Automated Planning and Decision Making
Prof. Ronen Brafman

Introduction to Classical Planning
Background

- Classical Planning is the first model developed in AI
- It was developed in order to compute good behaviors for robots
- It makes a lot of strong assumptions about the system
- Yet, it was used for important applications
- And the techniques developed for it were extended to handle more complex models
In comparison to MDPs:

- Actions are deterministic: that is, $tr(s, a, s') = 0$ or $tr(s, a, s') = 1$
- Instead of rewards, we talk about goals – or goal states.
  - You can think of goal states as state with large positive reward from which you cannot escape (sink states)
  - Cost is associated with every action, not the state
- Important difference: states have structure that we exploit:
  - states are vectors = assignments to some set of states variables
  - actions are defined by describing how they change the values of variables
  - typical actions change a small number of variables
**Blocks world**
The rules of the game

**Location on the table does not matter**

![Diagram of blocks on the table]

**Location on a block does not matter**

![Diagram of blocks on a block]

**At most one block on/under a block is allowed**

![Diagram prohibiting multiple blocks on/under another block]
Blocks world
The transition graph for three blocks

replace picture with state description
Deterministic Planning Models are Automata

- Our world is like an automaton
- Our actions change the world = letters in our 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
  - Solution: trivial – find shortest path from initial state to a goal state using Dijkstra’s algorithm
  - Complexity $O(|v| \log |v| + |E|)$
  - Good bye, and see you at the exam!?
# Blocks World

## Properties

<table>
<thead>
<tr>
<th>blocks</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>73</td>
</tr>
<tr>
<td>5</td>
<td>501</td>
</tr>
<tr>
<td>6</td>
<td>4051</td>
</tr>
<tr>
<td>7</td>
<td>37633</td>
</tr>
<tr>
<td>8</td>
<td>394353</td>
</tr>
<tr>
<td>9</td>
<td>4596553</td>
</tr>
<tr>
<td>10</td>
<td>58941091</td>
</tr>
</tbody>
</table>

1. Finding a solution is polynomial time in the number of blocks (move everything onto the table and then construct the goal configuration).
2. Finding a shortest solution is NP-complete (for a compact description of the problem).
Some Problems

- Representation: Transition function requires $O(|S|^2)$ space
  - Most entries are 0
- Representation: Reward function requires $O(|S|)$ space
- Representation: $|S|$ is exponentially in the number of blocks
- Planning: Finding shortest path to a goal state is easy?
  - Shortest path is low-order polynomial
  - ... in the number of nodes (= states)
- How can anyone specify a problem explicitly or solve it?
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
  ▶ Need efficient algorithms that operate on this specification
    ▶ Efficient as a function of the input, not the automaton
    ▶ Called planning algorithms
    ▶ Generate a plan (= word = path) without building the whole graph
Key Concepts

- **Model**: Describes those aspects of the world we want to capture.
  - Should be intuitive
  - Can be very large and difficult to work with directly
  - Our model: an automaton where letter = action

- **Language**: Tool for describing a model implicitly
  - We don’t want to explicitly list every state and the transition function
  - We use a language for describing them implicitly

- **Interpretation**: Maps between expressions in the language and the model
  - Something that tells us how to build the automation from a planning domain description

- **Queries**: Questions about the model we wish to answer
  - Classical planning: find a plan (=word) that takes you from the initial state to a goal state

- **Algorithms**: Compute answer to the queries
Classical Planning Problem Definition

Given a system with a set of deterministic operators/actions, an initial system state and a set of goal states, find a sequence of actions that transform the system from its initial state to a state satisfying the goal.
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
Back to Blocks World

Initial State

Goal State
Model & Language

- **Model**: Describes the problem domain (the automaton we saw earlier)
  - **S** - Set of possible states of the world/system
    - Possible block configurations
  - **A** - Set of deterministic actions. $A : S \rightarrow S$
    - Take the form Move($X,Y,Z$)
    - Move($B,C,Table$):

- **Language**: Describes a model.
  - How to represent a state?
  - How to represent an action?
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.
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 - Actions (alternative)

- Operators - described by a schema of variables composed of 2 lists:
  - Preconditions.
  - Effects

- 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)
  - Effects: On(X,Z), Clear(Y), ¬On(X,Y), ¬Clear(Z)
STRIPS  Action Semantics

- If State does not satisfy Preconditions, Action is not defined.
- Else, the result of the Action is obtained as follows:
  \[ \text{NewState} = (\text{State}/\text{DeleteList}) \cup \text{AddList} \]
- Note: Apply delete operation first, because it is possible that:
  \[ \text{AddList} \cap \text{DeleteList} \neq {} \]
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?
Goals are represented by a conjunction (AND) of positive literals

For example $\text{On}(A,B) \land \text{On}(B,C)$

A goal state is any state that satisfies this conjunction

More than one state could be a goal state
The Planning Problem

- $P = \langle V, A, I, G \rangle$
- $V$ - set of variables (not necessarily Boolean)
  - State space size: $\max\{\text{domain size}\} |V|$
  - STRIPS: boolean variables (propositions)
- $A$ - set of (deterministic) actions
  - STRIPS is just one way to describe actions
  - Many alternatives and extensions
- $I$ - initial state
  - STRIPS: list of propositions assigned true
- $G$ - goal condition
  - STRIPS: a conjunction of propositions
  - Could be an arbitrary formula over $V$
The Planning Problem

- $P = \langle V, A, I, G, c \rangle$
- $V$ - set of variables (not necessarily Boolean)
  - State space size: $\max\{\text{domain size}\}|V|$
  - STRIPS: boolean variables (propositions)
- $A$ - set of (deterministic) actions
  - STRIPS is just one way to describe actions
  - Many alternatives and extensions
- $I$ - initial state
  - STRIPS: list of propositions assigned true
- $G$ - goal condition
  - STRIPS: a conjunction of propositions
  - Could be an arbitrary formula over $V$
- $c : A \rightarrow R$
  - The cost of each action
Richer Languages:
Example – Transport Domain

(define (domain transport)
  (:requirements :typing :action-costs)
  (:types
    location target locatable - object
    vehicle package - locatable
    capacity-number - object )
  (:predicates
    (road ?l1 ?l2 - location)
    (at ?x - locatable ?v - location)
    (in ?x - package ?v - vehicle)
    (capacity ?v - vehicle ?s1 - capacity-number)
    (capacity-predecessor ?s1 ?s2 - capacity-number))
  (:functions
    (road-length ?l1 ?l2 - location) - number
    (total-cost) - number
  )
)
(:action drive)
  :parameters (?v - vehicle ?l1 ?l2 - location)
  :precondition (and
    (at ?v ?l1)
    (road ?l1 ?l2))
  :effect (and
    (not (at ?v ?l1))
    (at ?v ?l2)
    (increase (total-cost) (road-length ?l1 ?l2))
  )

(:action pick-up)
  :parameters (?v - vehicle ?l - location ?p - package ?s1 ?s2 - capacity-number)
  :precondition (and
    (at ?v ?l)
    (at ?p ?l)
    (capacity-predecessor ?s1 ?s2)
    (capacity ?v ?s2))
  :effect (and
    (not (at ?p ?l))
    (in ?p ?v)
    (capacity ?v ?s1)
    (not (capacity ?v ?s2))
    (increase (total-cost) 1))

(:action drop)
  :parameters (?v - vehicle ?l - location ?p - package ?s1 ?s2 - capacity-number)
  :precondition (and
    (at ?v ?l)
    (in ?p ?v)
    (capacity-predecessor ?s1 ?s2)
    (capacity ?v ?s1))
  :effect (and
    (not (in ?p ?v))
    (at ?p ?l)
    (capacity ?v ?s2)
    (not (capacity ?v ?s1))
    (increase (total-cost) 1)))
(define (problem transport-city)
 (:domain transport)
 (:objects
  city-loc-1 - location
  city-loc-2 - location
  city-loc-3 - location
  truck-1 - vehicle
  truck-2 - vehicle
  package-1 - package
  package-2 - package
  capacity-0 - capacity-number
  capacity-1 - capacity-number
  capacity-2 - capacity-number
  capacity-3 - capacity-number
  capacity-4 - capacity-number
 )
 (:init
 (= (total-cost) 0)
 (capacity-predecessor capacity-0 capacity-1)
 (capacity-predecessor capacity-1 capacity-2)
 (capacity-predecessor capacity-2 capacity-3)
 (capacity-predecessor capacity-3 capacity-4)
 (road city-loc-3 city-loc-1)
 (= (road-length city-loc-3 city-loc-1) 22)
 (road city-loc-1 city-loc-3)
 (= (road-length city-loc-1 city-loc-3) 22)
 (road city-loc-3 city-loc-2)
 (= (road-length city-loc-3 city-loc-2) 50)
 (road city-loc-2 city-loc-3)
 (= (road-length city-loc-2 city-loc-3) 50)
 (at package-1 city-loc-3)
 (at package-2 city-loc-3)
 (at truck-1 city-loc-3)
 (capacity truck-1 capacity-4)
 (at truck-2 city-loc-1)
 (capacity truck-2 capacity-3)
 )
 (:goal (and
 (at package-1 city-loc-2)
 (at package-2 city-loc-2)
 ))
 (:metric minimize (total-cost))
)
Questions We Can Ask

- Is there a plan?
- Find a plan?
- Find an optimal plan?
  - Optimal can mean different thing
    - Fewest actions
    - Fewest time-steps with parallel actions
    - Minimal sum of action cost
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
Original STRIPS Algorithm

- Maintain two data structures:
  - Current state.
  - Stack with sub-goals (i.e, state variables to achieve)
- Initialization:
  - Push all goal variables to the stack in some order.
  - Current state ← Initial state.
Current State = Initial State;
While ( stack is not empty ){

    X ← pop ( stack );

    Case:
    X is an action:
        Apply X to the current state; update state; record action
    X is a literal not satisfied in the current state:
        { Push an action that has X in its add list
        Push the preconditions of this action in some random order
        }
    X is a literal not satisfied in the current state:
        Pop X;
}

Output the list of executed actions
Problem Example

Recall the Blocks World:
- **Initial state:**
  - on(C,T), on(B,A), on(A,T), Clear(B), Clear(C)
- **Goal state:**
  - on(A,B), on(B,C), on(C,T)

In STRIPS:
- **Initial state:**
  - on(C,T), on(B,A), on(A,T), Clear(B), Clear(C)
- **Goal state:**
  - on(A,B), on(B,C), on(C,T)
STRIPS Algorithm Example

- Initial state:  
  - B
  - A
  - C

- Goal state:  
  - A
  - B
  - C

Goal state variables:

<table>
<thead>
<tr>
<th>Action</th>
<th>Pre-conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>On(A,B)</td>
<td></td>
</tr>
<tr>
<td>On(B,C)</td>
<td></td>
</tr>
<tr>
<td>On(C,T)</td>
<td></td>
</tr>
</tbody>
</table>

- Action rules:
  - We pop On(A,B), add the action Move(A,T,B), which has On(A,B) in its add-list, and push its pre-conditions to the stack.
  - We pop Clear(A), add the action Move(B,A,C), which has Clear(A) in its add-list, and push its pre-conditions to the stack.
STRIPS Algorithm Example

We pop these variables and because they are true no action follows.

We activate these operations on the current state.
Example: A Problem STRIPS Will not Solve:

- Initial state: (Diagram showing a green 'C' on top of a red 'A', followed by a blue 'B'.)
- Goal state: (Diagram showing a red 'A' on top of a blue 'B', followed by a green 'C'.)
Observations:

- The algorithm can get stuck when there is a precondition that no action can generate
  - Does this mean that there is no plan?
- The algorithm is not sound or complete
  - Sound = returns only valid plans
  - Complete = always returns a plan when one exists
  - Terminating = the algorithm always terminates in finite time
Improved STRIPS Algorithm

- Given a goal or a list of preconditions
  - We push the entire (conjunctive) goal or precondition on the stack
- When we pop a conjunctive precondition we push it back on stack, and above it we push the literals in it in some random order
- The new algorithm:
  - Initialization: Goal is pushed to the stack in conjunctive form.
  - Current state ← Initial state.
Improved STRIPS Algorithm

Push goal conjunct into stack;
Current state = Initial state;
While ( stack is not empty ){

X ← pop ( stack );

Case:
X is an action:{
    Apply X to the current state and update the state;
    Record the action
}
X is a conjunction of literals not satisfied in the current state:{
    Push X back;
    Push elements of X in some random order;
}
X is a literal not satisfied in the current state:{
    Push an action that has X in its add list
    Push the preconditions of this action
}
}

Output the list of execution actions
Algorithm Properties

Improved algorithm is:

- Sound
- Incomplete
- May not terminate