Research Area

• Methods that help make software autonomous, or help us make decisions, and their application to robotics
Focus of This Lecture

- Classical Planning
  - Example: manipulation planning (possible thesis topic)
- Planning under Uncertainty
- Multi-Agent Planning under Uncertainty (possible thesis topic)
  - Offline planning
  - Online planning
Applications of Planning

- System that can figure out how to achieve a goal
  - Autonomous systems (robots, game-playing agents, software agents)
  - Automated composition of web-services (e.g., planning a vacation)
  - Solve complicated planning problems (e.g., logistics, construction)

- Decision-support systems and decision analysis
  - Making decisions under uncertainty and complex objectives:
    - Strategic decisions such as whether to invest in some technology, where to build a new plant, etc.
Video
Classical Planning

- Our world is like a (very big) automaton
- Our actions change the world
  - Action = letter; All possible actions = alphabet
- A plan takes us from the current state to a goal state
- Current state = initial state
- Goal states = accepting states
- Plan = a word in the language of this automaton
  - = a word that takes us from the initial state to the goal state
Blocks world
The rules of the game

- Location on the table does not matter
  ![Diagram](image1)

- Location on a block does not matter
  ![Diagram](image2)

- At most one block on/under a block is allowed
  ![Diagram](image3)
Blocks world
The transition graph for three blocks
Example Applications

- Planning a sequence of operations to transform raw material to an end product
  - Metal bending machines use this
- Finding a sequence of actions for the robot that will result in a sample from an interesting rock
  - NASA’s Mars Rovers use such technology
- Find a sequence of actions to repair the current state
  - Xerox copiers use this
- Program a system for simulating a smart opponent
  - Elbit uses this
- Penetration testing -- find a sequence of actions that a malicious user could use to access your system
Trivial Solution

- Compute shortest path from initial state to a goal state using Dijkstra’s algorithm
  - Complexity $O(|v| \log |v| + |E|)$
Finding a solution is polynomial time in the number of blocks (move everything onto the table and then construct the goal configuration)

Finding a shortest solution is NP-complete (for a compact description of the problem).
Classical Planning

- Problem: world is too big and complicated
  - Too many states!
  - Can’t really write down the whole automaton

- Solution:
  - Need a compact (implicit) specification
    - Called a planning domain description language
    - More natural to use than an explicit model
  - find efficient algorithms that operate on this specification
    - Called planning algorithms
    - Generate a plan (= word)
    - Efficient as a function of the input, not the automaton
The STRIPS Language

- Stanford Research Institute Problem Solver – an automated planner. (Fikes & Nilsson 1975)
- The name was later used to refer to a formal language for describing its inputs.
- This language is the basis for most of the languages for expressing automated planning problems.
Back to Blocks’ World

Initial State

Goal State
Model

Blocks world
The transition graph for three blocks

AI Planning
Transition systems
Definition
Example
Matrices
Reachability
Algorithm
Succinct TS
STRIPS - States

- State – An assignment of truth values (true/false) to a set of state variables.
  - On(X,Y), where X,Y are blocks or the Table
  - Clear(X), where X is a block

- State representation in STRIPS: list of all variables whose truth value is true
  - On(A,Table), On(B,C), On(C,Table), Clear(A), Clear(B)
  - Recall from logic:
    - on(A,Table) is a (propositional) variable
    - A literal is a variable or its negation (on(A,B), -on(A,B), …)
STRIPS - Actions

- Operators - described by a schema of variables composed of 3 lists:
  - Preconditions.
  - Delete-list.
  - Add-list.

- Actions obtained by replacing operator variables by state variables (called grounding)

- Move(X,Y,Z) as STRIPS Action:
  - Precondition: On(X,Y), Clear(X), Clear(Z)
  - Add-List: On(X,Z), Clear(Y)
  - Delete-List: On(X,Y), Clear(Z)
STRIPS – Action Semantics

- If a State does not satisfy Preconditions, Action is not defined.
- Else, the result of the Action is obtained as follows:
  New-State=(State/Delete-List) U Add-List
Example: Apply Move(B,C,A)

- Move(B,C,A) as STRIPS Action:
  - Precondition: On(B,C), Clear(B), Clear(A)
  - Add-List: On(B,A), Clear(C)
  - Delete-List: On(B,C), Clear(A)
Example:

- What if we try to apply the same action in our new state?
- In what other states can we apply this action?
Goal Representation

- Goals are represented by a conjunction (AND) of positive literals
- For example On(A,B) & On(B,C)
- A goal state is any state that satisfies this conjunction
- More than one state could be a goal state
The Planning Problem

- $P = <V,A,I,G>$
- $V$ - set of variables (not necessarily Boolean)
- $A$ - set of (deterministic) actions compactly described
- $I$ - initial state
- $G$ - goal condition (some condition on $V$)
Our Focus: Plan Generation

- Find a sequence of actions that will get us from the initial state to a goal state
- Complexity: PSPACE-Complete
- Find a short (bounded) sequence of actions (if one exists) that ...
- Complexity: NP-complete
Example Research Problem

- Manipulation planning with fixed manipulation primitives
FORWARD SEARCH PLANNING

- We build a search tree
  - Root = initial state
  - Children of a node = all states reachable by applying a single action to the parent
- Leaf nodes
  - Goal states
  - Dead end states
1st Solution: Forward Search

Remember:

- We can search through the space in many ways.
- We would wish to find a good heuristic function and a good search method.

Initial State

{on-a-b,on-b-c,clear-a,on-table-c}

Goal Conditions

{on-table-a,on-table-b, on-table-c,clear-a,clear-b,clear-c}

Move-to-table-a-b

{on-table-a,on-b-c,clear-a,clear-b,on-table-c}

Move-from-table-a-b

{on-table-a, on-table-b, on-table-c,clear-a,clear-b,clear-c}

Move-to-table-b-c

Move-b-c-a

{on-a-b,on-b-c,clear-a,on-table-c}

{on-table-a,on-b-a,on-table-c,clear-b,clear-c}
How Do We Construct the Tree

- The order in which we add nodes to the tree is crucial
- There are many different options
Blind Search

- Main algorithms:
  - **DFS** – Depth First Search
    - Expand the deepest unexpanded node.
    - Fringe is a LIFO.
  - **BFS** – Breath First Search
    - Expand the shallowest unexpanded node.
    - Fringe is a FIFO.
  - **IDS** – Iterative Deepening Search
    - Combines the advantages of both methods.
    - Avoids the disadvantages of each method.

- There are many other variants (e.g., optimizing disk access, parallel search, etc.)
Breadth First Search
Depth First Search
Iterative Deepening Search
Heuristic Search

- Blind search is hopeless in large problems
- However, if we have some way of estimating how far we are from the goal, we can use heuristic-guided search
Informed/Heuristic Search

- **Heuristic Function** $h : S \rightarrow R$
  - For every state $s$, $h(s)$ is an estimation of the minimal distance/cost from $s$ to a solution.
    - Distance is only one way to set a price.
    - How to produce $h$? later on...

- **Cost Function** $g : S \rightarrow R$
  - For every state $s$, $g(s)$ is the minimal cost to $s$ from the initial state.

- $f = g + h$, is an estimation of the cost from the initial state to a solution.
Best First Search

- Greedy on h values.
- Fringe stored in a queue ordered by h values.
- In every step, expand the “best” node so far, i.e., the one with the best h value.
直線距離到布加勒斯特

- 奧拉德 (Oradea)：366
- 布加勒斯特 (Bucharest)：0
- 克拉約瓦 (Craiova)：160
- 多比瑞塔 (Dobreta)：242
- 埃福裡 (Eforie)：161
- 法格拉斯 (Fagaras)：176
- 吉吉古伊 (Giurgiu)：77
- 希斯索瓦 (Hirsova)：151
- 基斯 (Iasi)：226
- 魯戈伊 (Lugoj)：244
- 喬拉迪亞 (Mehadia)：241
- 尼姆特 (Neamt)：234
- 奧拉德 (Oradea)：380
- 尼斯泰 (Pitesti)：10
- 布加勒斯特 (Bucharest)：85
- 尼姆特 (Neamt)：98
- 希斯索瓦 (Hirsova)：92
- 基斯 (Iasi)：87
- 尼米奇維爾查 (Rimnicu Vîlcea)：99
- 布加勒斯特 (Bucharest)：90
- 摩羅蓋亞 (Mehadia)：138
- 尼米奇維爾查 (Rimnicu Vîlcea)：146
- 聖伊戈伊 (Sibiu)：140
- 佐裡尼 (Zerind)：151
- 佐裡尼 (Zerind)：75
- 佐裡尼 (Zerind)：71
- 佐裡尼 (Zerind)：118
- 羅伊 (Roia)：70
A* Search

- Idea: Avoid expanding paths that are already expensive.
- Greedy on f values.
- Fringe is stored in a queue ordered by f values.
- Recall, \( f(n) = g(n) + h(n) \), where:
  - \( g(n) \): cost so far to reach \( n \).
  - \( h(n) \): estimated cost from \( n \) to goal.
  - \( f(n) \): estimated total cost of path through \( n \) to goal.
Reaching Bucharest with BFS
Generating a Heuristic Function
Backward Searching

- Reverse the search:
  - Initial State: list of goal variables.
  - Actions: reversed operators.
  - Goal State: list of variables in the initialization.

- Problems:
  - How to describe all the goal states?
  - If we start from a set of states, we need to maintain them along the way.
  - How to reverse an operator?
Backward Search

- **How to describe all the goal states?**
  Every state that gives TRUE value to all goal conditions will be in the goal states. E.g. \{on-C-A\}

- **How to reverse an operator?**
  Lets look at an example:
  apply move-C-B-A in a state which satisfies \{on-c-a, clear-d\}.
  - What must a state satisfy so that applying move-C-B-A in it will result in a state which satisfies \{on-c-a, clear-d\}?  
    - All preconditions of move-C-B-A.
    - clear-d, as it is not in the add-list of move-C-B-A.
    - on-C-A is not required, as it is in the add-list of move-C-B-A.
  - What must a state satisfy so that applying move-C-B-A in it, will result in a state which satisfies \{clear-a\}?  
    - No such state exists, since \{clear-a\} is in move-C-B-A’s delete-list.
Regression

- Reversing an operator is called "Regression".
- Formally: \( \text{Regress}(\text{condition},\text{action}) \) is the weakest condition \( c \) such that applying \( a \) in a state satisfying \( c \) will result in a state satisfying \( \text{condition} \).
- For example:
  \[
  \text{Regress}\{\text{on}(c,a),\text{clear}(d)\}, \text{move}(c,b,a) = \{\text{on}(c,b),\text{clear}(a),\text{clear}(c),\text{clear}(d)\}
  \]
Regression

More Formally:

\[
\text{Regress}(\text{cond}, \text{action}) = \begin{cases} 
\text{preconditions}(\text{action}) \cup (\text{cond} \setminus \text{add-list}(\text{action})) & \text{if } \text{cond} \cap \text{del-list}(\text{action}) = \emptyset \\
\text{Regress}(\text{cond}, \text{action}) = \text{false} & \text{otherwise}
\end{cases}
\]

Note:
Regress(condition, action) is defined even if cond \cap add-list(action) = {}, but looking at such actions is pointless…
Backward Search Example

- Initial State: \{on-A-B, on-B-C, on-table-C, clear-A\}
- Goal: \{on-c-a\}

Diagram:

- Initial State
- Goal Conditions
- Move-c-b-a
- move-from-table-c-b
- move-c-a-b
- move-from-table-c-a
- move-to-table-b-c
- move-to-table-a-b
- \{on-c-a, clear-c, clear-b\}
- \{on-table-c, clear-a, on-b-c, clear-b\}
- \{on-table-c, clear-a, on-b-c, clear-a\}
- \{on-table-c, clear-a, clear-c\}
- \{on-c-b, clear-a, clear-c\}
- \{on-c-a\}
Uncertainty

- Two possible sources of uncertainty
  - Unknown initial state
  - Non-deterministic actions
- We will focus on the first
- We represent uncertainty by modeling the possible states of the world (possible worlds)
  - If we don’t know whether the car is in Beer-Sheva or Tel-Aviv, we consider two possible states
- The agent’s state of information is represented by a set of possible worlds
Example:

GoSouth  GoEast  GoEast  GoEast  GoSouth  GoNorth
Remark:

- the search space is $Pow(S)$
- $S$ contains 15 states,
- $Pow(S)$ contains 32767 belief states!
Searching in Belief Space

- Belief space is huge
- But we can represent our uncertainty symbolically. For example, we simply list those values that we know to hold
  - Agent knows car in Beer-Sheva: at(car,BS)
  - Agent doesn’t know location of car: {}
- Efficiently representing and searching belief space is difficult
- Notice that if we don’t know a precondition of an action, we cannot apply it!
Observations/Sensing

- Typically, our agents can learn about the world through sensing
  - A robot can apply its sensor to find its position
  - We can call the driver to find car’s location

- We can model this using an action with the effect of adding \( \text{at(car,BS) v at(car,TLV)} \)

- If we have observations, plans are no longer words — they are decision trees
Multi-Agent Planning Under Uncertainty (Possible Thesis Topic)

- Suppose we have a team of agents that wants to cooperatively achieve a goal
  - Example: rescue survivors of an earthquake
- They will be in different places and will make different observations, so their beliefs will be different
- Planning for them is challenging
Offline Planning

- We plan for all agents before we start acting
- Initially, all agents have the same set of possible worlds
- As they perform observations, their belief states will become different
  - Unless they can broadcast these observations
- The plan must take this into account:
  - An agent cannot be asked to do different things in situations it cannot distinguish between
QDec-POMDP Policies

Why do we have to no-op here?

Agent 1 must no-op because it cannot distinguish between these two histories
Agent 2 must no-op because it cannot distinguish between these two histories
QDec-POMDP Policies

Agent 1

Done

Push-up

Done

Push-up

Obs-box

Right

Push-up

No-op

Right

Obs-box

Agent 2

Done

Push-up

Done

Push-up

Obs-box

Left

Push-up

Left

Obs-box

No-op

Waiting for the other agent
Online Planning (hard!)

- Building a complete offline plan that considers all possibilities is difficult.
- In online planning, we plan for just a few possibilities, but may need to revise the plan as we learn unexpected information.
- How can an agent revise its plan when it does not know what the other agents have observed, and therefore, what they will do?