CSE 291 – AI Agents Search for Planning in Simulations

Prithviraj Ammanabrolu

Logistics

- Group signups are due Thursday

- Ideally 4-6 members

- HW0 due tonight

- HW1 released!

Logistics

For project proposals

- What task you're doing

- What env you intend to use? Are you making a sim yourself?
  - What are time estimates for how hard that would be

- Why is this interesting? If you make an agent in this env, who will care?

- Initial ideas on how to solve it (what will you do if X LLM doesn't work?)

Generally speaking the presentation will not be graded, but we will give feedback and expect a revised proposal slide deck to be submitted by the end of the week. That will be graded.

Basic components of a simulation

- S = set of all states
  - propositions that are true: you are in a house, door is open, knife in drawer
- A = set of all actions
  - take knife from drawer, walk through door
- T = transition matrix T: (S, A)$\rightarrow$S
  - (you are in a house & door is open, walk through door) $\rightarrow$ you are outside

There are pre-conditions that need to be met to perform a certain action, and post-conditions that are true after

What's in a PDDL task?

- Objects: Things in the world that interest us.
- Predicates: Properties of objects that we are interested in; can be true or false.
- Initial state: The state of the world that we start in.
- Goal specification: Things that we want to be true.
- Actions/Operators: Ways of changing the state of the world.

2 .pddl files, domain and problem

You have a simulation, now what?

- You need to plan out a policy

- Sequence of actions to get from start state to goal state

- A *plan* gives you a way of getting said policy
  - Can be expressed as constraints on actions you can perform

How to get a plan?

- Search is a way of getting possible plans for a given spec

Search Terminology

- State Space Search - each state is a node on the search tree, go from there

- Planning Space Search – searching through space of possible plans or constraints on actions

- Satisficing – looking longer and longer for a good enough solution

- Optimal – looking for the best possible solution there is

Standard (but not necessary) Assumptions

- No environment stochasticity, exact post conditions always manifest once action is executed

- No agent stochasticity, actions are always executed as planned

- Think about ways not having these assumptions would complicate things in the methods we talk about here on out

Properties of Forward Search

- Sound: plans generated by the traces will guarantee a solution if executed

- Complete: if a solution exists, then at least one of the search's traces will be a solution

Forward Search

- Some deterministic implementations of forward search:
  - breadth-first search
  - depth-first search
  - best-first search (e.g., A*)
  - greedy search
- Breadth-first and best-first search are sound and complete But they usually aren't practical, requiring too much memory
  - Memory requirement is exponential in the length of the solution
- In practice, more likely to use depth-first search or greedy search
  - Worst-case memory requirement is linear in the length of the solution
  - In general, sound but not complete
  - But classical planning has only finitely many states
  - Thus, can make depth-first search complete by doing loop-checking

Forward Search Example – BlocksWorld

unstack(x,y)
   Pre: on(x,y), clear(x), handempty
   Eff: ~on(x,y), ~clear(x), ~handempty,
         holding(x), clear(y)

stack(x,y)
   Pre: holding(x), clear(y)
   Eff: ~holding(x), ~clear(y),
         on(x,y), clear(x), handempty

pickup(x)
   Pre: ontable(x), clear(x), handempty
   Eff: ~ontable(x), ~clear(x), ~handempty, holding(x)

putdown(x)
   Pre: holding(x)
   Eff: ~holding(x), ontable(x), clear(x), handempty

Forward Search Example

Pickup(c) → S_1

...

Pickup(b) → S_i

Initial state

Goal state

Forward Search Issues

- Branching factor - lots of possible states and actions, deterministic searches waste time trying a bunch of unnecessary stuff

- State and action spaces can blow up memory and compute costs
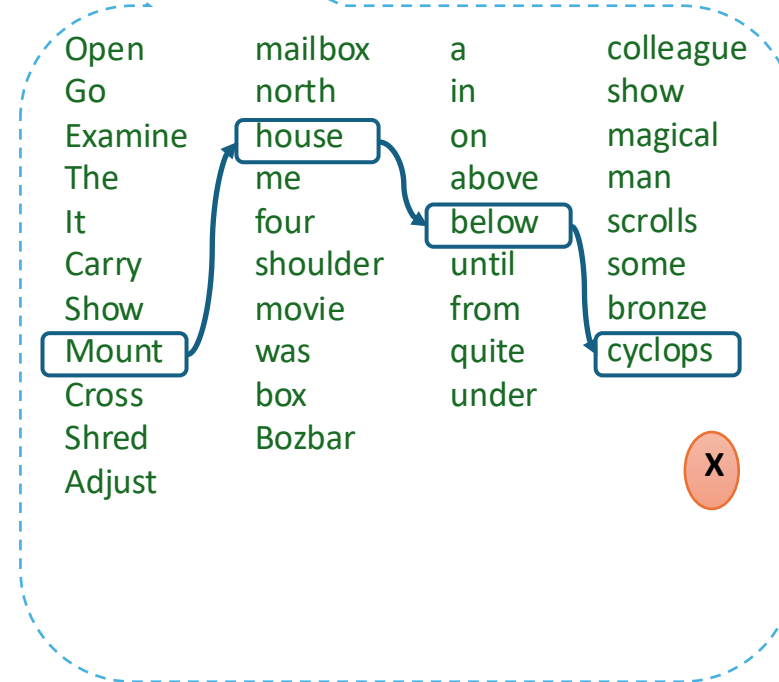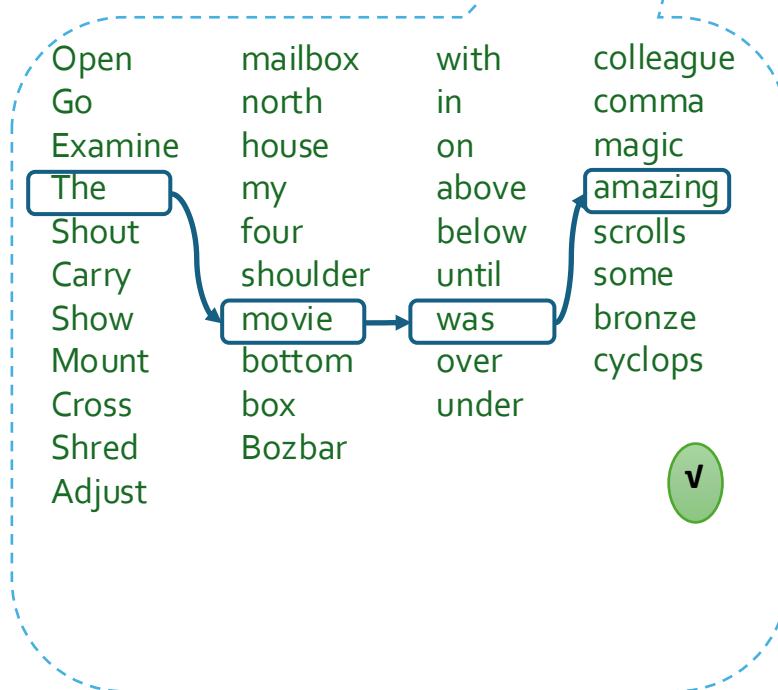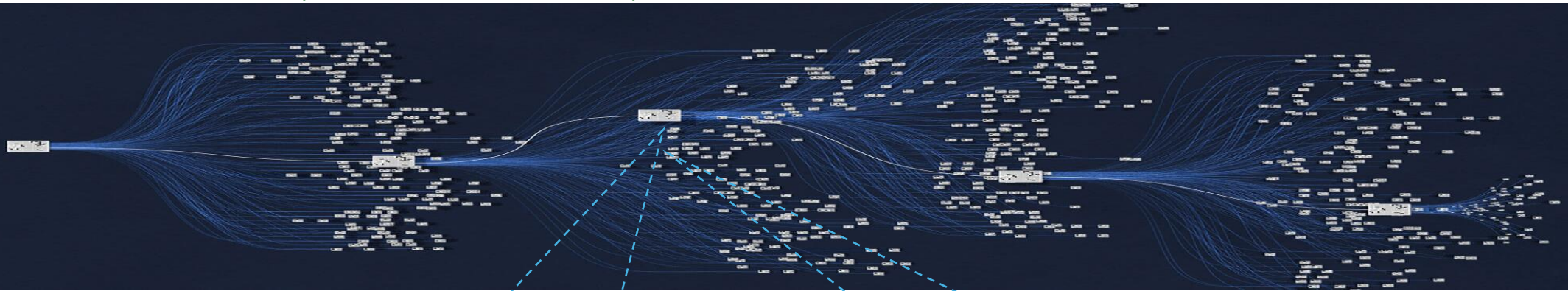
Step 1   Step 2   Step 3   Step 4   ...

Open	mailbox	with	colleague
Go	north	in	comma
Examine	house	on	magic
The	my	above	amazing
Shout	four	below	scrolls
Carry	shoulder	until	some
Show	movie	was	bronze
Mount	bottom	over	cyclops
Cross	box	under	
Shred	Bozbar		
Adjust			

✓

Open	mailbox	a	colleague
Go	north	in	show
Examine	house	on	magical
The	me	above	man
It	four	below	scrolls
Carry	shoulder	until	some
Show	movie	from	bronze
Mount	was	quite	cyclops
Cross	box	under	
Shred	Bozbar		
Adjust			

✗

Imagine a controller with ~50000 buttons. How to scale language planning?
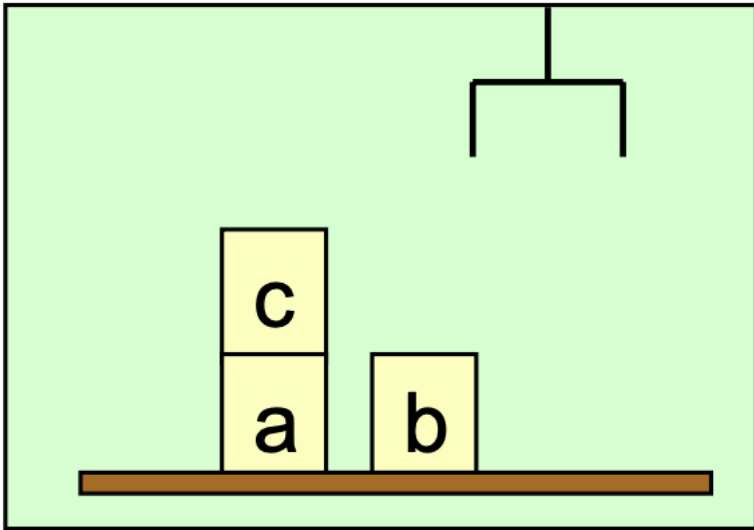**(Game of Go ~250, Chess ~35)**

Backward Search

- For forward search, we started at the initial state and computed state transitions
  - new state = $T(s,a)$
- For backward search, we start at the goal and compute inverse state transitions
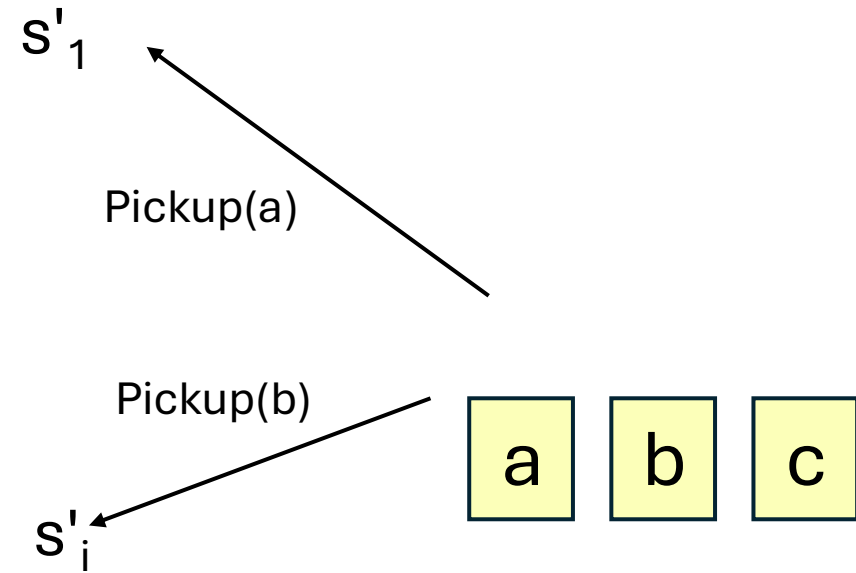 - new set of subgoals = $T^{-1}(g,a)$
- To define $T^{-1}(g,a)$, must first define relevance: An action a is relevant for a goal g if
  - a makes at least one of g's literals true, $g \cap \text{effects}(a) \neq \emptyset$
  - a does not make any of g's literals false, $g+ \cap \text{effects} -(a) = \emptyset$ and $g- \cap \text{effects} +(a) = \emptyset$

Backward Search

- To define $T^{-1}(g,a)$, must first define relevance: An action a is relevant for a goal g if
  - a makes at least one of g's literals true, $g \cap effects(a) \neq \emptyset$
  - a does not make any of g's literals false, $g^+ \cap effects^-(a) = \emptyset$ and $g^- \cap effects^+(a) = \emptyset$
  - If a is relevant for g, then $T^{-1}(g,a) = (g^- effects(a)) \cup precond(a)$
  - Otherwise, $T^{-1}(g,a)$ is undefined

Backward Search Example

Initial state

s'₁

Pickup(a)

...

Pickup(b)

s'ᵢ

Goal state

Backward Search Issues

- Branching factor
  - an operator o that is relevant for g may have many instances a_1, a_2, ..., a_n such that each a_i's input state might be unreachable from the initial state

- Goal states are actually described by constraints instead of exact list of propositions

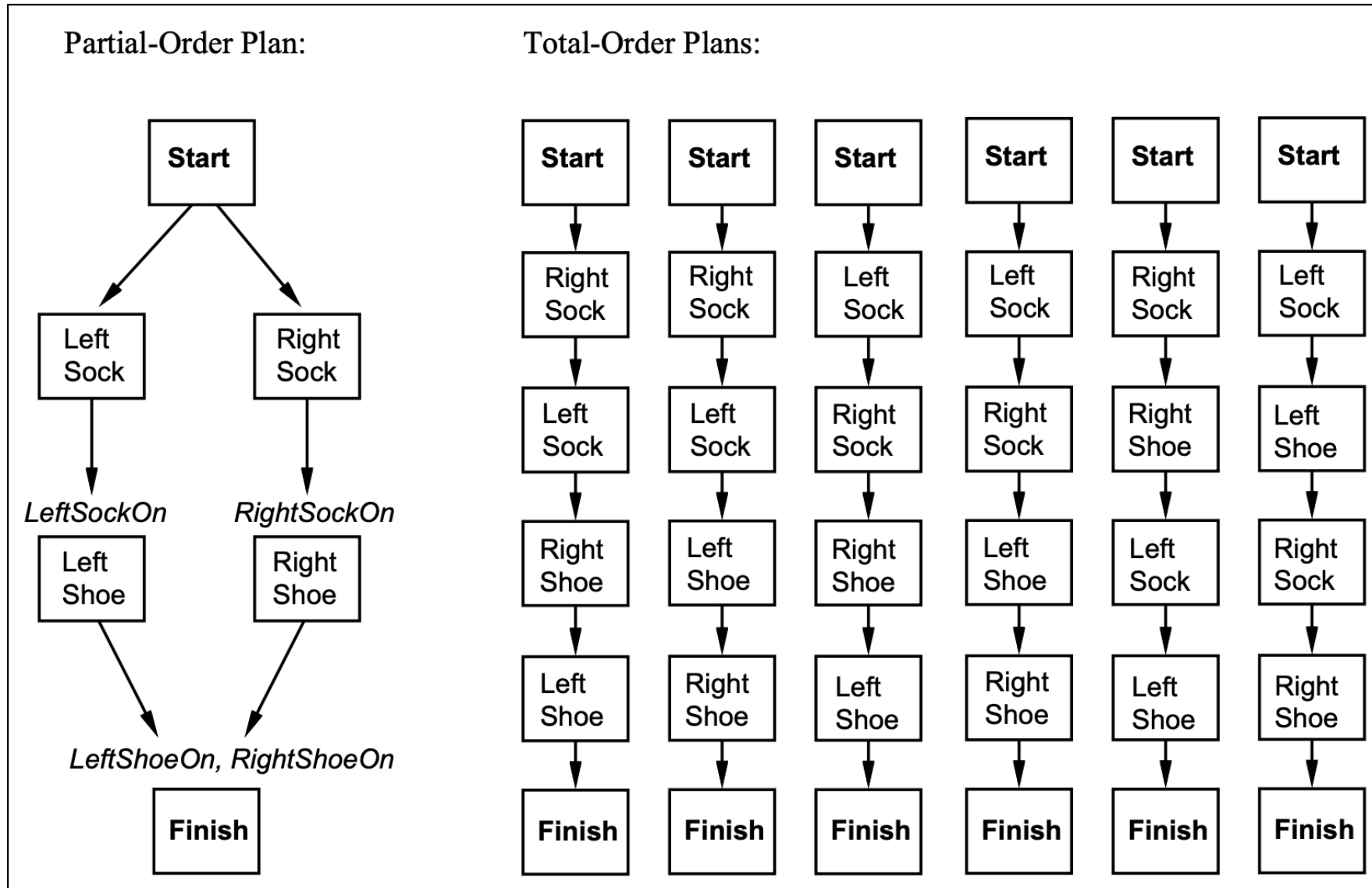- Generating predecessor states (inverting Transition matrix) is hard

Mitigations for Such Issues

- Pruning state or action space … somehow

1. Just describe constraints that need to be satisfied

2. Find a heuristic to move effectively through state space

Total Order and Partial Order Plans

- Exact order of actions may not matter
- If you can break down problem into subproblems, partial planning may be easier → some actions and constraints on when they can be executed
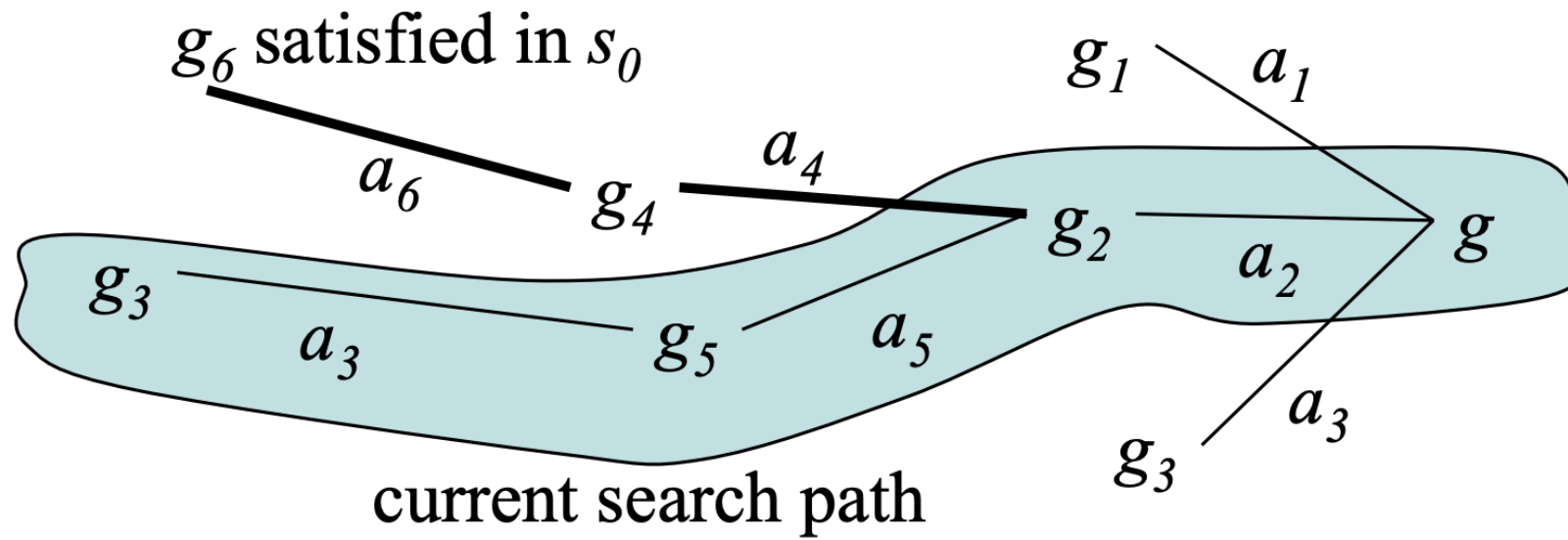- Partially ordered plans = planning space search (rather than state space search)

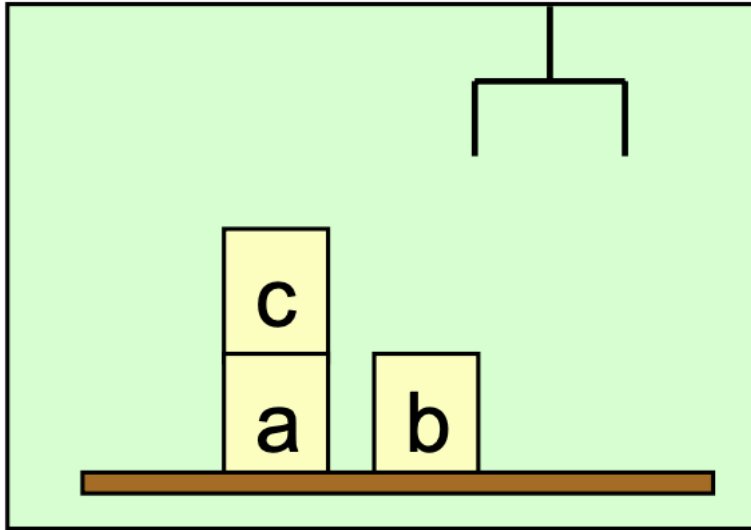# Total Order and Partial Order Plans

Heuristic Planning - STRIPS

- One of the first planning algorithms (Shakey the robot)
- π ← the empty plan
- do a modified backward search from g
  - ** each new subgoal is precond(a)
  - when you find an action that's executable in the current state, then go forward on the current search path as far as possible, executing actions and appending them to π
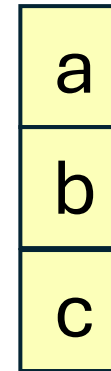 - repeat until all goals are satisfied

Heuristic Planning - STRIPS

STRIPS Example

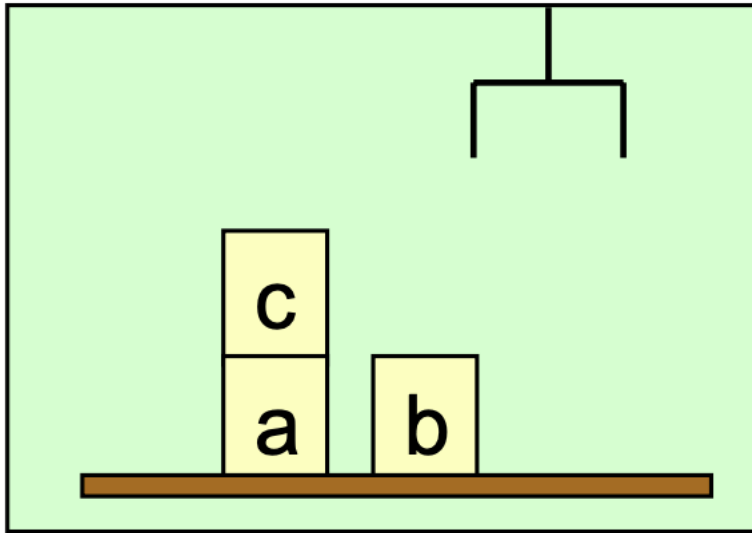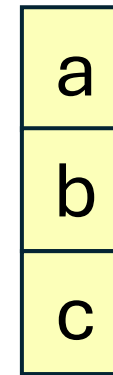- Exercise, solve this like STRIPS would

Initial state

Goal state

Limitation of STRIPS

- Exercise, solve this like STRIPS would
  - Move a on top of b
  - Move b on top of c
  - Contradictory subgoals

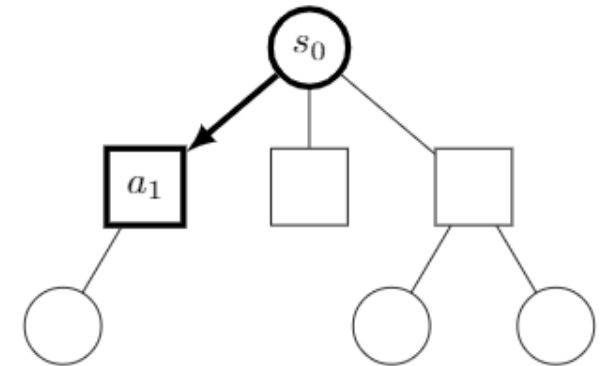Initial state

Goal state

Simulation Search

- Evolution of heuristic search, the model of the world is the heuristic that decides how the agent moves forward

- Use the simulation to build estimates of the "value" of being in a state – intuitively, if I am in a state what is the likelihood I will win

Monte Carlo Methods

- A set of methods that focus on learning better from simulated experiences collected by interacting with an environment

- When to use? You have a way of easily simulating an environment but it is too complex to solve deterministically with planning / search
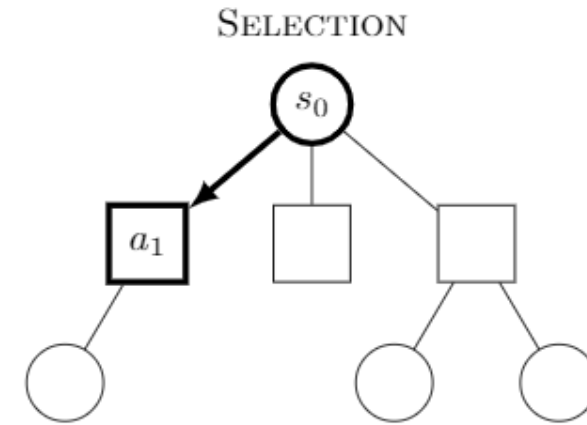
Monte Carlo Tree Search

- 4 phases of building out and simulating paths along a search tree
- Various forms of this used in everything from Alpha Zero to modern LLM inference
- For arbitrary problem with start state s_0 and actions a_i
- All states have attributes:
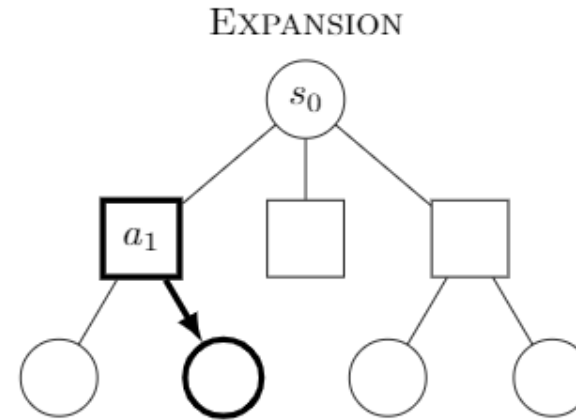 - Total simulation reward Q(s) and
  - Total no. of visits N(s)

MCTS Part 1 - Selection

- From the current state, pick an action to perform

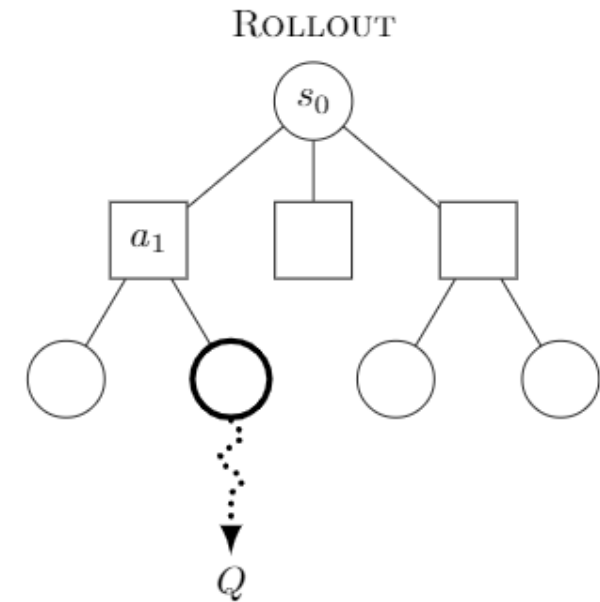- For now, assume we pick randomly

- Update N(s) as you pick a new state

SELECTION

MCTS Part 2 - Expansion

- Execute transition
- If resultant state is a terminal state, observe result (reward)
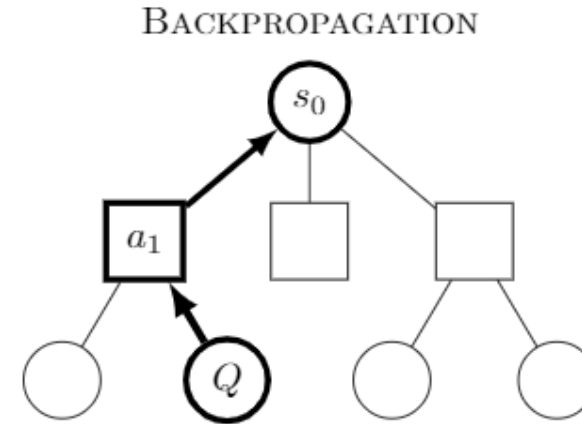
EXPANSION

s_0

a_1

MCTS Part 3 – Simulation / Rollout

- If it isn't a terminal state, finish a playout until it is

- For now, we will "cheat" and directly use our simulation for this

MCTS Part 4 – Backpropogation

- Add the reward of the simulated path to all node scores, this gives you Q(s)

BACKPROPAGATION

Improvements to MCTS Components

- Improvements are possible for each of the parts I talked about
- Think about that it would take to improve selection / expansion phases

Upper Confidence Trees (UCT)

- A way of improving the selection phase by treating selection as a multi-arm bandit problem: which possible action to select that maximizes the possible payout (reward) in the future

$$\text{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\ln N(v)}{N(v_i)}}$$

Upper Confidence Trees (UCT)

- A way of improving the selection phase by treating selection as a multi-arm bandit problem: which possible action to select that maximizes the possible payout (reward) in the future

$$\mathbf{UCT}(v_i, v) = \boxed{\frac{Q(v_i)}{N(v_i)}} + c\sqrt{\frac{\ln N(v)}{N(v_i)}}$$

Exploit

Upper Confidence Trees (UCT)

- A way of improving the selection phase by treating selection as a multi-arm bandit problem: which possible action to select that maximizes the possible payout (reward) in the future

$$\mathbf{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\ln N(v)}{N(v_i)}}$$

Exploit   Explore

Improvements to MCTS Components

- Improvements are possible for each of the parts I talked about
- Think about that it would take to improve selection / expansion phases
- Can you go further? How to improve the simulation phase?
- Can you add learning in here somehow?

In Class Activity

- Socratic Mind
- If you're new register first over here