Automated Planning and Decision Making
Prof. Ronen Brafman

Classical Planning Problems and Planning as Search
Planning Model - A Review

▶ Set of possible states of the world.
  ▶ Every state is described as an assignment to a set of state variables. In the blocks world, for example, we can use either:
    ▶ Variables for each block, telling what it is on.
    ▶ Boolean variables, telling for each pair of blocks A and B, whether A is on B.

▶ Set of operators/actions.
  ▶ An operator maps each state to the state obtained following its application.

▶ Initial State.

▶ Goal State(s).
Basic Assumption of the Model:

▶ Atomic time – action execution is indivisible and uninterruptible
▶ Deterministic effects
▶ Omniscience: complete knowledge of world state, initial state, and dynamics
▶ Agent is sole cause of change

Removing any of these leads to different extension of the model
Reminder: Language – STRIPS

- All variables are Boolean.
- State: list of TRUE valued variables.
- Action (operator):
  - Preconditions.
  - Add list.
  - Delete list.

Meaning:
- An action \( a \) is applicable in state \( s \), only if the preconditions of \( a \) are true in \( s \). Otherwise, it is undefined or has no effect.
- The result is a new state, \( s' \), received by removing the members of the delete-list from \( s \) and adding those in the add-list.

- Goal: a list (conjunction) of variables. Any state that satisfies all of them is a goal state.
Blocks world, with 3 blocks: A, B and C.
Initially, A is on B, and B is on C.
Goal is to get B on A.
Language:
  on(X Y), on-table(X), clear(X)
  Note: not every assignment to these variables describes a possible state. E.g. (on A B)(on A C)
STRIPS Example

- Actions: of the form \( \text{move}(X,Y,Z) \), meaning move \( X \) from \( Y \) to \( Z \).
- We can define the above by a few schemas of operators with variables:
  - \( \text{Move}(x,y,z) \)
    - Preconditions \{ on(x,y), clear(x), clear(z) \}
    - Delete list \{ on(x,y), clear(z) \}
    - Add list \{ on(x,z), clear(y) \}
  - \( \text{Move-to-table}(x,y) \)
    - Preconditions \{ on(x,y), clear(x) \}
    - Delete list \{ on(x,y) \}
    - Add list \{ on-table(x), clear(y) \}
  - \( \text{Move-from-table}(x,y) \)
    - Preconditions \{ on-table(x), clear(y) \}
    - Delete list \{ on-table(x), clear(y) \}
    - Add list \{ on(x,y) \}

- We will usually use the term "action" to describe a concrete operation such as \( \text{move}(A,B,C) \). We use "operator" or "schema", to describe an operation with variables such as \( \text{move}(x,y,z) \)
Other Languages

- Strips extension:
  - Conditional Actions: Moving a suitcase. Conditional effect: the contents of the suitcase move as well
  - Quantifiers: Move a suitcase. Every item inside is moved, too.

- SAS+:
  - Like STRIPS, but instead of boolean variables, general variables with a finite domain are used.
    - Example: in blocks’ world, we can use the variable on(x) whose value is the block x is on. Such as on(A)=B.
    - We also use clear(x), which is boolean (special case of multi-valued).
  - Distinguishes between:
    - Prevail conditions: Preconditions not changed by the action
      - Prevail of Move(A,B,C) is clear(A)
    - Preconditions: Preconditions that are changed by the action
      - Precondition of Move(A,B,C): On(A)=B, clear(C)
  - Effect notation rather than Add+Delete is typically used

- PDDL: Many versions that include extension to temporal actions, preferences, and much more
Complexity of STRIPS Planning

PlanSAT:

- Given a Strips problem $\langle P, A, I, G \rangle$ is the problem solvable?
- Complexity: PlanSAT is PSPACE-complete
  - This means that a TM would need a work-tape that is polynomial in the size of the input for its computation
  - Which as best as we know implies the computation can take exponential time (we know $\text{NP} \subseteq \text{PSPACE}$)
- Example of a problem with exponentially long plans:
  - $n$-Rings Towers of Hanoi

Bounded-PlanSAT:

- Given a Strips problem $\langle P, A, I, G \rangle$ and an integer $b$ (in unary representation), is there a plan with at most $b$ actions?
- Complexity: PlanSAT is NP-complete
Complexity in Concrete Domains

- Within concrete domains, BoundedPlanSAT is often harder than PlanSAT
  - In many benchmarks, Bounded-PlanSAT is NP-complete while PlanSAT is in P
  - Why is PlanSAT for Blocksworld in P?
- Examples: Blocksworld, Logistics
- Informally: optimal planning is almost never easy
Two Examples of Planning as Search
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
Backward Search

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

- 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 do we describe sets of states
  - How to reverse an operator?
Backward Search

- How to describe all the goal states?
  As usual. We just need to understand that variables not in the list are not assumed to be false.

- How to describe sets of states?
  If we are lucky we can use the same idea as goal states.

- How to reverse an operator?
  Let’s 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 the delete-list of move-C-B-A.
Reversing an operator is called "Regression".

Formally: Regress(condition, action) is the weakest condition $c$ such that applying $a$ in a state satisfying $c$ will result in a state satisfying condition.

For example:
\[
\text{Regress}\left(\{\text{on}(c,a), \text{clear}(d)\}\right), \text{move}(c,b,a) = \{\text{on}(c,b), \text{clear}(a), \text{clear}(c), \text{clear}(d)\}
\]
Regression

More Formally:

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

Note: \text{Regress}(\text{condition,action})\) is defined even if \text{cond} \cap \text{add-list}(\text{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\}
We defined two different search spaces:
- Different starting states
- Different operators
- Different termination criteria

Search space definition is independent of the search method
- We can search each of these spaces using different search methods
Important Point!

When solving a real-world problem using search, we have two decisions to make

1. What is the search space
   - The search space is the entire search tree or search graph
   - Need to specify: root node, search operator, leaf nodes (no children) and goal nodes

2. In what order will we visit the search nodes
   - Both decisions are important
   - Most of the algorithms we describe care only about the first question
   - We can then use our favorite search algorithm within them
   - Most work these days focuses on methods for making forward search efficient
   - This may change in the future
Algorithm ProgWS(world-state, goal-list, A, path)
1. If world-state satisfies each conjunct in goal-list
2. Then return path (termination)
3. Else let Act = choose from A an action whose precondition is satisfied in world-state (children)
   3.1 If no such choice was possible
   3.2 Then return failure (leaf node)
   3.3 Else let S = result of applying Act in world-state and return
      ProgWS(S, goal-list, A, concat(path, Act))

Also known as non-deterministic formulation of the algorithm
Search Space for BWD Search

Algorithm RegWS(init-state, cur-goal, A, path)

1. If init-state satisfies each conjunct in cur-goals
2. Then return path
3. Else do:
   3.1 let Act = choose from A an action whose effect matches at least one conjunct in cur-goals:
   3.2 Let G = the regres(cur-goals, Act)
   3.3 If G = false return failure
   3.4 Else return RegWS(init-state, G, A, concat(Act, path))
Making Search Efficient
Why is Search Hard?

- Even if we concentrate on short plans, their number is very large
  - Exponential in plan length
Main Techniques

- Pruning techniques
  - Recognizing irrelevant/unuseful actions
- Generating good Heuristic functions
  - More general- if $h(n) = \infty$ you can prune $n$. 
Pruning: Reachability Analysis

- Imagine we expand A and can find out that the goal is not reachable from C.

- We can prune the entire sub-tree rooted at C.

- But solving the reachability problem is as hard as solving the planing problem.
  - why?
Reachable-1 Algorithm
Reachable-1 Algorithm

- Solution: perform approximate reachability analysis
  - Recognize only some states from which the goal is unreachable (sound but incomplete)
Reachable-1 Algorithm

- Solution: perform approximate reachability analysis
  - Recognize only some states from which the goal is unreachable (sound but incomplete)
- Reachable-1 algorithm:
  - $S =$ propositions that hold in the current state $s_0$
  - Do {
    - For every action $A$ {
      - If the preconditions of $A$ are in $S$
        - Then add the Add-effects of $A$ to $S$
    }
  }
  - Until $S$ remains unchanged
Reachable-1 Algorithm

- Solution: perform approximate reachability analysis
  - Recognize only some states from which the goal is unreachable
    (sound but incomplete)

- Reachable-1 algorithm:
  - $S =$ propositions that hold in the current state $s_0$
  - Do {
    - For every action $A$
      - If the preconditions of $A$ are in $S$
        - Then add the Add-effects of $A$ to $S$
    }Until $S$ remains unchanged

- Propositions not in $S$ cannot be achieved.
  - If some goal proposition is not in $S$, then problem not solvable from $s_0$
Reachable-1 Algorithm

- Solution: perform approximate reachability analysis
  - Recognize only some states from which the goal is unreachable (sound but incomplete)

- Reachable-1 algorithm:
  - $S =$ propositions that hold in the current state $s_0$
  - Do {
      For every action $A$ {
        If the preconditions of $A$ are in $S$
        Then add the Add-effects of $A$ to $S$
      }
    } Until $S$ remains unchanged

- Propositions not in $S$ cannot be achieved.
  - If some goal proposition is not in $S$, the problem is not solvable from $s_0$

- For every state $s$ reached, prune $s$ if the set of propositions reachable from $s$ does not contain $G$. 
Reachable-1 Algorithm

- **Solution:** perform approximate reachability analysis
  - Recognize only some states from which the goal is unreachable (sound but incomplete)

- **Reachable-1 algorithm:**
  - $S =$ propositions that hold in the current state $s_0$
  - Do {
    - For every action $A$ {
      - If the preconditions of $A$ are in $S$
        - Then add the Add-effects of $A$ to $S$
    }
  }
  - Until $S$ remains unchanged

- Propositions not in $S$ cannot be achieved.
  - If some goal proposition is not in $S$, then problem not solvable from $s_0$

- For every state $s$ reached, prune $s$ if the set of propositions reachable from $s$ does not contain $G$.

- Can be generalized i.e., reachable-$k$
Reachable-1: Example

- Propositions: $p, q, r, s$
- Goal: $s$
- Actions:
  - $a_1: p \rightarrow q \land \neg p$
  - $a_2: q \land p \rightarrow r$
  - $a_3: r \rightarrow s$
Some actions may be irrelevant, they are not useful for achieving the goal.

Relevance analysis: similar to reachability, but backwards

Relevant-1:
- Start with $S=G$
- Add to $S$ all preconditions of actions that achieve some proposition in $G$.

What can we do with the resulting $G$?
Remember!

- FWD/BWD search define a search space
- You still need to select how to search it
  - Select a heuristic function (or blind?)
  - Select pruning technique
  - Select search algorithm
- FWD search is currently the most popular technique, and there are very strong heuristic functions