Artlificial Intelligence

Planning

Christopher Simpkins

Classical Planning

Classical planning is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.

PDDL: Planning Domain Definition Language

PDDL

Action schema precondition effect

Action schema:

$$
\begin{aligned}
&Action(Fly(p, from, to),\\
&\quad PRECOND : At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)\\
&\quad EFFECT : \neg At(p, from) \land At(p, to))
\end{aligned}
$$

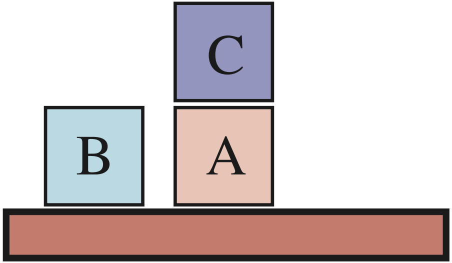
Ground (variable-free) action:

$$
\begin{aligned}
&Action(Fly(P_1, SFO, JFK),\\
&\quad PRECOND : At(P_1, SFO) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK)\\
&\quad EFFECT : \neg At(P_1, SFO) \land At(P_1, JFK))
\end{aligned}
$$

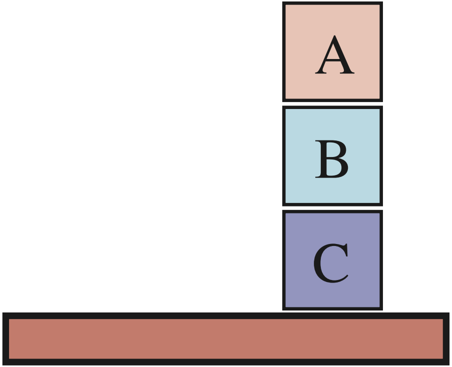
Air Cargo Transport

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

Blocks World



Start State



Goal State

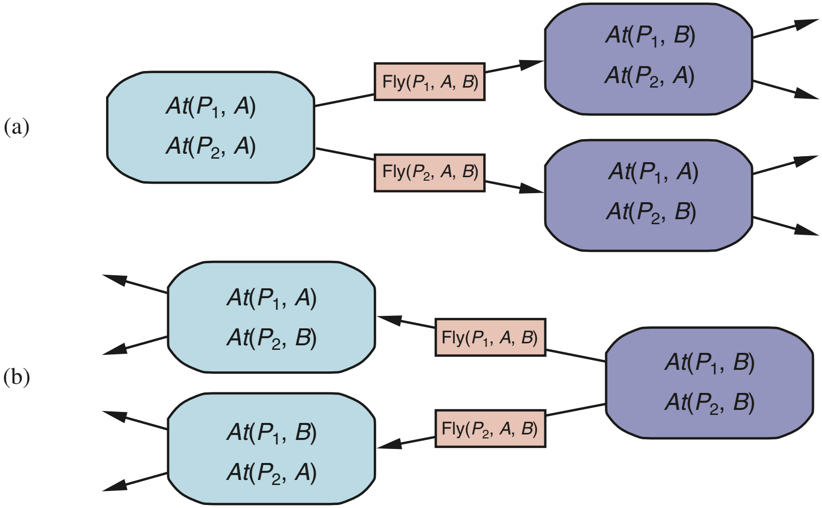
Blocks World PDDL

$$
\begin{array}{l}
\mathit{Init}(\mathit{On}(A,\mathit{Table}) \land \mathit{On}(B,\mathit{Table}) \land \mathit{On}(C,A) \\
\quad \land \mathit{Block}(A) \land \mathit{Block}(B) \land \mathit{Block}(C) \land \mathit{Clear}(B) \land \mathit{Clear}(C) \land \mathit{Clear}(\mathit{Table})) \\
\mathit{Goal}(\mathit{On}(A,B) \land \mathit{On}(B,C)) \\
\mathit{Action}(\mathit{Move}(b,x,y), \\
\quad \textsc{Precond}\text{:}\ \mathit{On}(b,x) \land \mathit{Clear}(b) \land \mathit{Clear}(y) \land \mathit{Block}(b) \land \mathit{Block}(y) \land \\
\qquad\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
\quad \textsc{Effect}\text{:}\ \mathit{On}(b,y) \land \mathit{Clear}(x) \land \neg \mathit{On}(b,x) \land \neg \mathit{Clear}(y)) \\
\mathit{Action}(\mathit{MoveToTable}(b,x), \\
\quad \textsc{Precond}\text{:}\ \mathit{On}(b,x) \land \mathit{Clear}(b) \land \mathit{Block}(b) \land \mathit{Block}(x), \\
\quad \textsc{Effect}\text{:}\ \mathit{On}(b,\mathit{Table}) \land \mathit{Clear}(x) \land \neg \mathit{On}(b,x))
\end{array}
$$

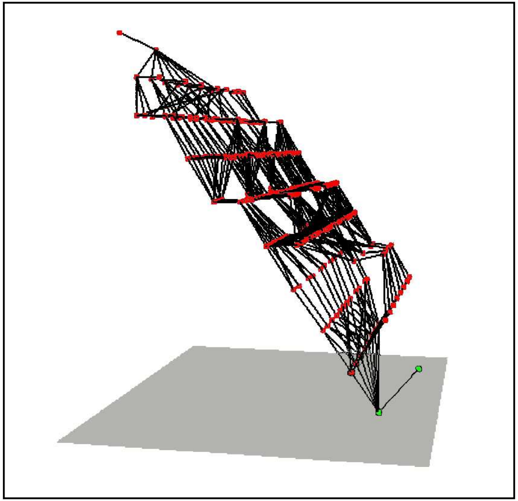
Classical Planning Algorithms

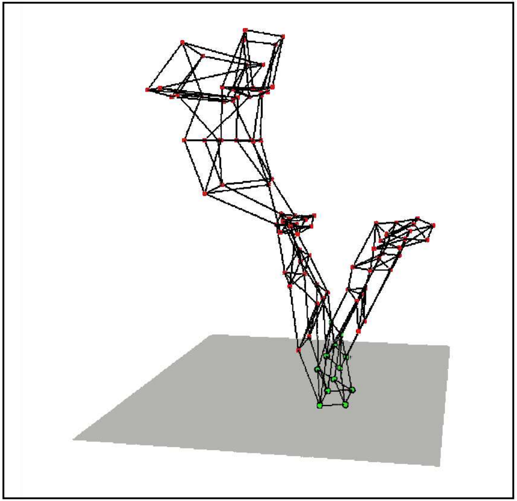
- Forward state space search
- Backward state space search
- SATPlan
- Graphplan
- Situation calculus
- Constraint satisfaction
- Partial-order planning

Forward and Backward State Space Planning



Heuristics for Planning





Hierarchical Planning

Hierarchical task network plans are built from:

- primitive actions, and
- high-level actions (HLA).

HLAs have one or more **refinements**.

- Refinements may contain other HLAs.
- A refinement with only primitive actions is an **implementation**.
- An HLA achieves a goal if at least one of its implementations achieves the goal.

Here are two goal-achieving implementations for the $Go(Home, SFO)$ HLA:

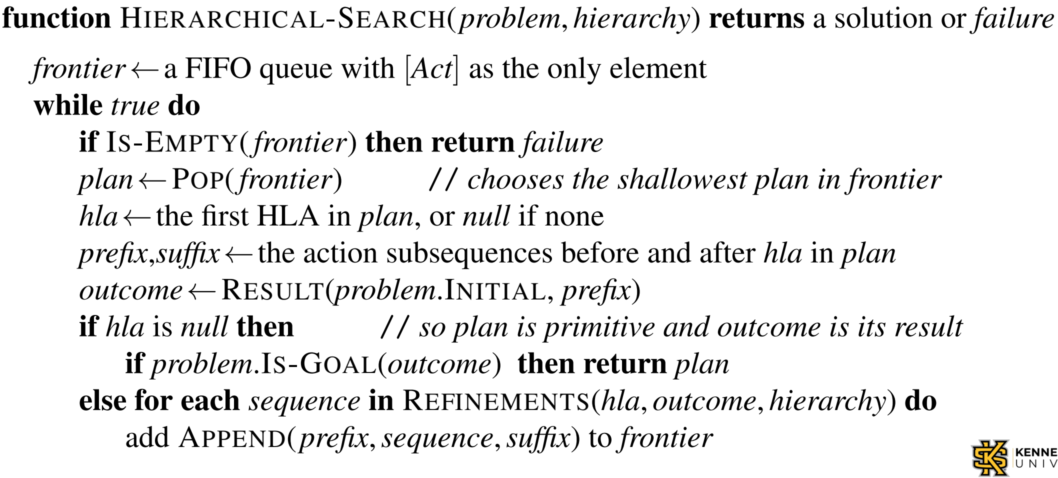
$$\begin{array}{l} \textit{Refinement}(\textit{Go}(\textit{Home},\textit{SFO}), \\ \quad \textsc{Steps}\text{: } [\textit{Drive}(\textit{Home},\textit{SFOLongTermParking}), \\ \qquad\qquad \textit{Shuttle}(\textit{SFOLongTermParking},\textit{SFO})]\ ) \\ \textit{Refinement}(\textit{Go}(\textit{Home},\textit{SFO}), \\ \quad \textsc{Steps}\text{: } [\textit{Taxi}(\textit{Home},\textit{SFO})]\ ) \end{array}$$

Refinements can be produced recursivley, as shown in this vacuum world navigation example:

$$\begin{array}{l} \textit{Refinement}(\textit{Navigate}([a,b],[x,y]), \\ \quad \textsc{Precond}\text{: } a=x \land b=y \\ \quad \textsc{Steps}\text{: } []\ ) \\ \textit{Refinement}(\textit{Navigate}([a,b],[x,y]), \\ \quad \textsc{Precond}\text{:} \textit{Connected}([a,b],[a-1,b]) \\ \quad \textsc{Steps}\text{: } [\textit{Left},\textit{Navigate}([a-1,b],[x,y])]\ ) \\ \textit{Refinement}(\textit{Navigate}([a,b],[x,y]), \\ \quad \textsc{Precond}\text{:} \textit{Connected}([a,b],[a+1,b]) \\ \quad \textsc{Steps}\text{: } [\textit{Right},\textit{Navigate}([a+1,b],[x,y])]\ ) \\ \ldots \end{array}$$

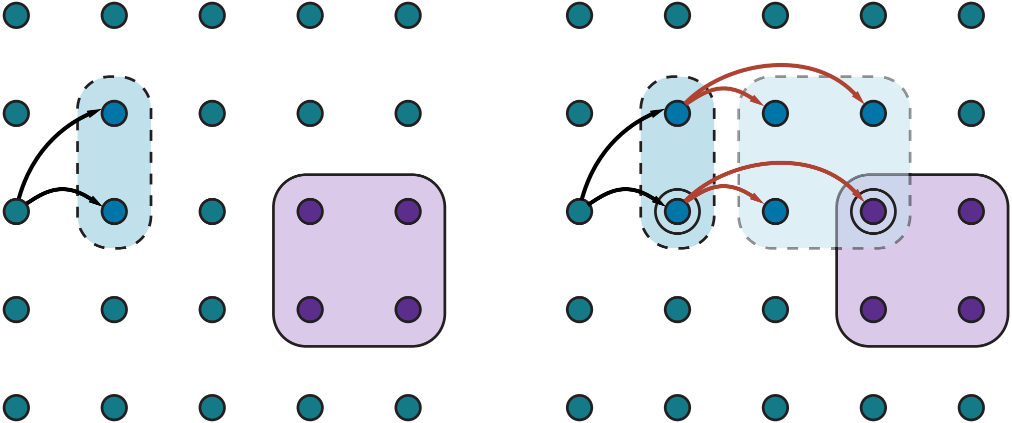
Hierarchical Forward Planning Search

A breadth-first implementation of hierarchical forward planning search. The initial plan supplied to the algorithm is *[Act]*. The REFINEMENTS function returns a set of action sequences, one for each refinement of the HLA whose preconditions are satisfied by the specified state, *outcome*.



```
function HIERARCHICAL-SEARCH(problem, hierarchy) returns a solution or failure
  frontier ← a FIFO queue with [Act] as the only element
  while true do
      if IS-EMPTY(frontier) then return failure
      plan ← POP(frontier)          // chooses the shallowest plan in frontier
      hla ← the first HLA in plan, or null if none
      prefix,suffix ← the action subsequences before and after hla in plan
      outcome ← RESULT(problem.INITIAL, prefix)
      if hla is null then          // so plan is primitive and outcome is its result
          if problem.IS-GOAL(outcome)  then return plan
      else for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
          add APPEND(prefix, sequence, suffix) to frontier
```

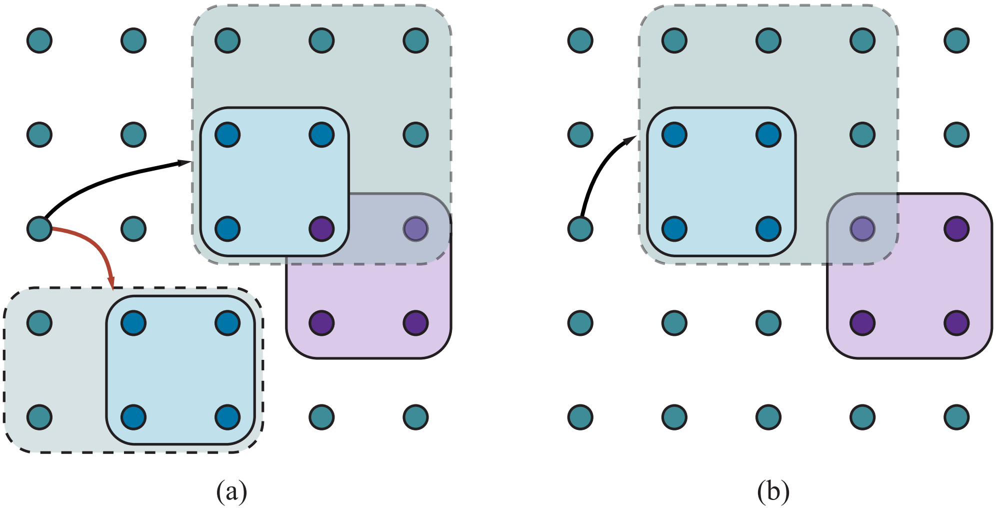
Reachable Sets



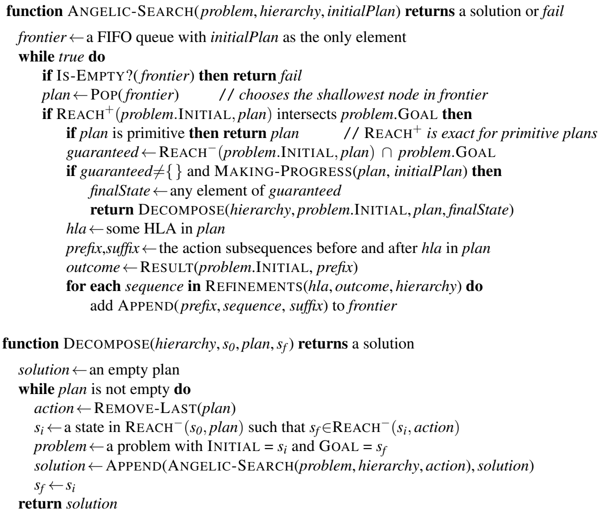
(a)

(b)

Goal Acievement



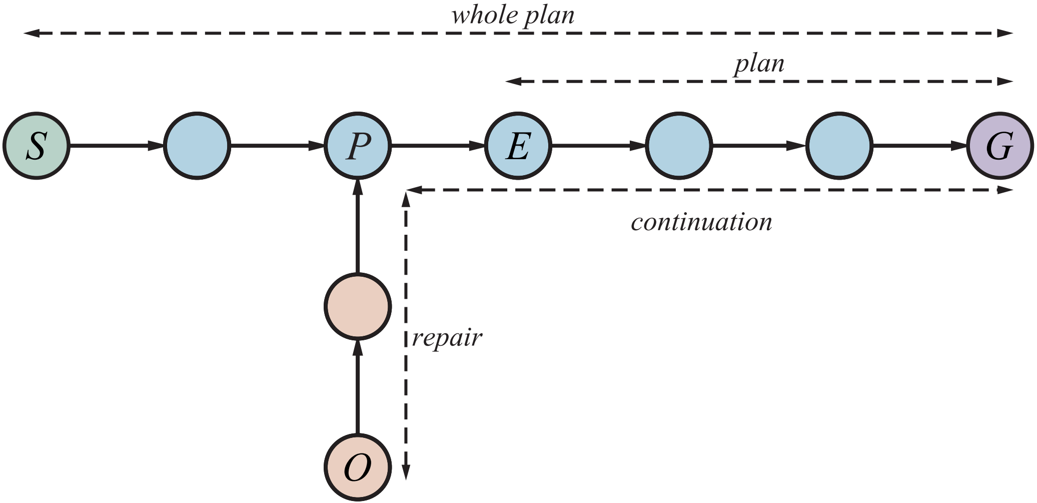
Angelic Search



```
function ANGELIC-SEARCH(problem, hierarchy, initialPlan) returns a solution or fail
  frontier ← a FIFO queue with initialPlan as the only element
  while true do
     if IS-EMPTY?(frontier) then return fail
     plan ← POP(frontier)      // chooses the shallowest node in frontier
     if $\text{REACH}^+$(problem.INITIAL, plan) intersects problem.GOAL then
        if plan is primitive then return plan      // $\text{REACH}^+$ is exact for primitive plans
        guaranteed ← $\text{REACH}^-$(problem.INITIAL, plan) $\cap$ problem.GOAL
        if guaranteed $\neq \{\}$ and MAKING-PROGRESS(plan, initialPlan) then
           finalState ← any element of guaranteed
           return DECOMPOSE(hierarchy, problem.INITIAL, plan, finalState)
        hla ← some HLA in plan
        prefix,suffix ← the action subsequences before and after hla in plan
        outcome ← RESULT(problem.INITIAL, prefix)
        for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
           add APPEND(prefix, sequence, suffix) to frontier

function DECOMPOSE(hierarchy, $s_0$, plan, $s_f$) returns a solution
  solution ← an empty plan
  while plan is not empty do
    action ← REMOVE-LAST(plan)
    $s_i$ ← a state in $\text{REACH}^-(s_0, plan)$ such that $s_f \in \text{REACH}^-(s_i, action)$
    problem ← a problem with INITIAL = $s_i$ and GOAL = $s_f$
    solution ← APPEND(ANGELIC-SEARCH(problem, hierarchy, action), solution)
    $s_f \leftarrow s_i$
  return solution
```

Online Planning



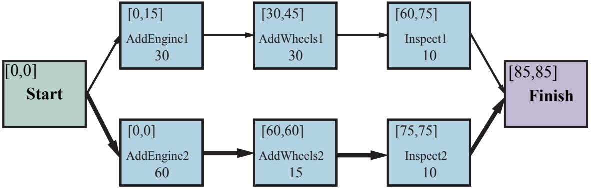
Resource Constraints

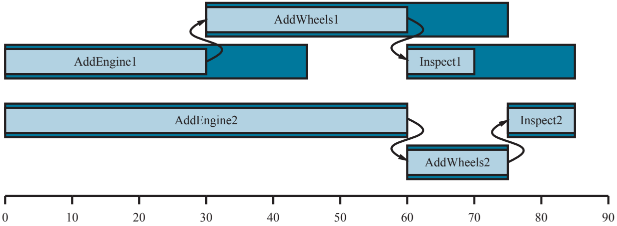
$$\begin{array}{l} \mathit{Jobs}(\{\mathit{AddEngine1} \prec \mathit{AddWheels1} \prec \mathit{Inspect1}\}, \\ \qquad \{\mathit{AddEngine2} \prec \mathit{AddWheels2} \prec \mathit{Inspect2}\}) \end{array}$$

$$\mathit{Resources}(\mathit{EngineHoists}(1),\ \mathit{WheelStations}(1),\ \mathit{Inspectors}(2),\ \mathit{LugNuts}(500))$$

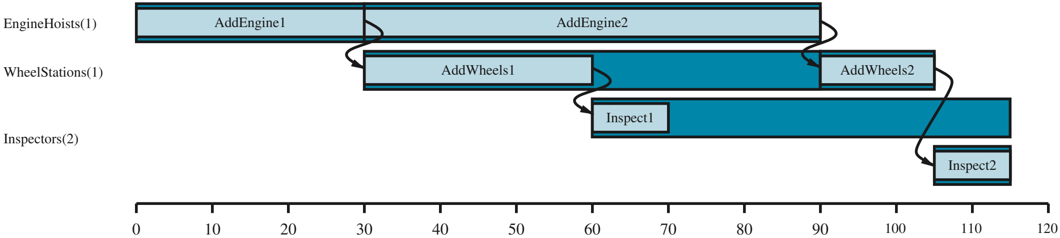
$$\begin{array}{l} \mathit{Action}(\mathit{AddEngine1},\ \textsc{Duration}{:}30, \\ \qquad \textsc{Use}{:}\mathit{EngineHoists}(\mathit{1})) \\ \mathit{Action}(\mathit{AddEngine2},\ \textsc{Duration}{:}60, \\ \qquad \textsc{Use}{:}\mathit{EngineHoists}(\mathit{1})) \\ \mathit{Action}(\mathit{AddWheels1},\ \textsc{Duration}{:}30, \\ \qquad \textsc{Consume}{:}\mathit{LugNuts}(20),\ \textsc{Use}{:}\mathit{WheelStations}(1)) \\ \mathit{Action}(\mathit{AddWheels2},\ \textsc{Duration}{:}15, \\ \qquad \textsc{Consume}{:}\mathit{LugNuts}(20),\ \textsc{Use}{:}\mathit{WheelStations}(1)) \\ \mathit{Action}(\mathit{Inspect}_i,\ \textsc{Duration}{:}10, \\ \qquad \textsc{Use}{:}\mathit{Inspectors}(1)) \end{array}$$

Temporal Constraints





Job-Schop Scheduling Solutions


EngineHoists(1)
WheelStations(1)
Inspectors(2)
AddEngine1
AddEngine2
AddWheels1
AddWheels2
Inspect1
Inspect2