Introduction: What is planning?

Deciding what to, how and when

- We want to achieve something
- We have things we can do
- We know when we can do each thing
- We know what happens when we do things
- We know what we cannot or are not allowed to do
- We know where we are now
- So, what do we do, when and how to achieve what we want?
Example: Decide what a rover does

What can a rover do?

• Drive, turn, stop, etc.
• Operate sensors (cameras, etc.)
• Operate arm
• Operate internal systems (storage, etc.)
• Communicate with Earth and orbiters

What limits rover operations?

• Rules of the world – cannot be in two places at once
• Rules about operations – no moving arm while driving
• Temporal limits – moving takes time
• Resource limits – have only limited energy budget
• etc.
But, isn’t this easy?

We humans do it all the time, right?
- Yes, but not always well or correctly – and certainly not optimally
  - Example:
- Also, we are not very good at large problems
  - Example:

Not as easy as it looks!
- Humans often end up needing help with planning
- Machines often have hard time with planning

But, lots of fun!
Planning for rover operations
Rover planning on board
Planning

Ubiquitous in Artificial Intelligence

• Basic idea in rational agent is ability to achieve goals
• To achieve goals you invariably have to plan

Planning is in fact a search problem

• Given: Current state, possible actions and goals
• Actions: Map one state to another, if applicable
• Result: Sequence of actions to achieve goal
• Method: Search for a path, using actions as steps, to get from current state to goal
Planning sounds familiar, right?
  • Sliding tiles (8 puzzle) is a planning problem

Planning is key to AI
  • Rational agents, seeking to achieve goals, have to plan
  • Rational agents must work in many different areas

Special purpose search not reusable
  • Search for 8-puzzle solutions not good for controlling rover

General planning
  • Methods to solve arbitrary planning problems
  • Often built on general search methods, but there is more to it
  • Look at representation, reasoning and search for planning
Planning: Plan

Logical representation
  • Situation calculus

STRIPS representation
  • States and Actions
  • Examples

Simple search methods
  • Forward search
  • Backward search
  • Heuristic search
Planning: Outline of lectures

Logical representation
  • Situation calculus

STRIPS representation
  • States and Actions
  • Examples

Simple search methods
  • Forward search
  • Backward search
  • Heuristic search
Planning: Outline (cont)

Partial Order Planning
  • Search with grounded values
  • Search with variables
  • Heuristics

Planning with Graphs
  • Planning graphs
  • Heuristics
  • “Graphplan” methods
Planning

Planning in the “Real World”
  • Time, Resources, Complex Relations, Constraints, etc.
  • Hierarchical Task Network Planning
  • Planning in a non-deterministic world
  • Plan execution and re-planning

Decisions in an uncertain world
  • Basic notions in probability
  • Basic axioms and Bayes rule
  • Probabilistic methods for decision-making
Situation Calculus

Basic Idea: Use logic and theorem proving
  • Describe each situation with a set of logical sentences
  • Describe actions and effects with logical axioms

Planning
  • “Prove” the goal as a new sentence
  • Or, “ask” whether goal can be proven from given sentences
Logical representation

Time in planning

- Would assume **time** is key element in planning
- but, many planners use “steps” not actual time
- We will use steps for now - okay as long as actions are sequential

Situations (steps)

- Use basic sentences from predicate logic
- Situation typically defined by a set of propositions that are true:
  - isConnected(mac,epson)
  - isConnected(mac,laserjet)
  - canPrint(mac,laserjet)
  - isBroken(epson)
- Assume propositions not in set are false (closed-world assumption)
Situations changing over “time”

Situations change between steps
  • Need to connect step and situations

Want to describe different situations
  • Could say: afterStep(s) = { isLinked(mac,epson), isLinked(mac,laserjet),...}
  • Problem: Does not fit predicate logic

So, we describe a relation between s and literal:
  • holds(isLinked(mac,epson),s)
  • holds(isLinked(mac,laserjet,s)
  • ...

Describing actions

Basic idea is to describe what changes

• Assume $s$ is situation and $a$ is an action
• Use $\text{result}(a,s)$ to describe result of applying $a$ in $s$
• Example: $s' = \text{result}(\text{printFile}(\text{mac},\text{epson},\text{foo}), s_0)$

Example description:

• If printer is connected and not broken, and print command is given for a file, then the result is that file has been printed

$$\text{holds}((\text{isLinked}(\text{mac},\text{epson}),s) \land \neg \text{holds}((\text{isBroken}(\text{epson}),s)) \rightarrow \text{holds}((\text{havePrintout}(\text{foo}), \text{result}(\text{printFile}(\text{mac},\text{epson},\text{foo}), s)))$$
Lýsing einstakra vandamála

Upphafsstaða
- Staðan í upphafi áætlunar
- Gerum ráð fyrir að stöðu sé lýst til fullnustu
  - T.d. með forsendu um lokaðan heim (closed world assumption)

Lýsing á markmiði
- Skilyrði sem lýsa markmiði áætlunar
- Yfirleitt er stöðu ekki lýst til fullnustu
  - Margar stöður fullnægja skilyrðinu

Dæmi:
- Upphafsstaða:
  - holds(linked(mac,epson),s₀), holds(hasFile(mac,foo), s₀),...
- Markmiðskilyrði:
  - holds(havePrintout(foo),s₁₉₅₀)
Example

Actions

• $\neg \text{holds(have(fork),s)} \rightarrow \text{holds(have(fork), result grab(fork), s)}$
• $\neg \text{holds(have(knife),s)} \rightarrow \text{holds(have(knife), result grab(knife), s)}$

Initial state

• $\neg \text{holds(have(fork),s0)} \land \neg \text{holds(have(knife),s0)}$

Goal

• $\text{holds(have(fork),s)} \land \text{holds(have(knife),s)}$

Can now solve with any logical theorem prover

Or, can we?
Small Problem

Describing an action:

- $\text{holds(isLinked(mac,epson),s)} \land \neg\text{holds(isBroken(epson),s)} \rightarrow \text{holds(havePrintout(foo), result printFile(mac,epson,foo), s)}$

The Frame Problem:

- How do we know that the printer does not get disconnected?
  - Would rather not have to add every possible thing, such as:
    $\text{holds(isLinked(mac,epson), result printFile(mac,epson,foo), s)}$

- How do we know this does not turn on the projector?
  - Definitely don’t want to have to add:
    $\text{holds(isLinked(mac,epson), \neg\text{holds(isBroken(epson),s)} \land \neg\text{holds(isOn(projector),s)}} \rightarrow \text{holds(havePrintout(foo), result printFile(mac,epson,foo), s)}$

Related problems are: Qualification and Ramification Prob’s
Classical planning (STRIPS)

Simpler method

- Instead of general logic, use special “planning” representation
- Build on logic ideas – use propositional literals
- Define directly mapping of state and action to new state
- Solves the frame problem

Simplification also adds limits

- Less representational power
  - Cannot use complex logical relations
  - Cannot have conditional effects
  - Cannot extend to handle time, arithmetic, continuous resources,
  - Cannot do a lot of things
- But, sufficient to handle simple planning problems
STRIPS Actions

Each action has two parts:

• Condition on applicability (Preconditions)
• Description of result when it is applied (Effects)

Preconditions:

• Set of literals that must hold in current state
• print(file,mac,epson) has preconditions:
  • hasFile(mac,file)
  • linked(mac,epson)
  • -broken(epson)
  • hasPaper(epson)
• Common usage:
  • print(file,mac,epson):
    • pre: hasFile(mac,file), linked(mac,epson), -broken(epson), ...
STRIPS actions

Effects:

• Sets of positive and negated literals
• Positive ones are “Add effects”
• Negated ones are “Delete effects”
• Example: printFile(file,mac,epson) has the effects:
  • havePrintout(file)
  • -hasPaper(epson)
    • Silly printer can only fit one page at a time
• Common representation for effects
  • printFile(file,mac,epson):
    • add: havePrintout(file)
    • del: hasPaper(epson)
Example: Blocks World

The BlocksWorld problem

• Have blocks, e.g., A, B, C, D, …
• Have a table (infinite and without specific locations)
• Can move a block from atop a block to another
• Can move a block from table to atop of another block
• Can move block from atop of a block on to table

State descriptions:

• loc(x, y): Block x is on top of y (or on table if y is “table”)
• clear(x): No block is on top of x
Example: Blocks World

Action: move(x,y,z)
• pre: clear(x), loc(x,y), clear(z)
• add: loc(x,z), clear(y)
• del: loc(x,y), clear(z)

Action: move-to-table(x,y)
• pre: clear(x), loc(x,y)
• add: loc(x,table), clear(y)
• del: loc(x,y)

Action: move-from-table(x,y)
• pre: clear(x), loc(x,y)
• add: loc(x,table), clear(y)
• del: loc(x,y)
Classical state space planning

Initial state

- The state at the beginning
- Described as a set of positive literals
  - Assume other propositions are false

Goal condition

- Condition that defines the goal
- Set of literals (positive or negative)
  - Other propositions can be either true or false

Example:

- Initial state:
  - linked(mac,epson), hasFile(mac,foo), hasFile(mac,file),...
- Goal condition:
  - havePrintout(foo)
STRIPS properties

Solves the frame problem
- Literals that do not appear in effects are unchanged

Does not solve the ramification problem
- Hard to describe effects of complex actions
- Example: Airplane full of people flies to London – the action needs to describe effect on each person

Does not solve the qualification problem
- Difficult to exhaustively list all conditions
- Example: Car will start only if there is no potato in the tailpipe

Still, permits state space search for planning
Example: Transport problem

**Objects**
- Packages
- Airplanes
- Airports

**Actions**
- Load a package into airplane
- Fly airplane from one airport to another
- Unload package from airplane
Forward state space search

Basic idea:
- Search starts from initial state
- Applicable actions determine possible next states
- Next states defined by states and action effects
- Use any favorite search technique to find a goal state

Breadth-First Search:
- L is the initial state and an empty path
- Repeat:
  - If L is empty, we are done and no solution can be found
  - Pick first state s from list L
  - If s satisfies goal condition, return s with path to s
  - Find all applicable actions in s and for each action a, add the result of applying a to s to end of L (with path to s + a)
Example: Blind search in Blocks World

Initial state:
• \text{loc}(A,\text{table})
• \text{loc}(B,C)
• \text{loc}(C,\text{table})
• \text{clear}(A)
• \text{clear}(B)

Goal:
• \text{loc}(B,A)
• \text{loc}(A,C)
Blind forward state space search in planning

Unfocussed and often impractical method

- All possible actions examined
  - Even those having nothing to do with initial state or goal
- Search space is very large

However, heuristic version is getting better

- With recent advances in heuristics has forward search become a reasonable technique
Regression planning (“Backwards search”)

Basic Idea:

• Start with the goal condition
• Select an action that achieves some element in goal condition
• Find a state that permits action and gives goal condition
• Apply recursively to the state found
• Use search method of choice

Finding regression state of given state $S$:

• Select a literal in $S$, say it is $L$ (and $L$ is either $p$ or $-p$)
• Find action $A$ that achieves $L$
  • has $p$ in add effects if $L= p$
  • has $p$ in del effects if $L= -p$
• Regressed state is $(s \setminus L \cup \text{pre}(A))$
Example:

Initial state:
- \text{loc}(A,\text{table})
- \text{loc}(B,C)
- \text{loc}(C,\text{table})
- \text{clear}(A)
- \text{clear}(B)

Goal condition:
- \text{loc}(B,A)
- \text{loc}(A,C)
About regression planning

More focused than forward search
• Only examines actions that in some way relate to the goal

Blind regression is impractical
• Blind search is invariably exponential in length of plan

Heuristics more complex than for forward search
• But, some promising techniques based on heuristic regression
Heuristic state space search

Blind search does not work in planning

- Branching factor is way too large
- Without heuristics, even small planning problems are unsolvable

Basic ideas for heuristics in state space search for planning

- Simplify preconditions – ignore some of them
- Simplify effects – ignore some of them
Heuristics from ignoring preconditions

Extreme version: Ignore all preconditions

- Assume every action can be applied in any state
- Find a plan from given state to a goal state
  - Easier problem to solve, but additions and effects still make it nontrivial
  - Length of plan is heuristic evaluation of \( s \)

Variations

- Assume effects of actions are independent
  - Length of plan then becomes number of literals in goal different from \( s \)
- Assume actions of have no delete effects
  - Finding plan then becomes very easy
Heuristics from ignoring effects

Basic idea

• Assume actions have no delete effects
• Solve the simplified planning problem from state s to goal state
• Length of plan is heuristic evaluation of state s

Implementation

• Solving a planning problem to get heuristics
• But, problem is much simpler and easier to solve
Separating subgoals

Basic idea

• Find a plan for each literal in goal condition
• Combine the plans to generate a complete plan

Problem!

• Initial state: loc(B,table), loc(A,table), loc(C,A), clear(B), clear(C)
• Goal: loc(A,B), loc(B,C)
• Called the Sussman anamoly