Planning and Search

Classical Planning

Outline

- ♢ Search vs. planning
- ♢ STRIPS operators
- ♢ PDDL
- ♢ Forward (progression) state-space search
- ♢ Backward (regression) relevant-states search

Planning

Planning is the process of computing several steps of a problem-solving procedure before executing any of them

This problem **can** be solved by search

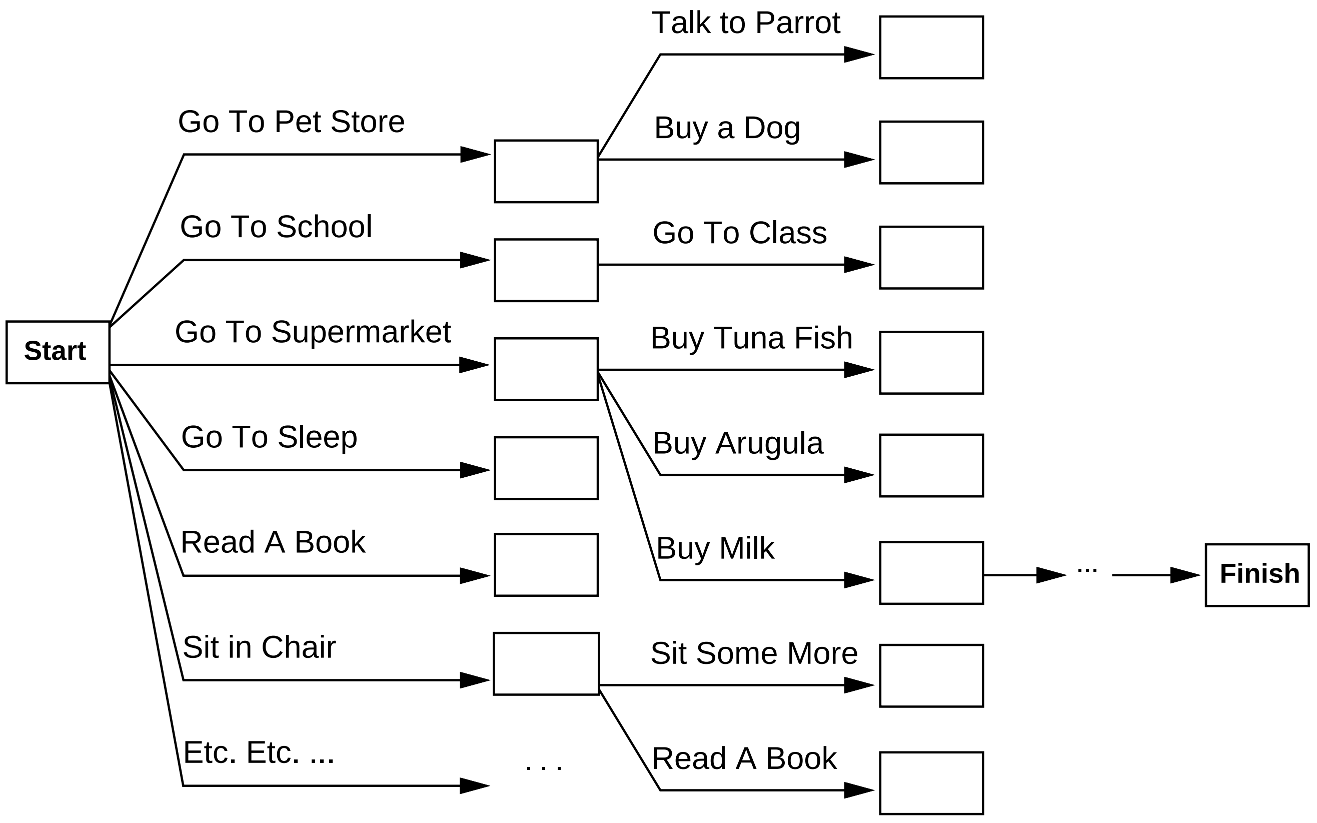
The main difference between search and planning is the representation of states

In search, states are represented as a single entity (which may be quite a complex object, but its internal structure is not used by the search algorithm)

In planning, states have structured representations (collections of properties) which are used by the planning algorithm

Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*
Standard search algorithms seem to fail miserably:



After-the-fact heuristic/goal test inadequate

Search vs. planning contd.

Planning systems do the following:

1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

	Search	**Planning**
States	data structures	Logical sentences
Actions	code	Preconditions/outcomes
Goal	code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

Classical planning

Assumptions are:

(1) Environment is deterministic

(2) Environment is observable

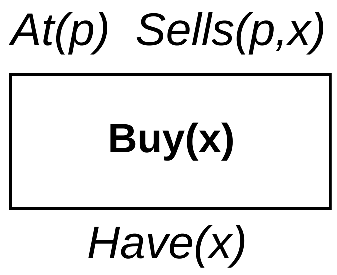
(3) Environment is static (it only in response to the agent's actions)

STRIPS operators

STRIPS planning language (Fikes and Nilsson, 1971)

Tidily arranged actions descriptions, restricted language

Action: $Buy(x)$
Precondition: $At(p), Sells(p, x)$
Effect: $Have(x)$



[Note: this abstracts away many important details!]

Restricted language $\Rightarrow$ efficient algorithm
Precondition: conjunction of positive literals
Effect: conjunction of literals

PDDL

Planning Domain Definition Language

A bit more relaxed that STRIPS

Preconditions and goals can contain negative literals

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

is called an action schema

Planning domain

States are sets of fluents (ground, functionless atoms). Fluents which are not mentioned are false (this is called closed world assumption).

$a \in \textsc{Actions}(s)$ iff $s \models \textsc{Precond}(a)$

$\textsc{Result}(s, a) = (s - \textsc{Del}(a)) \cup \textsc{Add}(a)$

where $\textsc{Del}(a)$ is the list of literals which appear negatively in the effect of a, and $\textsc{Add}(a)$ is the list of positive literals in the effect of a.

Example (slightly modified)

Action: $Buy(x)$
Precondition: $At(p), Sells(p,x), Have(Money)$
Effect: $Have(x), \neg Have(Money)$

$\textsc{Del}(Buy(Jaguar)) = \{Have(Money)\}$

$\textsc{Add}(Buy(Jaguar)) = \{Have(Jaguar)\}$

If $s = \{At(JDealer), Sells(JDealer, Jaguar), Blue(Sky), Have(Money)\}$,

$Buy(Jaguar) \in \textsc{Actions}(s)$

$\textsc{Result}(s, Buy(Jaguar) = (s - \{Have(Money)\}) \cup \{Have(Jaguar)\}$

$= \{At(JDealer), Sells(JDealer, Jaguar), Blue(Sky), Have(Jaguar)\}$

Planning problem

Planning problem = planning domain + initial state + goal

Goal is a conjunction of literals: $Have(Jaguar) \land \neg At(Jail)$

Can solve planning problem using search

Backward and forward search

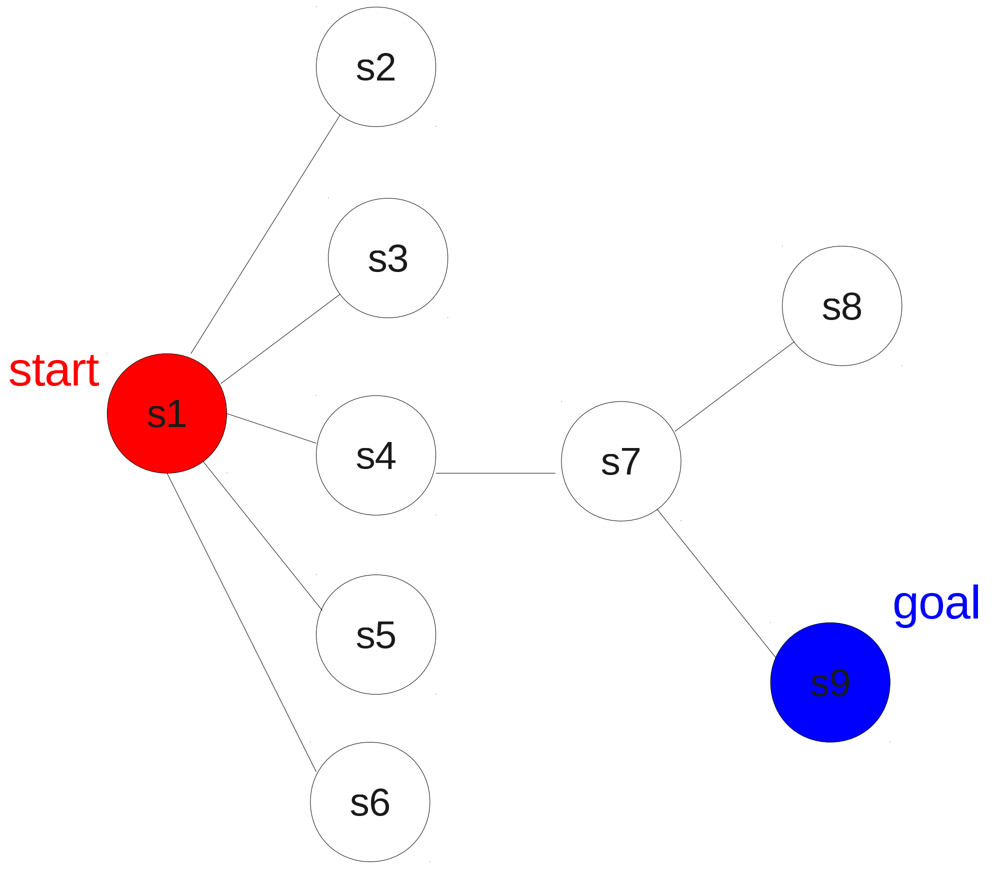
So far in the search lectures we only looked at forward search from the initial state to a goal state

Nothing prevented us from searching from a goal state to the initial state

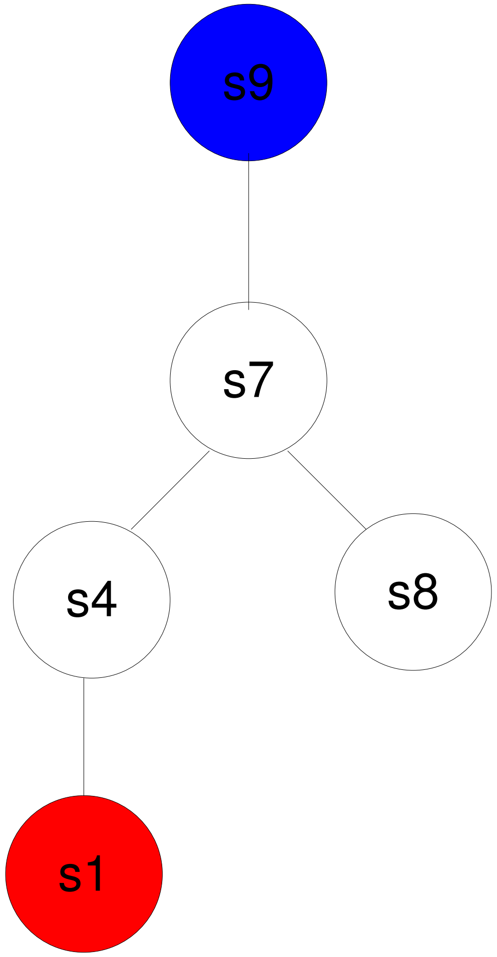
Sometimes given the branching factor it is more efficient to search backward

Motivating example: imagine trying to figure out how to get to some small place with few traffic connections from somewhere with a lot of traffic connections

Example



Example: backward search tree



Backward search

Can use any search method, breadth-first or depth-first or iterative deepening or A^*...

If there are several goal states, search backwards from each in turn

Planning can use both forward and backward search (progression and regression planning)

Simple example of forward planning

Planning domain:

Predicates: At, Sells, Have

Two action schemas:

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p,x), Have(Money)$
EFFECT: $Have(x), \neg Have(Money)$

ACTION: $Go(x,y)$
PRECONDITION: $At(x)$
EFFECT: $At(y)$, $\neg At(x)$

Simple example of forward planning 2

Planning problem: planning domain above plus

Objects: $Money$, J (for Jaguar), $Home$, G (for Garage)

Initial state: $At(Home) \wedge Have(Money) \wedge Sells(G, J)$

Goal state: $Have(J)$

Note: state descriptions are always ground (no variables). Goal description may have variables: $At(x) \wedge Have(y)$. A property with a variable such as $At(x)$ is satisfied at a state if there is a way of substituting an object for x so that the resulting formula is true in the state. An atomic ground formula $At(Home)$ is true iff it is in the state description. A negation of a ground atom $\neg At(G)$ is true iff the atom $At(G)$ is not in the state description.

Simple example of forward planning 3

```
--------------                --------------         -----------
| At(Home)    | Go(Home,G)   |At(Garage) | Buy(J)     |At(Garage)|
| Have(Money)|----------->   |Have(Money)|---------->  |Have(J)   |
| Sells(G,J) |                |Sells(G,J) |            |Sells(G,J)|
--------------                --------------         ------------
```

Go(Home,Home)
applicable and
does not change
the state)

Go(Garage,Home) and
Go(Garage,Garage)
also available

Buy(x) not
available for
any x (don’t have
Sells(Home, x))

Backward (regression) planning

Also called relevant-states search

Start at the goal state(s) and do regression (go back).

To be precise, there we start with a ground goal description g which describes a set of states (all those where $Have(J)$ holds but $Have(Money)$ may or may not hold, for example).

Backward (regression) planning 2

Given a goal description g and a ground action a, the regression from g over a gives a state description g':

$g' = (g - \textsc{Add}(a)) \cup \{\textsc{Precond}(a)\}$

For example, if the goal is $Have(J)$

$g' = (\{Have(J)\} - \{Have(J)\}) \cup$

$\{At(p), Sells(p, J), Have(Money)\} =$

$\{At(p), Sells(p, J), Have(Money)\}$

note that g' is partially uninstantiated (p is a free variable). In our example, there is only one match for p, namely G, but in general there may be several.

Backward (regression) planning 3

Which actions to regress over?

Relevant actions: have an effect which is in the set of goal elements and no effect which negates an element of the goal.

For example, $Buy(Jaguar)$ is a relevant action.

Search backwards from g, remembering the actions and checking whether we reached an expression applicable to the initial state.

Simple example of backward planning

```
Have(J)      Buy(J)      At(x)             Go(y,x)     At(y)
        <-----------     Have(Money) <----------      Have(Money)
                         Sells(x,J)                    Sells(x,J)

                         Does not match                Matches the
                         the initial                   initial state
                         state yet                     with y/Home and x/G
```

Slightly more elaborate example

if the goal is $Have(Jaguar) \land \neg At(Jail)$,

$g' = (\{Have(Jaguar), \neg At(Jail)\} - \{Have(Jaguar)\}) \cup$

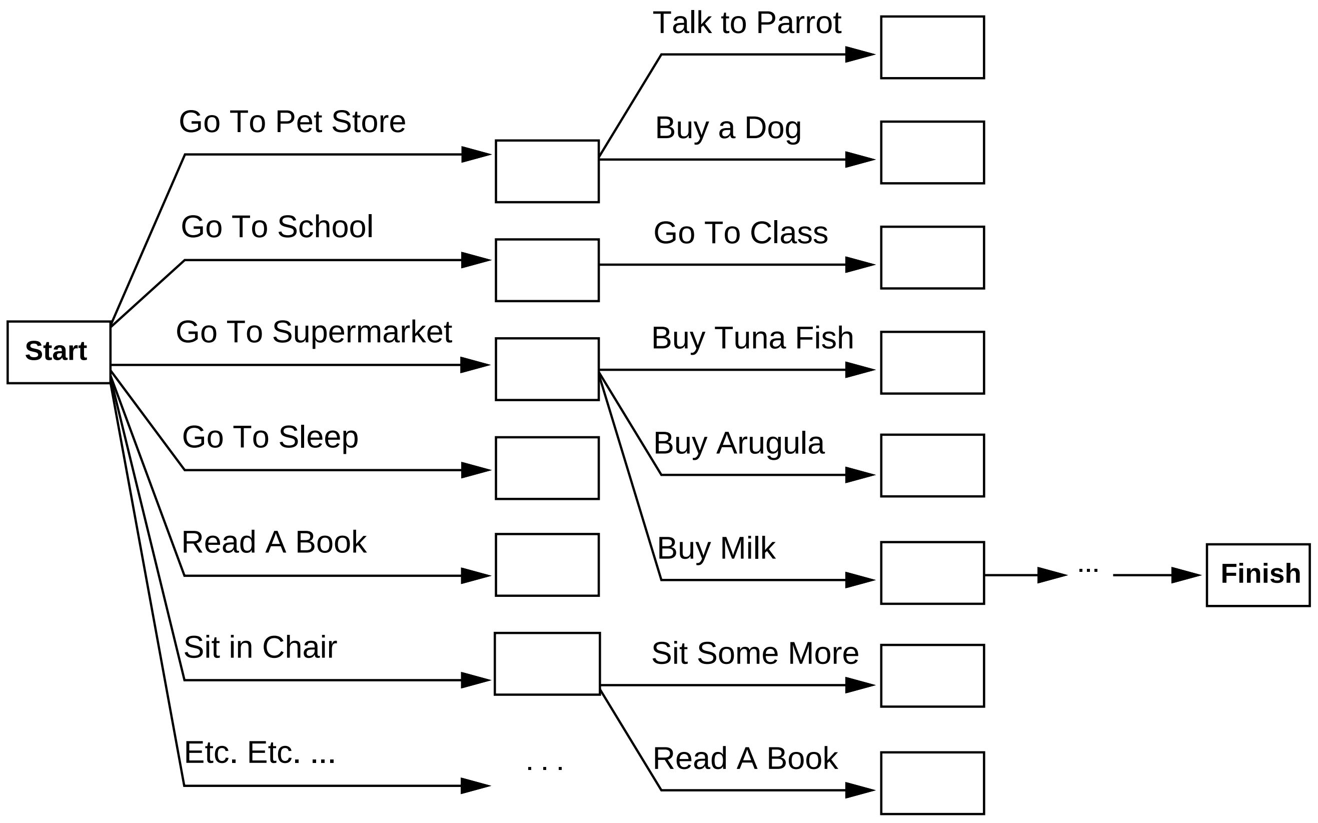
$\{At(p), Sells(p, Jaguar), Have(Money)\} =$

$\{\neg At(Jail), At(p), Sells(p, Jaguar), Have(Money)\}$

$Buy(Jaguar)$ is a relevant action. If we had an extra action $Steal(Jaguar)$ which also resulted in $Have(Jaguar)$ but had an additional effect of $At(Jail)$, it would not be a relevant action.

Comparison of forward and backward planning

If there are lots of actions, searching for a solution starting from the initial state looks hopeless



Comparison of forward and backward planning 2

However, it turns out we can automatically derive good heuristics (and remember how much better A^* is compared to uninformed search)

Two basic approaches:

1) add more edges to the graph (make more actions possible), and use solutions to the resulting problem as a heuristic. (There are often more efficient algorithms to solve the relaxed problem.)

Examples: remove (some) preconditions, ignore delete lists...

Action $Slide(t, s_1, s_2))$
Precond: $On(t, s_1) \land Tile(t) \land Blank(s_2) \land Adjacent(s_1, s_2)$
Effect: $On(t, s_2) \land Blank(s_1) \land \neg On(t, s_1) \land \neg Blank(s_2)$

removing $Blank(s_2)$ will enable tiles to move to occupied places: Manhattan distance heuristic

2) abstract the problem (make the search space smaller).

Comparison of forward and backward planning 3

Backward planning considers a lot fewer actions/relevant states than forward search, but uses sets of states (g, g') - hard to come up with good heuristics.

Use of logic and deduction

In situation calculus (lecture 8), planning **is** deduction

In ‘normal’ planning, we only need to check whether a state description entails some property: $s \models P(A, B) \wedge \neg Q(A)$ for example

In simple cases, like in this lecture, this just involves checking that $P(A, B)$ is in the list of properties s has, and $Q(A)$ is not (closed world assumption: if $Q(A)$ is not listed, then $\neg Q(A)$ must be true)

However, often planning domains are described using additional axioms, and then checking $s \models P(A, B)$ may involve more complex reasoning (whether $P(A, B)$ follows from the description of s and the axioms).

Next lecture

More classical planning

Goal-stack planning (based on another textbook: Elaine Rich and Kevin Knight, Artificial Intelligence).

Sussman anomaly