Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*
Standard search algorithms seem to fail miserably:

- Buy Tuna Fish
- Buy Arugula
- Buy Milk
- Go To Class
- Talk to Parrot
- ...  
- Go To Supermarket
- Buy Tuna Fish
- Go To School
- Buy Arugula
- Buy Milk
- Go To Sleep
- Read A Book
- Sit in Chair
- Etc. Etc. ...

After-the-fact heuristic/goal test inadequate

Search vs. planning contd.

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

<table>
<thead>
<tr>
<th>Search</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>Lisp data structures</td>
</tr>
<tr>
<td>Actions</td>
<td>Lisp code</td>
</tr>
<tr>
<td>Goal</td>
<td>Lisp code</td>
</tr>
<tr>
<td>Plan</td>
<td>Sequence from $S_0$</td>
</tr>
</tbody>
</table>
STRIPS operators

Tidily arranged actions descriptions, restricted language

**Action**: \( Buy(x) \)

**Precondition**: \( At(p), Sells(p,x) \)

**Effect**: \( Have(x) \)

[Note: this abstracts away many important details!]

Restricted language \( \Rightarrow \) efficient algorithm

- Precondition: conjunction of positive literals
- Effect: conjunction of literals
  - positive effect: add literals
  - negative effect: remove literals (negated literals)

Backward State-space Search

- aka Regression Planning
- similar to Backward Chaining
- Difficult if the goal is described as constraints (e.g. 4 gallons in the large jug)—potentially many goal states.
- A goal can be divided into sub-goals (children).

**State-space formulation**

- Initial State: goal state
- Actions: operations that can achieve the goal/sub-goal
  - not undo any super-goals [parent goals/preconditions]
  - successors:
    * sub-goals (unsatisfied preconditions)
- Goal test: no sub-goals (no unsatisfied preconditions)

Forward State-space Search

- aka Progression Planning
- similar to Forward Chaining
- State-space formulation

- Initial State: initial KB
- Actions: operators whose preconditions are satisfied
  - successors:
    * positive effect: add literals
    * negative effect: remove literals (negated literals)
- Goal test: goal state
- Step cost: typically 1

Admissible Heuristics

- Relaxed problem
  - remove all preconditions—every action is applicable
  - remove all negative effects—no action removes a literal (note that the goal is a conjunction of literals)
  - subgoal independence—achieving one subgoal does not affect achieving another subgoal
Keeping track of change—Situation Calculus

Facts hold in situations, rather than eternally.
E.g., \( \text{Holding}(\text{Gold}, \text{Now}) \) rather than just \( \text{Holding}(\text{Gold}) \).

**Situation calculus** is one way to represent change in FOL:
- Adds a situation argument to each non-eternal predicate
- E.g., \( \text{Now} \) in \( \text{Holding}(\text{Gold}, \text{Now}) \) denotes a situation

Situations are connected by the **Result** function:
\( \text{Result}(a, s) \) is the situation that results from doing \( a \) in \( s \).

**Describing actions I**

"Effect" axiom—describe changes due to action
\[ \forall s \quad \text{AtGold}(s) \Rightarrow \text{Holding}(\text{Gold}, \text{Result}(\text{Grab}, s)) \]

"Frame" axiom—describe non-changes due to action
\[ \forall s \quad \text{HaveArrow}(s) \Rightarrow \text{HaveArrow}(\text{Result}(\text{Grab}, s)) \]

Frame problem: find an elegant way to handle non-change
   (a) representation—avoid frame axioms
   (b) inference—avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—
   what if gold is slippery or nailed down or …

Ramification problem: real actions have many secondary consequences—
   what about the dust on the gold, wear and tear on gloves, …

**Describing actions II**

**Successor-state axioms** solve the representational frame problem

Each axiom is "about" a predicate (not an action per se):
\[ P \text{ true afterwards} \Leftrightarrow (\text{an action made } P \text{ true} \lor P \text{ true already and no action made } P \text{ false}) \]

For holding the gold:
\[ \forall a, s \quad \text{Holding}(\text{Gold}, \text{Result}(a, s)) \Leftrightarrow \]
\[ [(a = \text{Grab} \land \text{AtGold}(s)) \lor (\text{Holding}(\text{Gold}, s) \land a \neq \text{Release})] \]

**Making Plans**

Initial condition in KB:
\[ \begin{align*}
\text{At}(&\text{Agent}, [1, 1], S_0) \\
\text{At}(&\text{Gold}, [1, 2], S_0)
\end{align*} \]

Query: \( \text{Ask}(\text{KB}, \exists s \quad \text{Holding}(\text{Gold}, s)) \)
   i.e., in what situation will I be holding the gold?

Answer: \( \{ s/\text{Result}(\text{Grab, Result}(\text{Forward, S}_0)) \} \)
   i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at \( S_0 \) and that \( S_0 \) is the only situation described in the KB.
Represent plans as action sequences \([a_1, a_2, \ldots, a_n]\)

PlanResult\((p, s)\) is the result of executing \(p\) in \(s\)

Then the query \(\text{Ask}(KB, \exists p \text{ Holding}(\text{Gold}, \text{PlanResult}(p, S_0)))\)
has the solution \(\{p/\{\text{Forward, Grab}\}\}\)

Definition of PlanResult in terms of Result:
\[
\forall s \; \text{PlanResult}([], s) = s \\
\forall a, p, s \; \text{PlanResult}([a|p], s) = \text{PlanResult}(p, \text{Result}(a, s))
\]

Planning systems are special-purpose reasoners designed to do this type of
inference more efficiently than a general-purpose reasoner

Partial Order Planning

- sequential planning: forward (or backward) step-by-step search
- Consider planning a trip to New York by flying
  1. start with finding how to get from home to the Melbourne airport
  2. start with finding how to get from the New York airport to hotel
  3. start with finding a plane ticket from Melbourne to New York
- least commitment strategy—delay making commitments to steps that
  are less important/constrained

Components of Partial Order Planning

- **Actions**
  - “Start” action: no preconditions, effects = initial state
  - “Finish” action: preconditions = goal state, no effects
  - (regular) actions with preconditions and effects
- **Ordering constraints** between actions
  - \(A \prec B\): \(A\) is before \(B\) (partial order)
  - LeftSock \prec LeftShoe
- **Causal links** from effect of one action to the precondition of another
  - \(A \rightarrow B\): \(A\) achieves precondition \(c\) for \(B\)
  - LeftSock \(\leftarrow\) \text{LeftSockOn} \rightarrow LeftShoe
  - other actions cannot conflict with the causal link: \(\neg\text{LeftSockOn}\)
- **Open preconditions**
  - not achieved by any action yet
  - planner: add actions until there are no open preconditions
Example

Start
At(Home) Have(Ban.) Have(Drill) Have(Milk)

At(HWS) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)

Have(Milk) At(Home) Have(Ban.) Have(Drill)

Finish

Example

Start
At(Home) Have(Ban.) Have(Drill)

At(HWS) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)

Have(Milk) At(Home) Have(Ban.) Have(Drill)

Finish

Planning process

Operators on partial plans:
- add a link from an existing action to an open condition
- add a step to fulfill an open condition
- order one step wrt another to remove possible conflicts

Gradually move from incomplete/vague plans to complete, correct plans

Backtrack if an open condition is unachievable or if a conflict is unresolvable

Topological Sorting in graphs
POP algorithm sketch

```plaintext
function POP(initial, goal, operators) returns plan
    plan ← Make-Minimal-Plan(initial, goal)
    loop do
        if Solution?(plan) then return plan
        Sneed, c ← SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, Sneed, c)
        RESOLVE-THREATS(plan)
    end

function SELECT-SUBGOAL(plan) returns Sneed, c
    pick a plan step Sneed from STEPS(plan)
    with a precondition c that has not been achieved
    return Sneed, c

procedure CHOOSE-OPERATOR(plan, operators, Sneed, c)
    choose a step Sadd from operators or STEPS(plan) that has c as an effect
    if there is no such step then fail
    add the causal link Sadd −→ Sneed to LINKS(plan)
    add the ordering constraint Sadd < Sneed to ORDERINGS(plan)
    if Sadd is a newly added step from operators then
        add Sadd to STEPS(plan)
        add Start < Sadd < Finish to ORDERINGS(plan)

procedure RESOLVE-THREATS(plan)
    for each Sthreat that threatens a link S_i −→ S_j in LINKS(plan) do
        choose either
        Demotion: Add Sthreat < S_i to ORDERINGS(plan)
        Promotion: Add S_j < Sthreat to ORDERINGS(plan)
        if not CONSISTENT(plan) then fail
    end
```

Clobbering and promotion/demotion

A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., Go(Home) clobbers At(Supermarket):

Demotion: put before Go(Supermarket)

Promotion: put after Buy(Milk)

Properties of POP

Nondeterministic algorithm: backtracks at choice points on failure:
- choice of Sadd to achieve Sneed
- choice of demotion or promotion for clobberer
- selection of Sneed is irrevocable

POP is sound, complete, and systematic (no repetition)

Extensions for disjunction, universals, negation, conditionals

Can be made efficient with good heuristics derived from problem description

Particularly good for problems with many loosely related subgoals
Example: Blocks world

"Sussman anomaly" problem

Start State

Goal State

\[
\begin{align*}
\text{Clear}(x) & \quad \text{On}(x,z) & \quad \text{Clear}(y) \\
\text{PutOn}(x,y) & , & \text{PutOnTable}(x) \\
\neg\text{On}(x,z) & , & \neg\text{Clear}(y) & \quad \text{On}(x,y) \\
\text{Clear}(z) & , & \text{On}(x,y) & , & \text{On}(x,z) \\
\text{Clear}(z) & , & \text{On}(x,y) & , & \text{On}(x,z) \\
\end{align*}
\]

+ several inequality constraints

[Linear planners (find a plan for each subgoal and concatenate the plans) can’t find a solution]

Example contd.

\[
\begin{align*}
\text{On}(A, B) & \quad \text{On}(B, C) \\
\text{START} & , & \text{On}(C, A) & , & \text{On}(A, \text{Table}) & , & \text{Cl}(B) & , & \text{On}(B, \text{Table}) & , & \text{Cl}(C) \\
\text{PutOn}(B, C) & , & \text{PutOn}(A, B) \\
\text{FINISH} & , & \text{On}(A, B) & , & \text{On}(B, C) \\
\end{align*}
\]
Heuristics

Which open precondition to choose?

- most constrained open precondition
  - can be satisfied in the fewest number of ways
- can provide substantial speedups
  - if it can’t be satisfied, stop early and return fail
  - if it can be satisfied by only one way, no choice anyhow and can reduce the number of possibilities later on