CS3243: Introduction to Artificial Intelligence **Fall 2020**

Lecture 2: August 19

Lecturers: Prof. Kuldeep S. Meel *Scribe: Ang Zheng Yong*

2.1 Administrative details

2.1.1 Consultation hours

1. Prof. Kuldeep: Tuesdays 3-4pm
2. Dr. Ler: Fridays 10-11am

2.1.2 Miscellaneous

1. Submissions due on Week 3:
   (a) Lecture notes scribing
   (b) Assignment 1
2. Projects:
   - Project 1 description and grouping will be released next week.
   - There will be a peer review component to ensure fairness. However, teams should have the following to assist in the peer review process:
     (a) Open communication channel (partners should respond to each other within 24 hours).
     (b) List of tasks to be done by each person.

2.2 Recap of Week 1

In week 1, we have covered:

1. Reflex agents
   - Passive.
   - Next state depends only on percept.
2. Model-based reflex agents
   - Passive.
   - Next state depends on the percept, and the model of the world.
3. Goal-based reflex agents
   - Active.

- Next state depends on the current state, the percept, and the model of the world.
- Typically only have a single goal.

4. Utility-based agents
   - Might have multiple conflicting goals, and the agent might not be able to achieve all of its goals.
   - Agent may assign a value (i.e. weight) to each goal, and attempt to balance them.

2.3 More on... agents

Definition 2.1 (Autonomous agents)**.**

Autonomous agents *are agents which can update its agent program based on its experiences.*

The goal of this module is to design *rational autonomous agents*.

2.3.1 How do we model a problem?

Abstraction: we can abstract away the unnecessary details in the problem, and attempt to capture the basic properties of the problem. Then, we can model these properties as a graph.

Let's assume that we have a goal-based agent, in a deterministic and fully-observable task environment, e.g. a mopbot. What will be some useful abstractions in this case?

1. State: $< L, A, B >$
   - L represents the location of the mopbot.
   - A and B represent the status of locations A and B respectively, i.e. whether they are clean or dirty.
2. Actions: {left, right, clean, idle}
   - The list of actions that can be performed by the mopbot.
3. Transition model: g : State $\times$ Action $\rightarrow$ State.
   - Difference between a transition model and an agent function:
     - Agent function: given the current state and percept, what should I do?
     - Transition model: given the current state, what will happen if I perform a certain action?
4. Performance measure (a.k.a. cost function): h : State $\times$ Action $\rightarrow \mathbb{R}$.
5. Goal state: $< *, C, C >$, where C indicates that the location is clean.
   - The goal state may not necessarily only consist a single state.
   - In the case of the mopbot, the goal state can be both $< A, C, C >$ and $< B, C, C >$.
6. Start state

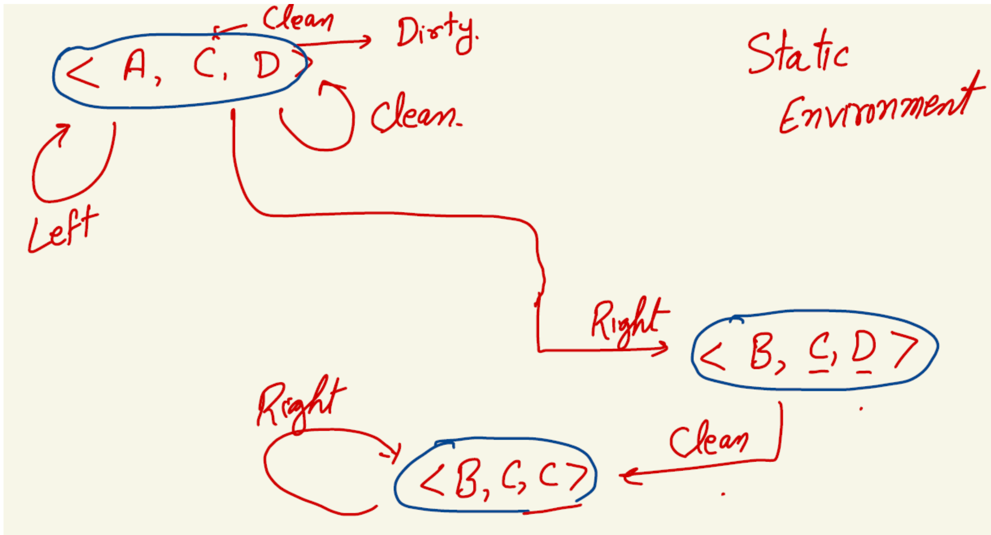

Figure 2.1: Transition model of the mopbot (incomplete)

We can illustrate the transition model using a graph, as shown above. However, the graph representing the transition model can potentially be infinitely large, even for seemingly small problems.[1]

After modelling our problem as a graph, we can now begin exploring the techniques used in graph search.

Remark: In fact, mathematics can be seen as a form of goal-based learning.

- When proving mathematical statements, we often start from an initial statement, and try to reach our final statement (complete our proof).

$$\text{Initial statement} \rightarrow \quad \rightarrow \quad \ldots \quad \rightarrow \quad \rightarrow \text{Final statement}$$

[1] Examples include the $(3n + 1)$ problem, and the Four Fours problem

2.4 Uninformed search

Let us assume that we have obtained the following graph: How can we find a path from s_0 to s_2?

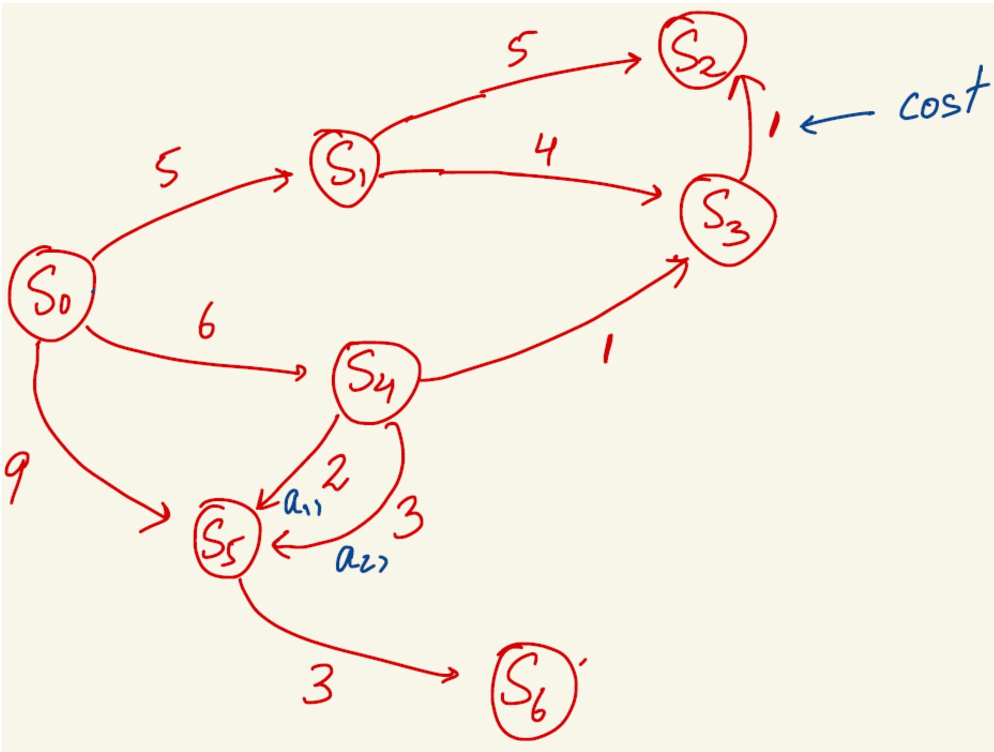

Figure 2.2: A graph with start state s_0 and end state s_2

1. Tree search:
   (a) At our current state, we review our list of performable actions.
   (b) If there is at least one performable action, we will execute the action.
   (c) If we hit a dead end, we will go back and try out another action which hasn't been executed.
   - Limitation: this algorithm makes many unnecessary repeated moves.
     - E.g. after exploring s_6 through $s_0 \to s_5 \to s_6$, we will hit a dead end and start exploring another route from s_0. Let's say we take the route $s_0 \to s_4 \to s_5 \to \ldots$. Then, we will continue traversing s_5 for a second time, even though we have already traversed it before.
2. Graph search:
   - The algorithm is similar to that of tree search, but we will now memoize the nodes that we have traverse.
     - What do we memoize?
       (a) State
       (b) Parent

   (c) Action (that can be carried out at the node)
   (d) Path cost

- Limitation: we will require additional memory to store our memo table (i.e. space-time tradeoff).

3. Breadth-first search

Algorithm 1 BFS

```
procedure BFS(u)
    if GoalTest(u) then
        return path(u)
    end if
    F ← queue(u)
    E ← {u}
    while F is not empty do
        u ← F.pop()
        for all children v in u do
            if GoalTest(v) then
                return path(v)
            else
                if v not in E then
                    E.add(v)
                    F.push(v)
                end if
            end if
        end for
    end while
    return FAIL
end procedure
```

- Limitation: not optimal.

4. Dijkstra.

2.4.1 Algorithmic analysis

How do we determine whether an algorithm is good?

1. Completeness: the algorithm will find a path to the goal, if the goal is reachable from the starting state.
2. Optimality: if the algorithm finds a path, the path will be of minimum cost.
3. Time complexity.
4. Space complexity.

Exercise: Is BFS complete?

Proof:
Assume that there is a path from the start state s_0 to the goal state s_g, i.e. $\pi : s_0, s_{i_1}, \ldots, s_{i_k}, s_g$, of length $k+1$.

Base case: when $k+1=0$, it means that $s_0 = s_g$.

IH: we assume that the algorithm is able to reach nodes at distance $\leq k$.

When we have reached s_{i_k}, we should have either:

- already explored s_g, i.e. s_g is along the path from s_0 to s_{i_k}, or
- s_g will be explored next.

Thus, BFS is complete.

$\square$

Exercise: Is BFS optimal?

Proof:
No. We will proof this by contradiction.

Referring to Figure 2.2, BFS will return $s_0 \to s_1 \to s_2$, instead of $s_0 \to s_4 \to s_3 \to s_2$, which is the actual optimal path.

$\square$

Exercise: What are the time and space complexities of BFS?

Taking b to be the branching factor of our graph, and d to be the depth (i.e. the minimum path length from the start to our goal),

- Time complexity: $b + b^2 + \ldots + b^d = O(b^{d+1})$
- Space complexity: $O(b^{d+1})$

So BFS is not optimal. How can we do better?

1. We can replace the queue used by our frontier F in Algorithm 1 with a priority queue.
   - The nodes will be ordered according to their cost from the starting node.
   - When node u is popped from F, it will be the minimum cost node that is unexplored.
2. We should not mark the child nodes as "explored" too early in Algorithm 1.
   - By marking them as "explored" early on, we will not update/relax their weights after going through them once more from another path.