Introducing Stochasticity

CS 580

Intro to Artificial Intelligence

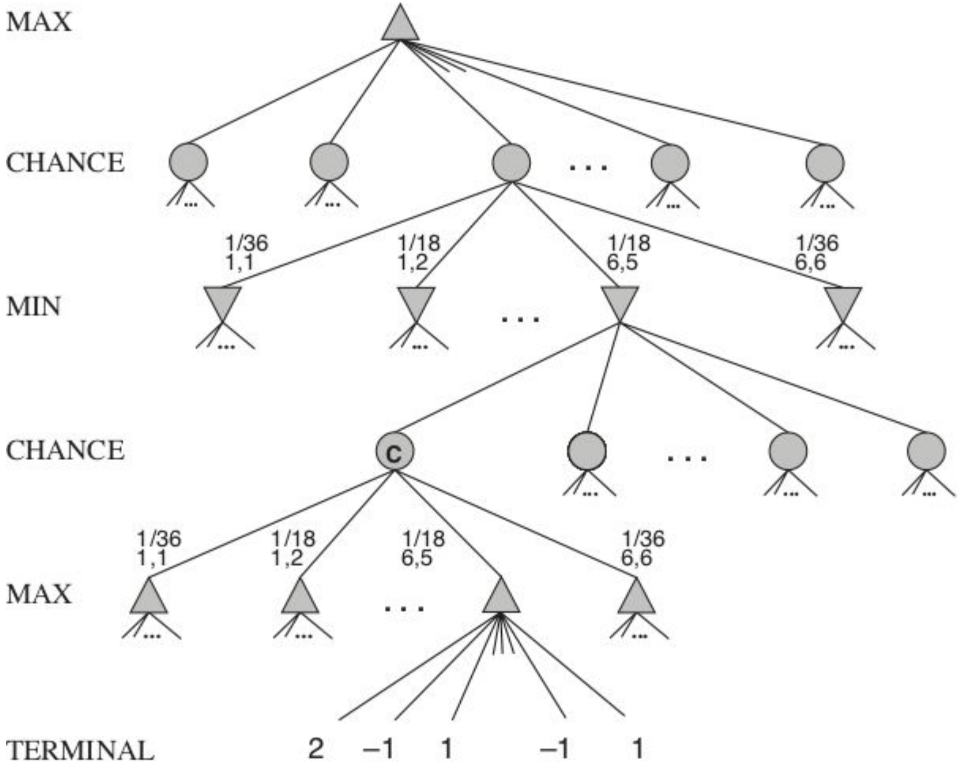
Games of Chance

Plenty of games include elements of randomness

- Poker, Blackjack, Solitaire (shuffling)
- Backgammon, Monopoly (dice)
- Roulette, Pachinko, Slots (mechanical)

We can still use search, if we modify our search tree to include probabilities with edges

For adversarial games, we can introduce another player, “Chance”, and use a slightly modified version of `Minimax`



Expectiminimax

The "`Expectiminimax`" value is the same as `Minimax` except that for chance nodes we take the **expected value** of all the children

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_a P(a)\text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

We end up with a version of adversarial search that uses this instead of `Minimax-Value`.

Stochastic Actions

In the single-agent case, instead of having a "Chance" player, we can model actions as having **non-deterministic** outcomes

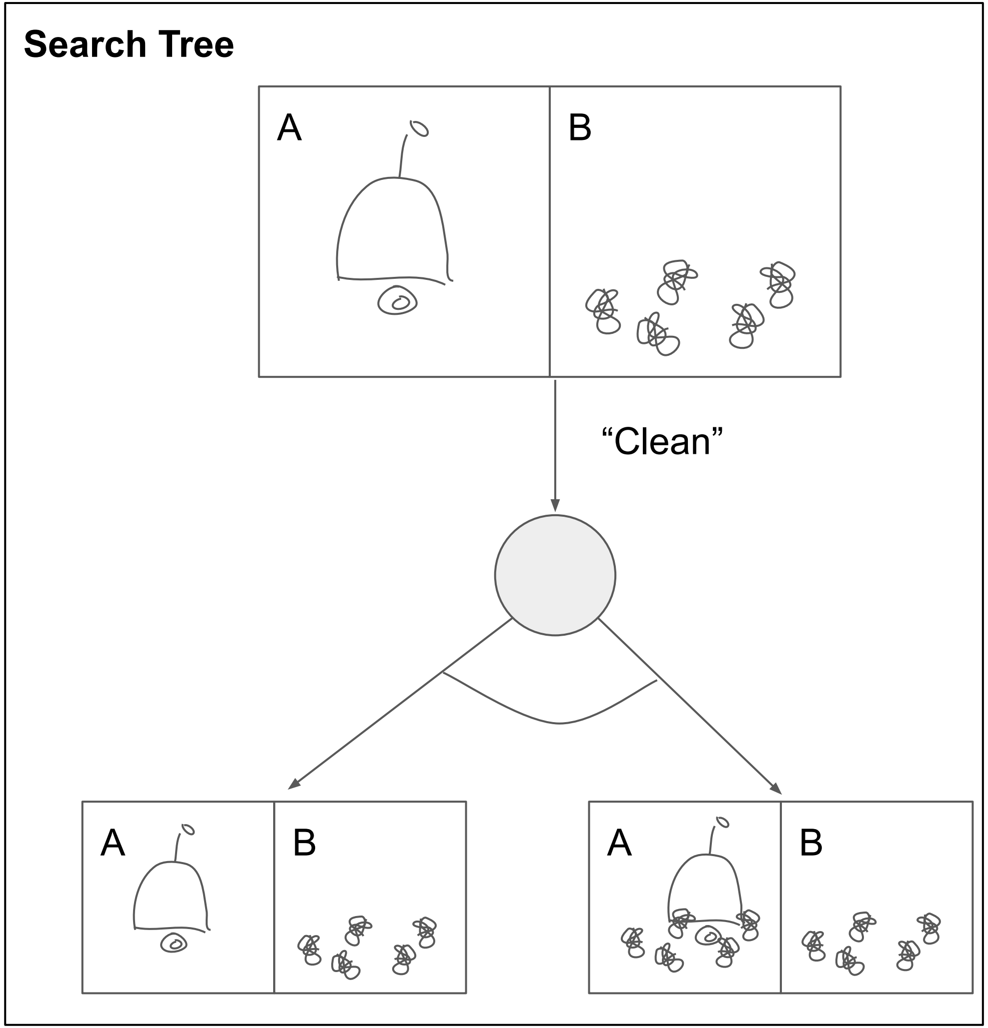
Deterministic actions

`Result`(s,a) = s'

Stochastic actions

`Result`(s,a) = $\{s_1, s_2, \ldots, s_k\}$

This type of representation is called an "`And-Or` search tree"



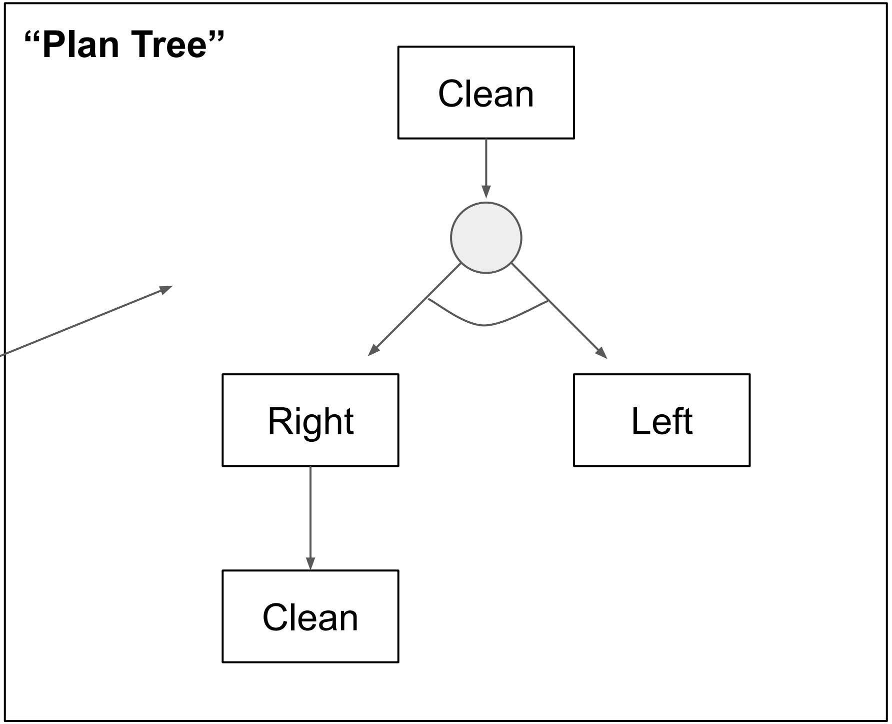
Solutions for `And-Or` search trees (no cycles)

As long as there are no cycles, we can still do search!

The result of searching an `And-Or` tree is no longer a sequence of actions, but a **contingency plan**:

[Clean, **if** S=5 **then** [Right, Clean] **else** [Left]]

An acyclic contingency plan has a nested set of sub-plans for each possible outcome, and can **also** be represented as a tree



One version of `And-Or` Search

```
def and_or_search(prob):
    or_search(prob.initstate,prob,[])
```

```
def or_search(state, prob, path):
    if prob.goal_test(state):
        return empty plan
    if state in path:
        return failure   #cycle
    for action in prob.acts(state):
        S = prob.result(state,action)
        np = [state,]+path
        plan = and_search(S, prob, np)
        if plan != failure:
            return [action,]+plan
    return failure
```

```
def and_search(states, prob, path):
    plans = list()
    for s in states:
        subplan=or_search(s,prob,path)
        if subplan == failure:
            return failure
        plans.append(subplan)
    return plans
```

Like DFS, but with a base-case for cycles, and alternating AND/OR layers. Compare Fig 4.11

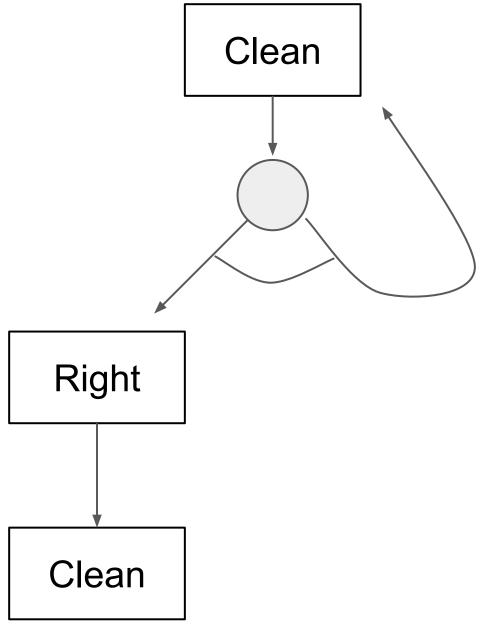
Solutions for `And-Or` search trees (with cycles)

Actually, even if there are cycles, we can *sometimes* find a solution.

If each outcome of a non-deterministic action occurs **eventually**, we can still find a "solution" that will… eventually… get to the goal.

To keep our solutions compact, we can introduce **labels** for repeated steps

[L_1: Clean, **if** S=5 **then** L_1 **else** [Right ...]]

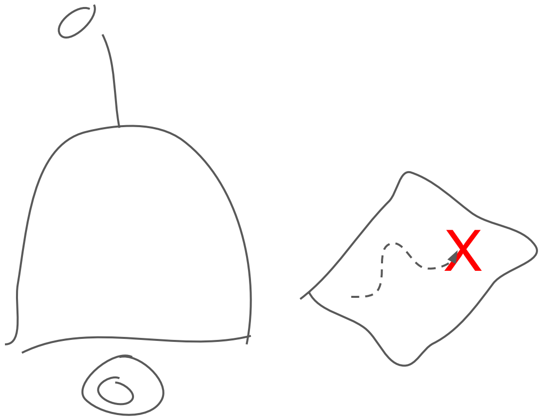


From contingency plans to policy

These “plans” are starting to get increasingly complex…

Instead of a sequence of actions (or a tree of sequences), why not find a **mapping** from state to action so that no matter what state we end up in, we can **immediately** know what to do next? This kind of mapping is called a **policy.**

Analogy
A “plan” is a list of directions
A “policy” is a map marked with arrows



High level review

Uninformed search

BFS, DFS, IDS, UCS

Informed search

A*, heuristics

Adversarial search

`Minimax` and alpha-beta pruning

Non-deterministic

`Expectiminimax`, `And-Or`, and contingency plans

High level preview

<u>Before the midterm</u>

Search
DFS, BFS, UCS, A*, Heuristics, Adversarial Search, Expectiminimax, Contingency Plans

Stochastic actions
Markov Decision Processes, Reinforcement Learning

<u>After the midterm</u>

Decisions with uncertainty
Probability, Bayes Nets, Hidden Markov Models, Filtering

Optimization
Hill Climbing, Simulated Annealing, Evolutionary Algorithms, Gradient Descent

Machine Learning
Linear Regression, Neural Networks, Deep Learning, Decision Trees, Random Forests, Clustering*