Automated Planning

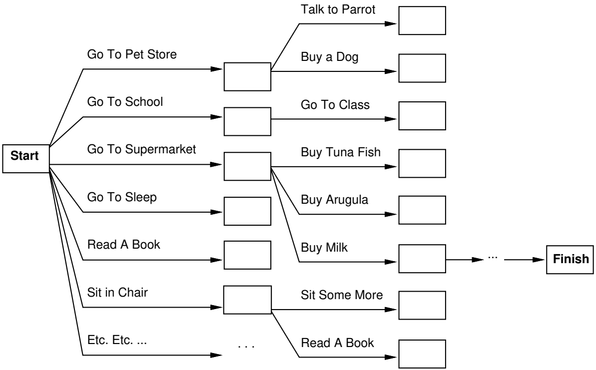
Jihoon Yang

Machine Learning Research Laboratory
Department of Computer Science & Engineering
Sogang University

Search vs. Planning

Consider the task *get milk, bananas, and a cordless drill*

Standard search algorithms seem to fail miserably:



Search vs. Planning

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

CSP, SAT

Planning as state space search

Planning as a search problem: search from the initial state through the space of states, looking for a goal

Algorithms for planning:

– **Progression**: forward state-space search
– **Regression**: backward relevant-space search

Heuristics for planning: need to find good domain-specific heuristics for planning problems

Planning Domain Definition Language (PDDL)

State: a conjunction of fluents that are ground, functionless atoms

Actions: a set of action schema that is a set of ground actions

E.g.,

$$Action : Fly(p, from, to),$$
$$Precond : At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$
$$Effect : \neg At(p, from) \wedge At(p, to)$$

The preconditon and effect are conjunctions of literals that may contain variables

A planning domain is defined by a set of action schemas

– A planning problem within the domain: initial state and a goal (conjunctions of literals)

Example Domain: Air cargo transport

$$
\begin{aligned}
&Init(At(C_1,\ SFO) \land At(C_2,\ JFK) \land At(P_1,\ SFO) \land At(P_2,\ JFK)\\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)\\
&\quad \land Airport(JFK) \land Airport(SFO))\\
&Goal(At(C_1,\ JFK) \land At(C_2,\ SFO))\\
&Action(Load(c,\ p,\ a),\\
&\quad \text{PRECOND: } At(c,\ a) \land At(p,\ a) \land Cargo(c) \land Plane(p) \land Airport(a)\\
&\quad \text{EFFECT: } \neg At(c,\ a) \land In(c,\ p))\\
&Action(Unload(c,\ p,\ a),\\
&\quad \text{PRECOND: } In(c,\ p) \land At(p,\ a) \land Cargo(c) \land Plane(p) \land Airport(a)\\
&\quad \text{EFFECT: } At(c,\ a) \land \neg In(c,\ p))\\
&Action(Fly(p,\ from,\ to),\\
&\quad \text{PRECOND: } At(p,\ from) \land Plane(p) \land Airport(from) \land Airport(to)\\
&\quad \text{EFFECT: } \neg At(p,\ from) \land At(p,\ to))
\end{aligned}
$$

Solution plan:

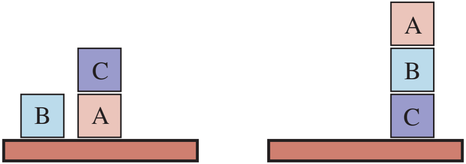
$$
\begin{aligned}
&[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),\\
&\ Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)]
\end{aligned}
$$

Example Domain: Spare tire problem

$Init(Tire(Flat) \wedge Tire(Spare) \wedge At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(obj, loc),$
  PRECOND: $At(obj, loc)$
  EFFECT: $\neg At(obj, loc) \wedge At(obj, Ground))$
$Action(PutOn(t, Axle),$
  PRECOND: $Tire(t) \wedge At(t, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Spare, Axle)$
  EFFECT: $\neg At(t, Ground) \wedge At(t, Axle))$
$Action(LeaveOvernight,$
  PRECOND:
  EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
    $\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Flat, Trunk))$

Solution plan:
$[Remove(Flat, Axle), Remove(Spare, Trunk), PutOn(Spare, Axle)]$

Example Domain: The blocks world



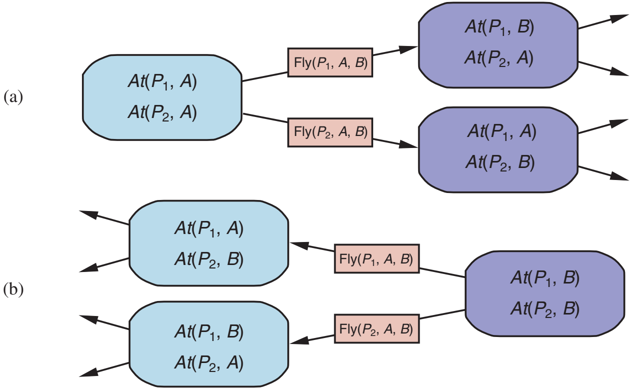
Start State Goal State

$$
\begin{aligned}
&Init(On(A, Table) \land On(B, Table) \land On(C, A) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table)) \\
&Goal(On(A, B) \land On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land \\
&\qquad\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
&\quad \text{EFFECT: } On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Block(b) \land Block(x), \\
&\quad \text{EFFECT: } On(b, Table) \land Clear(x) \land \neg On(b, x))
\end{aligned}
$$

Solution plan:
$[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$

Classical Planning Algorithms

- Forward/Backward state-space search
- Can be solved by applying heuristic search algorithms



cf. Planning as Boolean satisfiability: SAT-based planner with propositional form translated from a PDDL description

Heuristics for Planning

- Recall that an admissible heuristic can be derived by defining a relaxed problem (easier to solve)
- Planning search problem is a graph: **nodes (states), edges (actions)**
- Problem relaxation
  - Add more edges to the graph, making easier to find a path (e.g. ignore-preconditions heuristic, ignore-delete-lists heuristic), or
  - Group multiple nodes together, forming an abstraction of the state space that has fewer states and thus easier to search
- Domain-independent pruning: symmetry reduction, serializable subgoals
- *FastForward (FF; Hoffmann, 2005)*: A system with ignore-delete-list heuristic and nonstandard hill-climbing search (modified to keep track of the plan)

Summary

- Planning systems are problem-solving algorithms that operate on explicit factored representation of states and actions
- PDDL describes initial and goal states as conjunctions of literals, and actions in terms of their preconditions and effects
- State-space search can operate in forward/backward directions, with effective heuristics derived by subgoal independence assumptions and by various relaxations of the planning problem
- Other approaches include encoding a planning problem as a Boolean satisfiability problem or as a constraint satisfaction problem
- Further issues:
  - Hierarchical planning
  - Planning and acting in nondeterministic, partially observable environments
  - Planning and scheduling with resource constraints
- Fast Downward Stone Soup (FDSS): a portfolio (collection of algorithms) planner, a winner in the 2018 International Planning Competition, a ML approach to learn a good portfolio [AAAI 2015]