PLANNING

REAL-WORLD PROBLEMS

- Planning: the task of coming up with a sequence of actions that will achieve a goal
  - Search-based problem-solving agent
  - Logical planning agent
  - Complex/large scale problems?

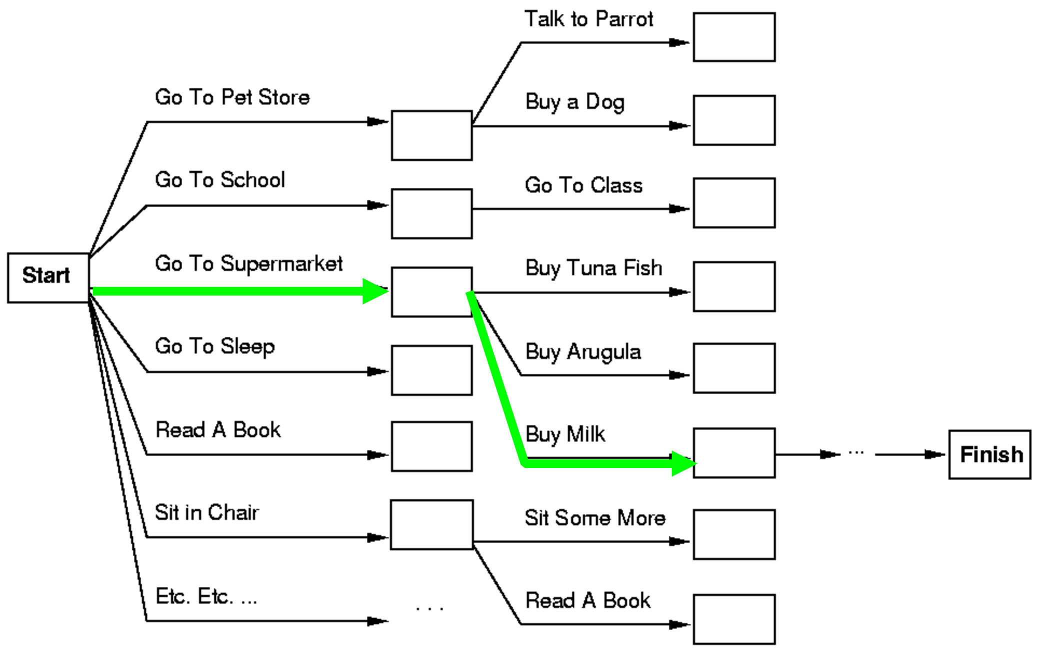
- For the discussion, we consider **classical planning environments** that are fully observable, deterministic, finite, static and discrete (in time, action, objects and effects)

PROBLEMS WITH STANDARD SEARCH

- Overwhelmed by irrelevant actions
- Finding a good heuristic function is difficult
- Cannot take advantage of problem decomposition

EXAMPLE

- Consider the task: get milk, bananas, and a cordless drill
  - Standard search algorithms seem to fail miserably
  - Why? Huge branching factor & heuristics



PROBLEM DECOMPOSITION

- Perfectly decomposable problems are delicious but rare
  - Partial-order planner is based on the assumption that most real-world problems are **nearly decomposable**
  - Be careful, working on some subgoal may undo another subgoal

PLANNING VS. PROBLEM SOLVING

- Planning agent is very similar to problem solving agent
  - Constructs plans to achieve goals, then executes them
- Planning agent is different from problem solving agent in:
  - Representation of goals, states, actions
  - Use of explicit, logical representations
  - Way it searches for solutions

PLANNING VS. PROBLEM SOLVING

- Planning systems do the following:
  - divide-and-conquer
  - relax requirement for sequential construction of solutions

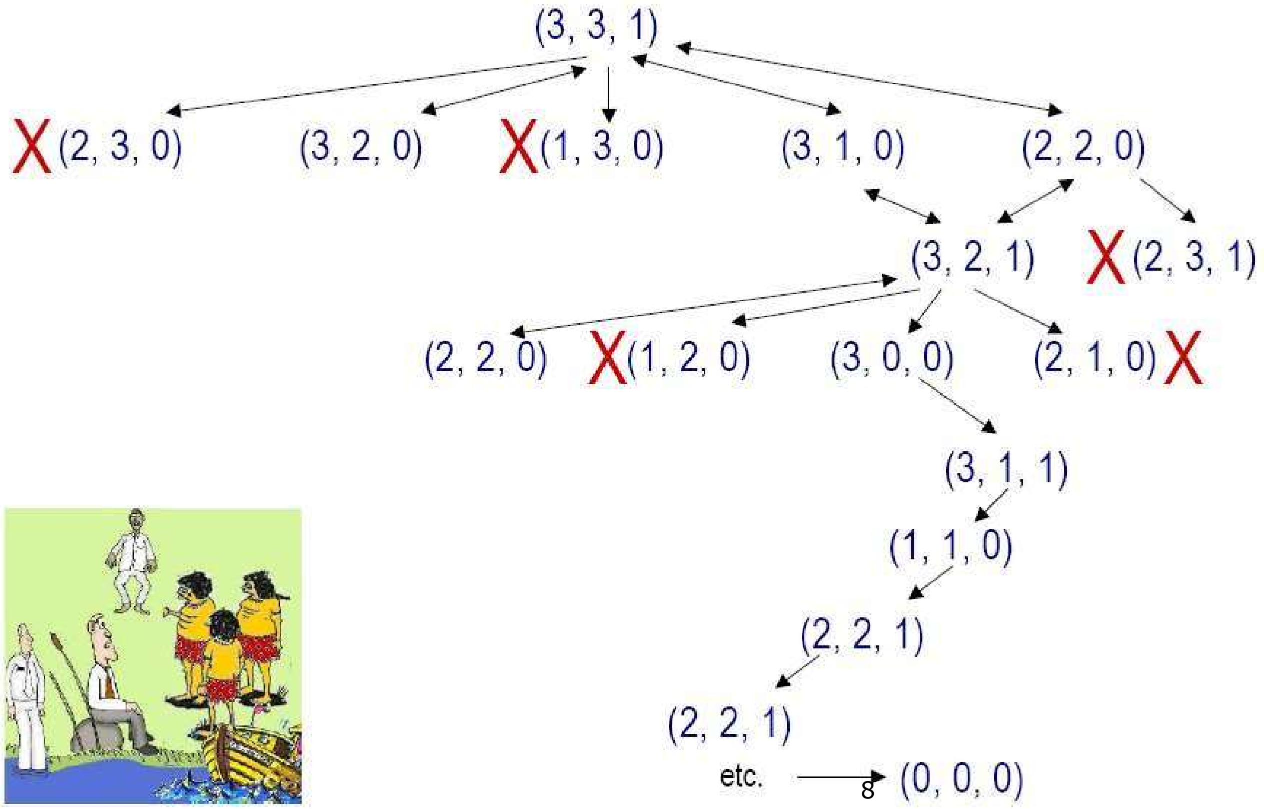
	Problem Sol.	**Planning**
States	data structures	logical sentences
Actions	code	preconditions/outcomes
Goal	code	logical sentences
Plan	sequence from s_0	constraints on actions

FAMOUS PROBLEM SOLVER TASK: “MISSIONARIES AND CANNIBALS”

In the missionaries and cannibals problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people,

under the constraint that, for both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals

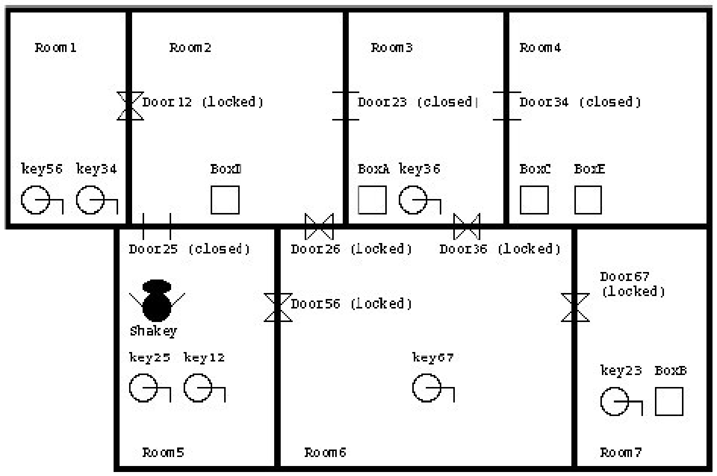
(if they were, the cannibals would eat the missionaries).



PLANNING-BASED APPROACH TO ROBOT CONTROL

- Job of planner: generate a goal to achieve, and then construct a plan to achieve it from the current state

- Must define representations of:
  - Actions: generate successor state descriptions by defining preconditions and effects
  - States: data structure describing current situation
  - Goals: what is to be achieved
  - Plans: solution is a sequence of actions



DEFINITION OF CLASSICAL PLANNING

Classical planning is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.

PDDL: planning researchers have invested in a factored representation using a family of languages called PDDL, the Planning Domain Definition Language (Ghallab et al., 1998)

PDDL

In PDDL, a state is represented as a conjunction of ground atomic fluents.

- "ground" means no variables,
- "fluent" means an aspect of the world that changes over time, and

- "ground atomic" means there is a single predicate, and if there are any arguments, they must be constants.
- For example,
  - Poor^Unknown might represent the state of a hapless agent,
  - At(Truck1;Melbourne)^At(Truck2;Sydney) could represent a state in a package delivery problem.

PDDL uses database semantics:

- the closed-world assumption means that any fluents that are not mentioned are false, and
- the unique names assumption means that Truck1and Truck2 are distinct.

ACTION SCHEMA

An **action schema** represents a family of ground actions.

For example, here is an action schema for flying a plane from one location to another:

Action(*Fly*(*p;from;to*)*;*
PRECOND: *At*(*p;from*)^*Plane*(*p*)^*Airport*(*from*)^*Airport*(*to*)
EFFECT:*:* *At*(*p;from*)^*At*(*p;to*))

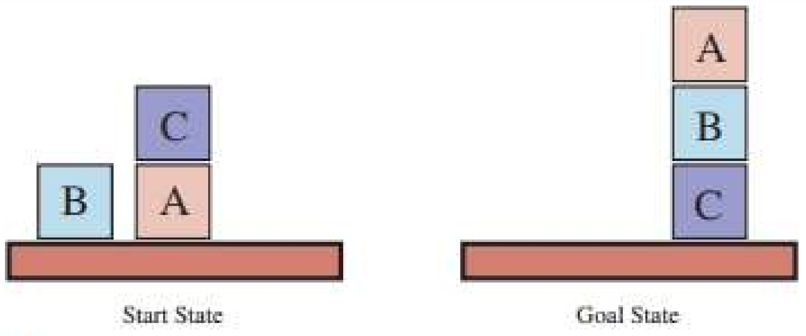


Figure 11.3 Diagram of the blocks-world problem in Figure 11.4.

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table))$
$Goal(On(A, B) \land On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$\qquad (b \neq x) \land (b \neq y) \land (x \neq y),$
$\quad$ EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land Block(x),$
$\quad$ EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$

Figure 11.4 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$.

ALGORITHMS FOR CLASSICAL PLANNING

Forward state-space search for planning
We can solve planning problems by applying any of the heuristic search algorithms

Backward search for planning
In backward search (also called **regression search**) we start at the goal and apply the actions backward until we find a sequence of steps that reaches the initial state.

At each step we consider **relevant actions** (in contrast to forward search, which considers actions that are **applicable**). This reduces the branching factor significantly, particularly in domains with many possible actions

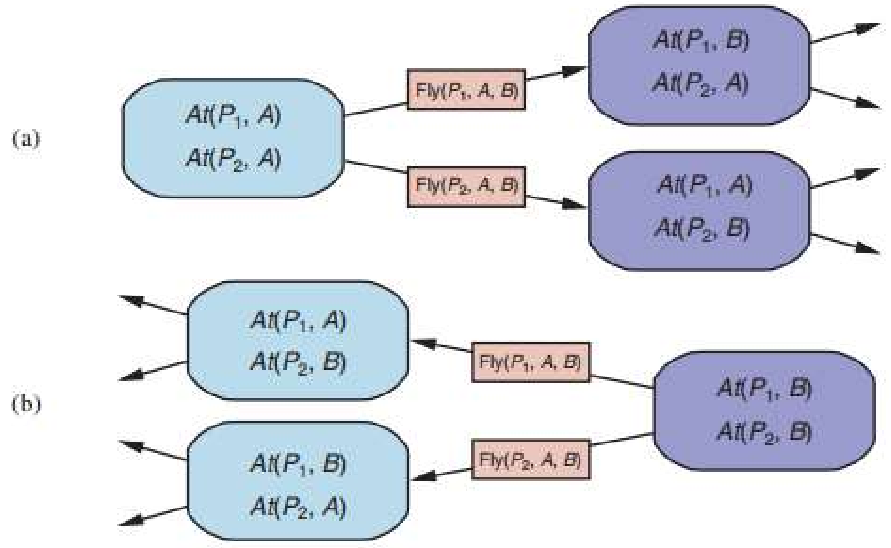


Figure 11.5 Two approaches to searching for a plan. (a) Forward (progression) search through the space of ground states, starting in the initial state and using the problem's actions to search forward for a member of the set of goal states. (b) Backward (regression) search through state descriptions, starting at the goal and using the inverse of the actions to search backward for the initial state.

HEURISTICS FOR PLANNING

An **admissible heuristic** can be derived by defining a **relaxed problem** that is easier to solve.

A search problem is a graph where the nodes are states and the edges are actions.

The problem is to find a path connecting the initial state to a goal state.
There are two main ways we can relax this problem to make it easier:

- by adding more edges to the graph, making it strictly easier to find a path, or
- by grouping multiple nodes together, forming an abstraction of the state space that has fewer states, and thus is easier to search

HIERARCHICAL PLANNING

Hierarchy of actions

- In terms of major action or minor action

Lower level activities would detail more precise steps for accomplishing the higher level tasks.

EXAMPLE

- Planning for ”Going to Goa this Cristmas”
  - Major Steps :
    - Hotel Booking
    - Ticket Booking
    - Reaching Goa
    - Staying and enjoying there
    - Coming Back
  - Minor Steps :
    - Take a taxi to reach station / airport
    - Take photos

MOTIVATION

Reduces the size of search space

- Instead of having to try out a large number of possible plan ordering, plan hierarchies limit the ways in which an agent can select and order its primitive operators

- If entire plan has to be synthesized at the level of most detailed actions, it would be ***impossibly long.***
- Natural to 'intelligent' agent

GENERAL PROPERTY

Postpone attempts to solve mere details, until major steps are in place.

Higher level plan may run into difficulties at a lower level, causing the need to return to higher level again to produce appropriately ordered sequence.

PLANNER

- Identify a hierarchy of conditions
- Construct a plan in levels, postponing details to the next level
- Patch higher levels as details become visible
- Demonstrated using ABSTRIPS

Ref : [1,2]

HIERARCHICAL DECOMPOSITION

Hierarchical decomposition, or hierarchical task network (**HTN**) planning, uses **abstract operators** to **incrementally** decompose a planning problem from a **high-level goal** statement to a **primitive plan network**

Primitive operators represent actions that are **executable**, and can appear in the final plan

Non-primitive operators represent **goals** (equivalently, **abstract actions**) that require further decomposition (or *operationalization*) to be executed

There is no "right" set of primitive actions: One agent's goals are another agent's actions!

TIME, SCHEDULES, AND RESOURCES

Classical planning talks about what to do, in what order, but does not talk about time:

- how long an action takes and when it occurs.

For example, in the airport domain we could produce a plan saying

- what planes go where,
- carrying what,
- but could not specify departure and arrival Scheduling times.

This is the subject matter of scheduling.
Resource constraint

- The real world also imposes resource constraints:
- an airline has a limited number of staff, and staff who are on one flight cannot be on another at the same time.

PLANNING VS. SCHEDULING

Planning: given one or more goals, generate a sequence of actions to achieve the goal(s)

Scheduling: given a set of actions and constraints, allocate resources and assign times to the actions so that no constraints are violated

Traditionally, **planning** is done with **specialized logical reasoning methods**

Traditionally, **scheduling** is done with **constraint satisfaction, linear programming, or OR methods**

However, **planning and scheduling are closely interrelated** and can't always be separated

REPRESENTING TEMPORAL AND RESOURCE CONSTRAINTS – JOB SCHEDULING

job-shop scheduling problem:

consists of a set of jobs, each of which has a collection of actions with ordering constraints among them.

Each action has a duration and a set of resource constraints required by the action.

A constraint specifies

- a type of resource (e.g., bolts, wrenches, or pilots),
- the number of that resource required, and
- whether that resource is consumable (e.g., the bolts are no longer available for use) or reusable (e.g., a pilot is occupied during a flight but is available again when the flight is over).

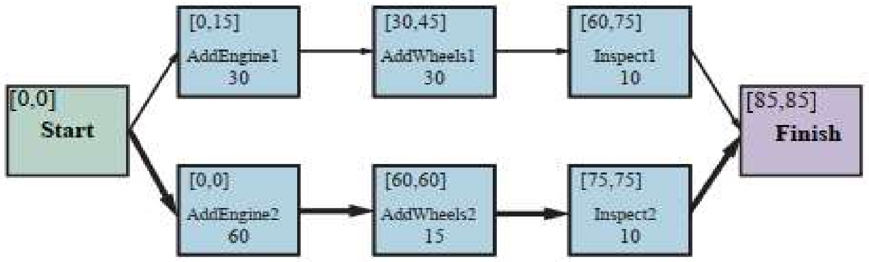
Actions can also produce resources (e.g., manufacturing and resupply actions).

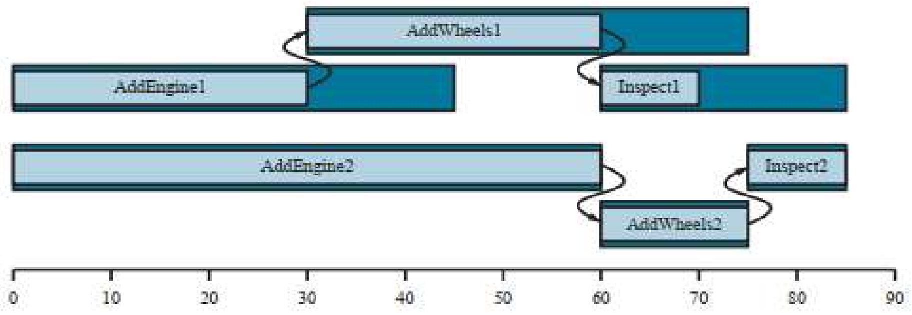
JOB SHOP SCHEDULING

A solution to a job-shop scheduling problem specifies the start times for each action and must satisfy all the temporal ordering constraints and resource constraints.

Makespan:

- the cost function is just the total duration of the plan,





CRITICAL PATH METHOD

Critical path method (CPM) to this graph to determine the possible start and end times of each action.

A **path** through a graph representing a partial-order plan is a linearly ordered sequence of actions beginning with *Start* and ending with *Finish*.

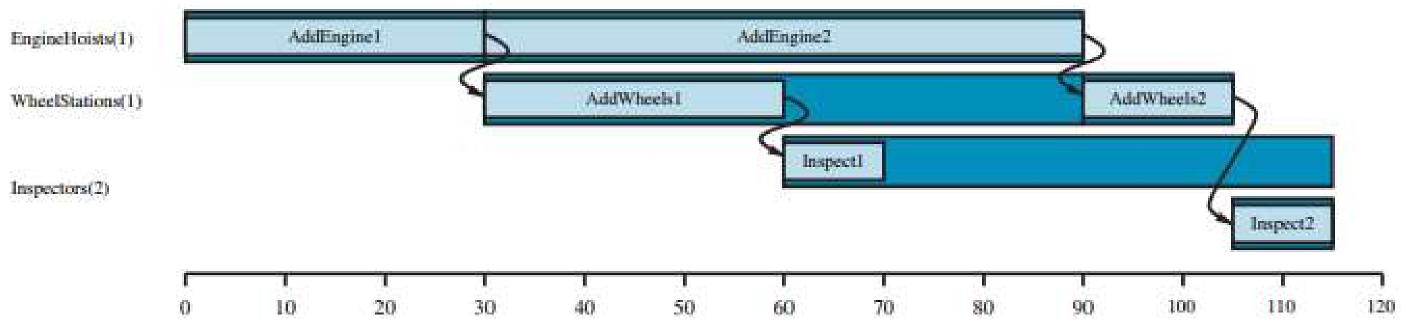


Figure 11.15 A solution to the job-shop scheduling problem from Figure 11.13, taking into account resource constraints. The left-hand margin lists the three reusable resources, and actions are shown aligned horizontally with the resources they use. There are two possible schedules, depending on which assembly uses the engine hoist first; we've shown the shortest-duration solution, which takes 115 minutes.