Artlificial Intelligence

Planning

Christopher Simpkins

KENNESAW STATE UNIVERSITY

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 Example:

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

Ground (variable-free) action:

$$\begin{aligned}&Action(Fly(P1, SFO, JFK),\\&\quad PRECOND: At(P1, SFO) \wedge Plane(P1) \wedge Airport(SFO) \wedge Airport(JFK)\\&\quad EFFECT: \neg At(P1, SFO) \wedge At(P1, JFK))\end{aligned}$$

Air Cargo Transport

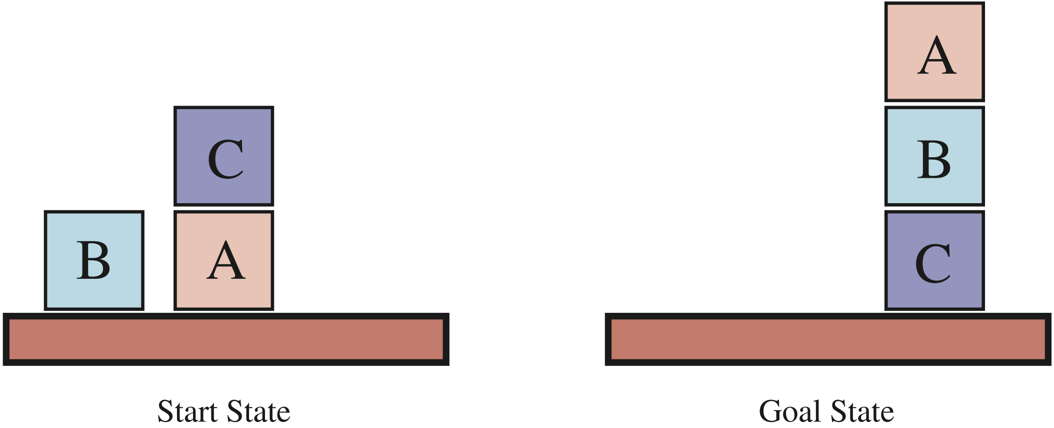
$$
\begin{aligned}
&Init(At(C_1,\ SFO) \land At(C_2,\ JFK) \land At(P_1,\ SFO) \land At(P_2,\ JFK) \\
&\qquad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\qquad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1,\ JFK) \land At(C_2,\ SFO)) \\
&Action(Load(c,\ p,\ a), \\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

KENNESAW STATE UNIVERSITY

Spare Tire

$$
\begin{aligned}
&Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))\\
&Goal(At(Spare, Axle))\\
&Action(Remove(obj, loc),\\
&\quad \textsc{Precond}{:}\ At(obj, loc)\\
&\quad \textsc{Effect}{:}\ \neg At(obj, loc) \land At(obj, Ground))\\
&Action(PutOn(t,\ Axle),\\
&\quad \textsc{Precond}{:}\ Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle) \land \neg At(Spare, Axle)\\
&\quad \textsc{Effect}{:}\ \neg At(t, Ground) \land At(t, Axle))\\
&Action(LeaveOvernight,\\
&\quad \textsc{Precond}{:}\\
&\quad \textsc{Effect}{:}\ \neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)\\
&\qquad\qquad \land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat,\ Trunk))
\end{aligned}
$$

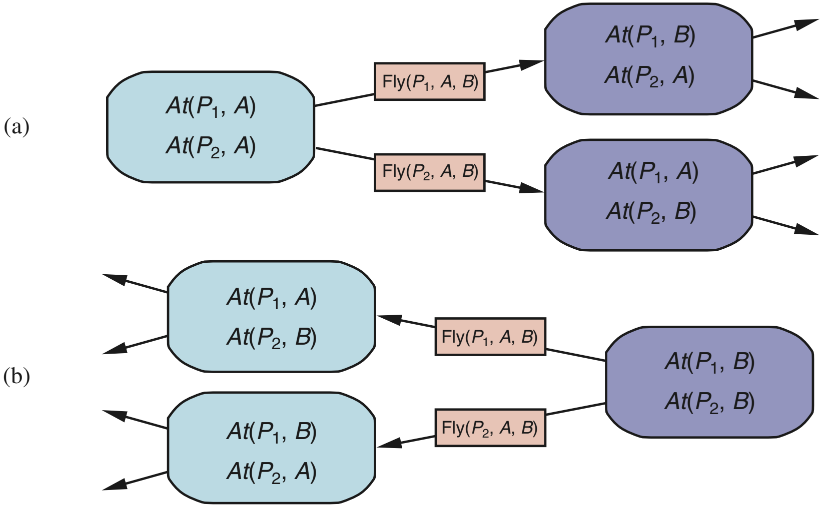
Blocks World



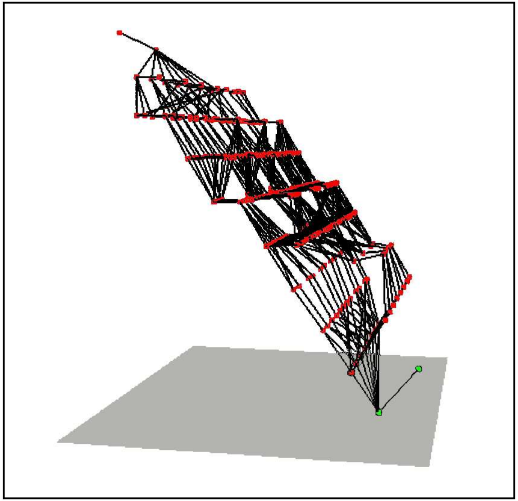
Blocks World

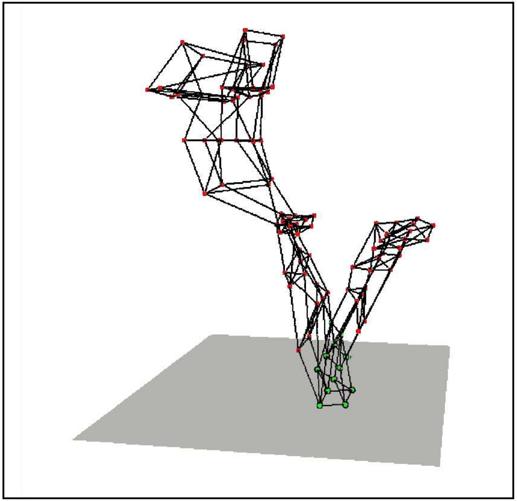
$$
\begin{aligned}
&Init(On(A, Table) \land On(B, Table) \land On(C, A) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table)) \\
&Goal(On(A, B) \land On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \textsc{Precond}: On(b, x) \land Clear(b) \land Clear(y)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

Forward and Backward State Space Planning



Heuristics for Planning





Hierarchical Planning

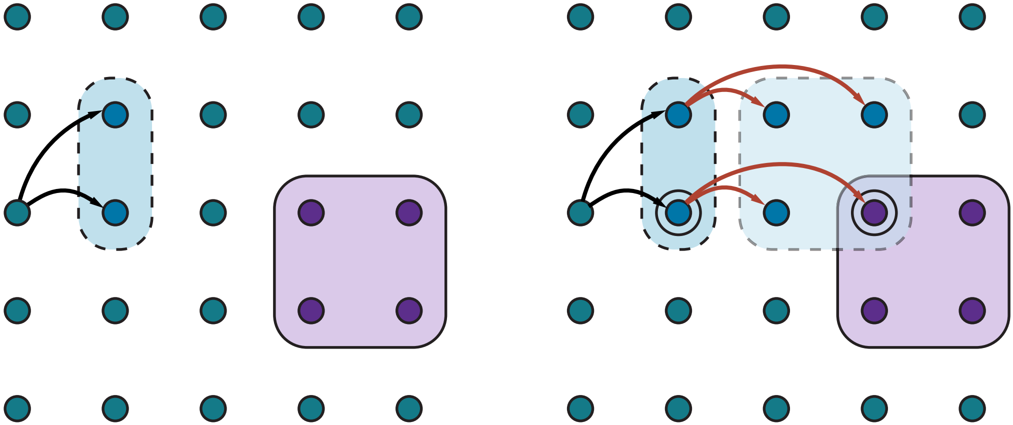
$$
\begin{aligned}
&\mathit{Refinement}(\mathit{Go}(\mathit{Home}, \mathit{SFO}),\\
&\quad \text{STEPS: } [\mathit{Drive}(\mathit{Home}, \mathit{SFOLongTermParking}),\\
&\qquad\qquad \mathit{Shuttle}(\mathit{SFOLongTermParking}, \mathit{SFO})]\ )\\
&\mathit{Refinement}(\mathit{Go}(\mathit{Home}, \mathit{SFO}),\\
&\quad \text{STEPS: } [\mathit{Taxi}(\mathit{Home}, \mathit{SFO})]\ )
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Refinement}(\mathit{Navigate}([a,b],[x,y]),\\
&\quad \text{PRECOND: } a = x \ \wedge\ b = y\\
&\quad \text{STEPS: } [\,]\ )\\
&\mathit{Refinement}(\mathit{Navigate}([a,b],[x,y]),\\
&\quad \text{PRECOND:} \mathit{Connected}([a,b],[a-1,b])\\
&\quad \text{STEPS: } [\mathit{Left}, \mathit{Navigate}([a-1,b],[x,y])]\ )\\
&\mathit{Refinement}(\mathit{Navigate}([a,b],[x,y]),\\
&\quad \text{PRECOND:} \mathit{Connected}([a,b],[a+1,b])\\
&\quad \text{STEPS: } [\mathit{Right}, \mathit{Navigate}([a+1,b],[x,y])]\ )\\
&\ldots
\end{aligned}
$$

Hierarchical Planning

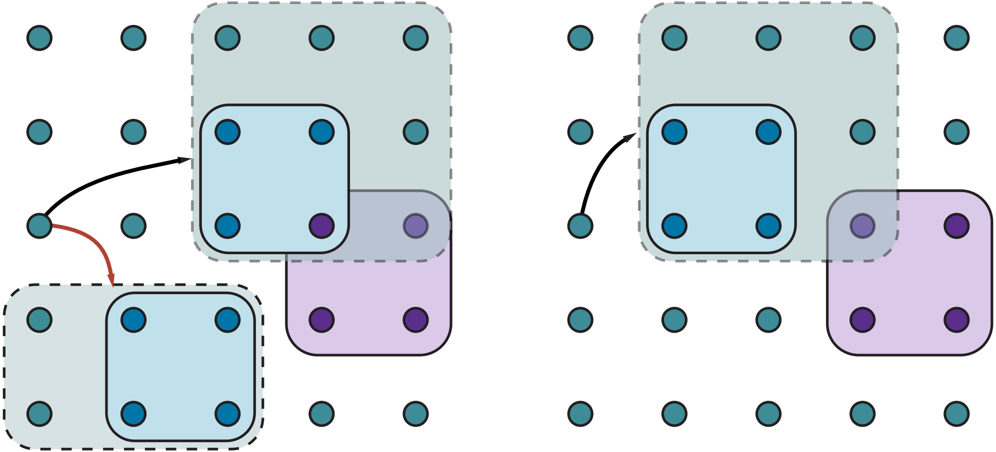
```
function HIERARCHICAL-SEARCH(problem, hierarchy) returns a solution or failure
  frontier ← a FIFO queue with [Act] as the only element
  while true do
     if IS-EMPTY(frontier) then return failure
     plan ← POP(frontier)         // chooses the shallowest plan in frontier
     hla ← the first 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



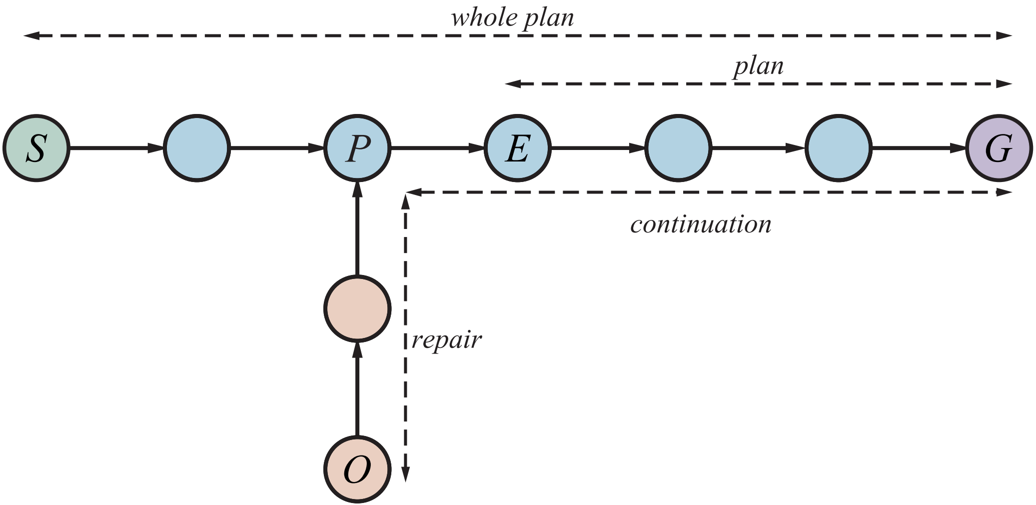
(a) (b)

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 REACH+(problem.INITIAL, plan) intersects problem.GOAL then
          if plan is primitive then return plan        // REACH+ is exact for primitive plans
          guaranteed ← REACH−(problem.INITIAL, plan) ∩ problem.GOAL
          if guaranteed≠{ } 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 REACH−(s_0, plan) such that s_f ∈ 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 ← s_i
  return solution
```

Online Planning

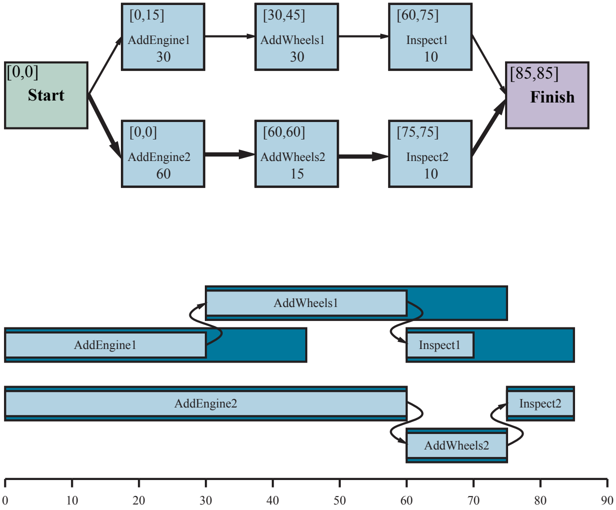


Resource Constraints

$$
\begin{aligned}
&Jobs(\{AddEngine1 \prec AddWheels1 \prec Inspect1\},\\
&\qquad \{AddEngine2 \prec AddWheels2 \prec Inspect2\})\\
\\
&Resources(EngineHoists(1),\ WheelStations(1),\ Inspectors(2),\ LugNuts(500))\\
\\
&Action(AddEngine1,\ \textsc{Duration}{:}30,\\
&\qquad \textsc{Use}{:}EngineHoists(\mathit{1}))\\
&Action(AddEngine2,\ \textsc{Duration}{:}60,\\
&\qquad \textsc{Use}{:}EngineHoists(\mathit{1}))\\
&Action(AddWheels1,\ \textsc{Duration}{:}30,\\
&\qquad \textsc{Consume}{:}LugNuts(20),\ \textsc{Use}{:}WheelStations(1))\\
&Action(AddWheels2,\ \textsc{Duration}{:}15,\\
&\qquad \textsc{Consume}{:}LugNuts(20),\ \textsc{Use}{:}WheelStations(1))\\
&Action(Inspect_i,\ \textsc{Duration}{:}10,\\
&\qquad \textsc{Use}{:}Inspectors(1))
\end{aligned}
$$

KENNESAW STATE UNIVERSITY

Temporal Constraints



Job-Schop Scheduling Solutions