently.

The *Hit* action can be described as follows:

$Action(Hit(a, Ball)$,

CONCURRENT: $b \neq a \Rightarrow \neg Hit(b, Ball)$

PRECOND: $Approaching(Ball, loc) \wedge At(a, loc)$

EFFECT: $Returned(Ball))$

Conversely, for some actions the desired effect is achieved only when another action occurs concurrently. Example: carrying a heavy piece of furniture.

Multiple agent: cooperation and coordination

Each agent **makes its own plan**. We assume that the goals and knowledge base are shared. There may be different plans for one goal.

PLAN 1:

$A : [Go(A, RightBaseline), Hit(A, Ball)]$

$B : [NoOp(B), NoOp(B)]$

PLAN 2:

$A : [Go(A, LeftNet), NoOp(A)]$

$B : [Go(B, RightBaseline), Hit(B, Ball)]$

If agents choose different plans, the combined effect does not work.

There is an issue of coordination. Different ways:

1. Conventions and social laws, i.e. “stick to your side of the court”
2. Communication, i.e. a tennis player could shout “Mine!”
3. Plan recognition: recognize the joint plan that the other agent is starting to execute and behave accordingly.

Evolutionary conventions and flocking

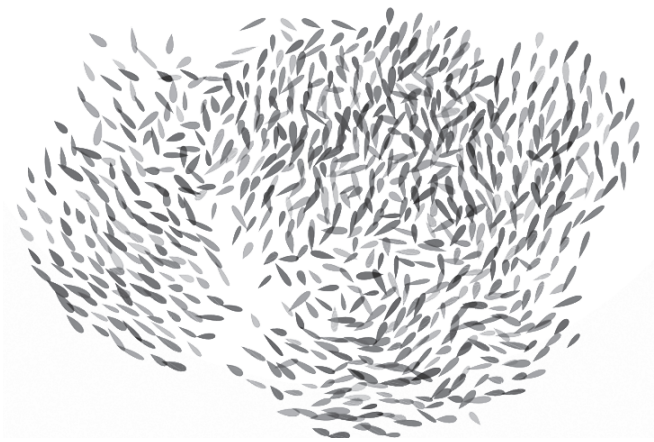
In both cases the behavior of animals in nature is a source of inspiration.

1. Conventions developed through evolutionary processes.
   - Colonies of ants execute very elaborate joint plans without any centralized control.
   - They do so playing very specific roles and strategies apparently with limited communication.

2. Flocking behavior of birds

   Maximize the weighted sum of three components:
   - Cohesion
   - Separation
   - Alignment

   Emergent behavior of flying as a pseudo-rigid body (more on **Computational neuroscience**)



(a)

Flocking simulation by local interactions

(b)

Flocking behavior

Conclusions

- ✓ Interaction between planning and scheduling taking into account actions with duration and resource constraints
- ✓ Hierarchical planning and decomposition is fundamental for managing the complexity (HTN planning).
- ✓ We only scratched the surface of other important areas: planning and acting in nondeterministic domains (estimation of belief states, perceiving and acting, online planning) and multi-agent environments.
- ✓ Important is also the issue of storing and reutilizing plans (kind of learning): either in abstract form (**explanation based learning**) or as concrete plans to be retrieved by similarity (**case based planning**). More on AIMA chapter 19.

Your turn

- ✓ Planning with resource constraints. You may look at some popular planners such as SAPA (Do and Kambhampati, 2001), T4 (Haslum and Geffner, 2001).
- ✓ HTN planning. Discuss approaches.
- ✓ O-PLAN (Bell and Tate, 1985), combines HTN planning with scheduling. Presentation.
- ✓ Reusing plans: *Explanation based learning* or *Case based planning*
- ✓ Non deterministic planning approaches.
- ✓ Multi-agent planning approaches.

References

- ✓ Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach* (3rd edition). Pearson Education 2010 [Chapter 10]