UNIVERSITY OF BIRMINGHAM

Artificial Intelligence 1
Uninformed Search

Dr Leonardo Stella

School of Computer Science

2023/2024 - Week 8

Aims of the Session

This session aims to help you:

- Understand the concept of asymptotic analysis
- Formulate a search problem
- Explain the steps involved in Breadth-First Search (BFS), Depth-First Search (DFS) and its variations
- Apply the algorithms to solve search problems

Outline

Outline

Outline

Problem-Solving Agents

- In this lecture, we introduce the concept of a goal-based agent called **problem-solving agent**

Definition: An **agent** is 'something', an 'entity', that perceives the world (or the environment) and acts in this environment.

Definition: A problem-solving agent is an agent that

- uses atomic representations (each state of the world is perceived as indivisible);
- requires a precise definition of the problem and its goal/solution.

Formulating a Search Problem

Definition: Formulating a search problem is the process to formally define a search for a solution. A search problem is defined by five components:

- **Initial state**, the state where the agent starts its search;
- **Action set**, the set $\mathcal{A}$ describing the actions that can be executed in any state $s_i \in \mathcal{S}$;
- **Transition model**, a mapping between states and actions, i.e., the states resulting from executing each action $a_i \in \mathcal{A}$ in every state $s_i \in \mathcal{S}$;
- **Goal test**, to determine if a state is a goal state;
- **Path cost function**, which assigns a value (cost) to each path.

- The first three components define the **state space** of the problem
- The state space can take the form of a directed graph or network

Solution, Cost and Path

- The agent's task is to find out how to act, now and in the future
- In other words, the task consists in finding the sequence of actions from the initial state to a goal state

Definition: The **solution** of a search problem is the sequence of actions from the initial state to a goal state. The **cost** of a solution is the sum of the cost of the actions from the initial state to the goal state.

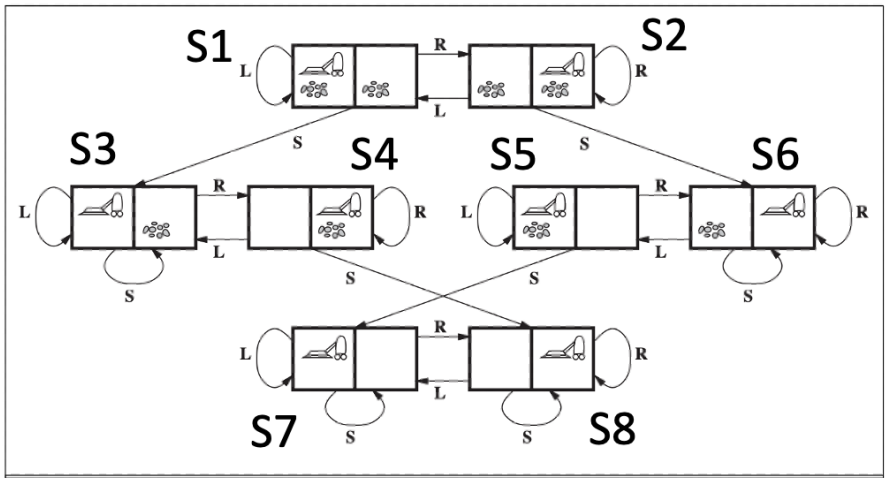
Definition: A **path** in the state space is a sequence of states connected by a sequence of actions.

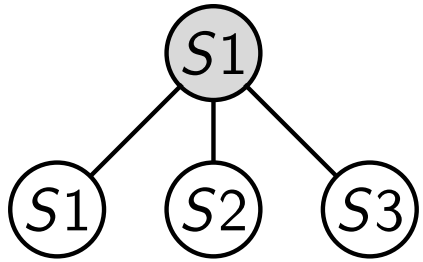
Searching for Solutions

- The solution to a search problem is a sequence of actions from initial state to a goal state
- Possible solutions, or sequences of actions, form a **search tree**:
  - The initial state is placed at the root
  - Actions correspond to the branches
  - Nodes correspond to the state space
- The process consists of expanding the current node (state) by applying each possible actions; it generates a new set of states

Searching for Solutions

- Let us consider the vacuum world example from before
- If $S1$ is the initial state and $\{S7, S8\}$ is the set of goal states, the search tree after expanding the root node is:



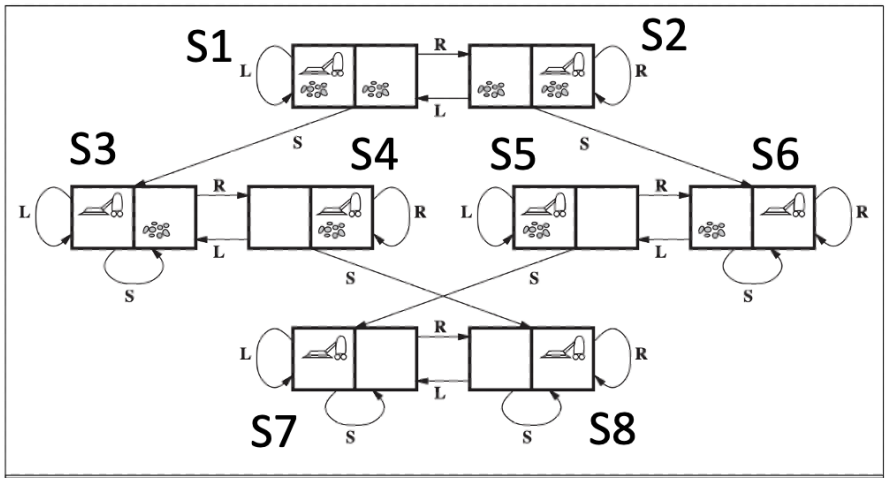


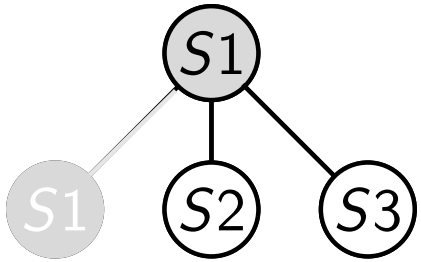
Searching for Solutions

- Each node resulting from the first expansion is a **leaf node**

Definition: The set of all leaf nodes available for expansion at any given time is called the **frontier** (another common term is the **open list**).

- The path from $S1$ to $S1$ is a **loopy path** (or **repeated state**) and in general is not considered unless specified:





Discussion

- It is important to note that typical AI problems have a large number of states and it is virtually impossible to draw the state space graph
- On the contrary, the state space graph for the vacuum world example has a small number of states
- For example, the state space graph for the game of chess is very large
- The process of formulating a search problem is a creative process. However, one formulation can be better than another

Outline

Uninformed Search Strategies

Definition: **Uninformed Search** (also called **blind search**) is a term used to define the set of strategies having no additional information about the state space beyond that provided in the problem formulation.

- Uninformed search strategies can only generate successors and distinguish a goal state from a non-goal state
- The key difference between two uninformed search strategies is the **order** in which nodes are expanded

Breadth-First Search

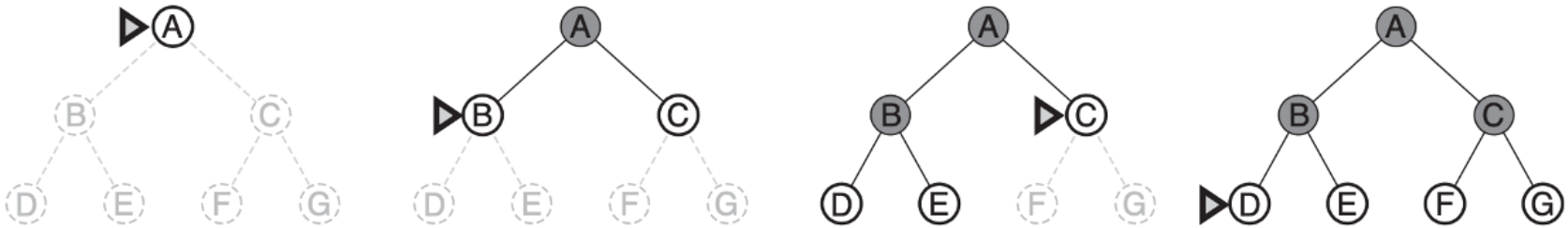
- Breadth-First search is one of the most common search strategies:
  - The root node is expanded first
  - Then, all the successors of the root node are expanded
  - Then, the successors of each of these nodes
  - Uninformed search strategies can only generate successors and distinguish a goal state from a non-goal state
- In different words, the order in which frontier nodes are expanded is level by level (a given depth of the tree)
- This is equivalent to expanding the **shallowest** unexpanded node in the frontier; equivalent to a **queue (FIFO)**

Breadth-First Search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Breadth-First Search

- Steps of the BFS algorithm:
  - **Expand** the shallowest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is added to the frontier



Measuring Performance

- We can evaluate the performance of an algorithm based on the following criteria:
  - **Completeness**, is the algorithm guaranteed to find a solution provided that one exists?
  - **Optimality**, is the strategy capable of finding the optimal solution?
  - **Time complexity**, how long does the algorithm take to find a solution?
  - **Space complexity**, how much memory is needed to perform the search?

Measuring Performance

- Generally, the size of the space graph is typically used, i.e., $|\mathcal{V}| + |\mathcal{E}|$, the size of the set of vertices set and edges, respectively
- In AI, we use an implicit representation of the graph via the initial state, actions and transition model (as the graph could be infinite)
- Therefore, the following three quantities are used:
  - **Branching factor**, the maximum number of successors of any node: b
  - **Depth** of the shallowest goal node (number of steps from the root): d
  - The **maximum length** of any path in the state space: m

Performance of BFS

- Let us evaluate the performance of BFS.
  - **Completeness**: if the goal node is at some finite depth d, then BFS is complete and will find the goal (given that b is finite)
  - **Optimality**: BFS is optimal if the path cost is a nondecreasing function of the depth of the node (e.g., all actions have the same cost)
  - **Time complexity**: $\mathcal{O}(b^d)$, assuming a uniform tree where each node has b successors, we generate $b + b^2 + \cdots + b^d = \mathcal{O}(b^d)$
  - **Space complexity**: $\mathcal{O}(b^d)$, if we store all expanded nodes, we have $\mathcal{O}(b^{d-1})$ explored nodes in memory and $\mathcal{O}(b^d)$ in the frontier

Outline

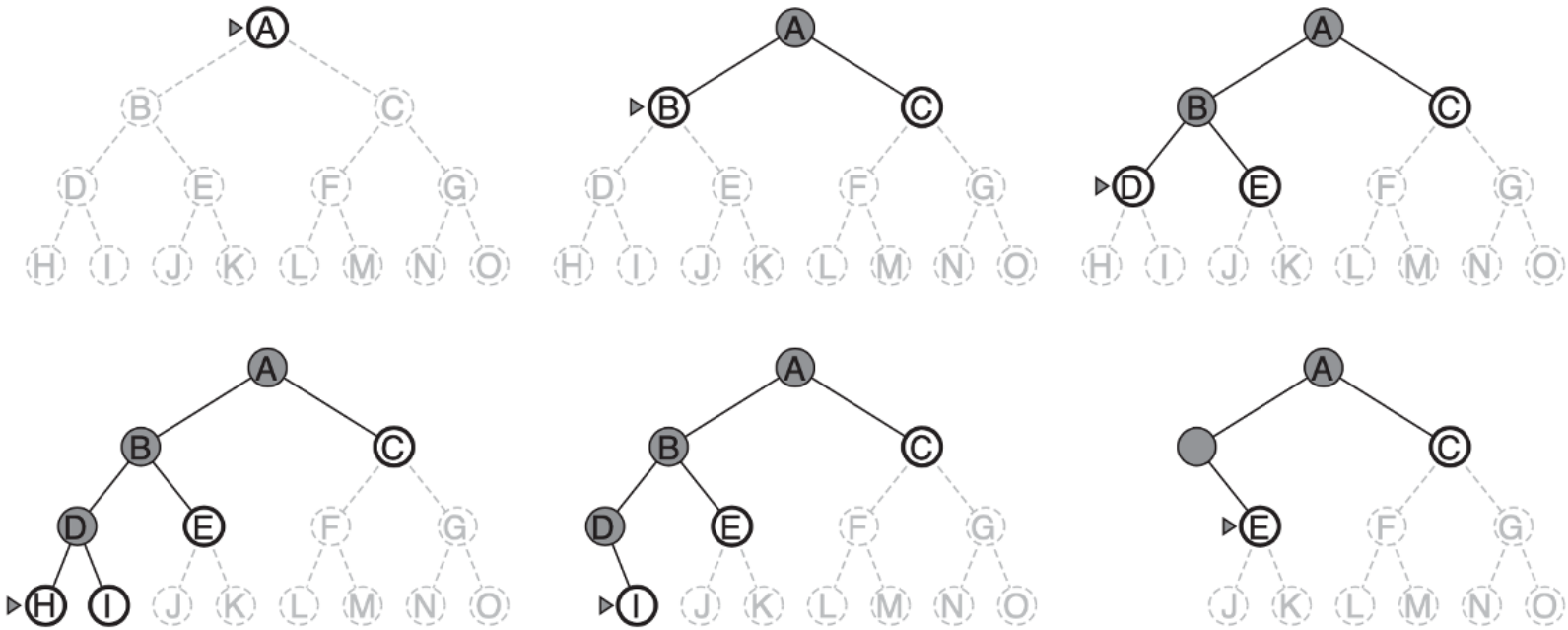
Depth-First Search

- Depth-First search is another common search strategy:
  - The root node is expanded first
  - Then, the first (or one at random) successor of the root node is expanded
  - Then, the deepest node in the current frontier is expanded
- This is equivalent to expanding the deepest unexpanded node in the frontier; a **stack (LIFO)** is used for expansion
- Basically, the most recently generated node is chosen for expansion

Depth-First Search

- Steps of the DFS algorithm:
  - **Expand** the deepest node in the frontier
  - **Do not add** children in the frontier if the node is already in the frontier or in the list of visited nodes (to avoid loopy paths)
  - **Stop** when a goal node is visited

Depth-First Search



Performance of DFS

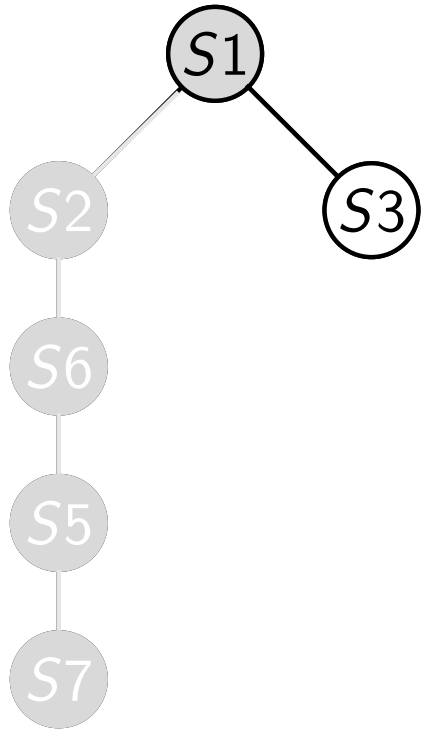
- Let us evaluate the performance of DFS.
  - **Completeness**: DFS is not complete if the search space is infinite or if we do not avoid infinite loops; it is complete if the search space is finite
  - **Optimality**: DFS is not optimal as, e.g., it can expand the left subtree when the goal node is in the first level of the right subtree
  - **Time complexity**: $\mathcal{O}(b^m)$, as it depends on the maximum length of the path in the search space (in general m can be much larger than d)
  - **Space complexity**: $\mathcal{O}(b^m)$, as we store all the nodes from each path from the root node to the leaf node

Depth-First Search - Variations

- Depth-First Search comes with several issues:
  - Not optimal
  - High time complexity
  - High space complexity
- DFS with less memory usage (saving space, reducing complexity)
- Depth-Limited Search, for completeness

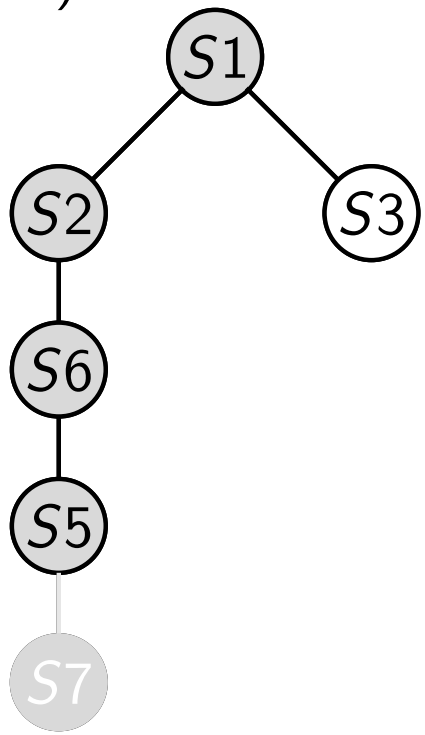
Depth-First Search with Less Memory Usage

- Let us consider the example in the image
- Node $S7$ is not a goal node and does not have children
- The next step of DFS would be to expand $S3$
- We can remove the left subtree from memory, since it has been fully explored
- This reduces the space complexity to $\mathcal{O}(bm)$
- We need to store a single path along with the siblings for each node on the path



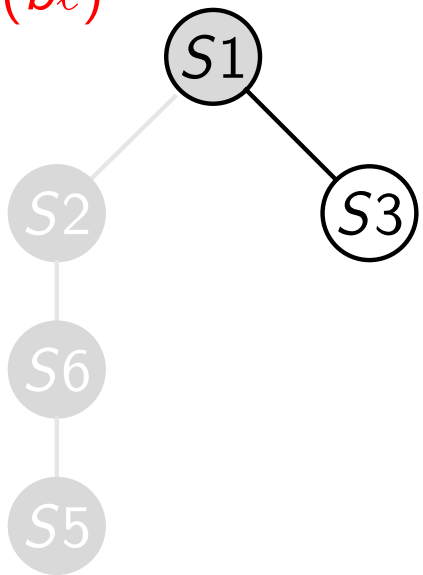
Depth-Limited Search

- In problems with infinite state spaces, DFS is not complete
- This issue can be mitigated by setting a depth limit ℓ
- This approach is called **depth-limited search** (DLS)
- For $\ell = 3$, we have
- If we choose $\ell < d$,
  we add another source of incompleteness
- DLS is not optimal even when $\ell > d$
- However, the time complexity reduces to $\mathcal{O}(b^\ell)$



DLS with Less Memory Usage

- Like before, we can remove the explored paths from memory after we have reached the depth limit ℓ
- By doing that, the space complexity reduces to $\mathcal{O}(b\ell)$



Comparing Uninformed Search Strategies

Criterion/Strategy	**BFS**	**DFS**	**DFS (LMU)**	**DLS (LMU)**
Completeness	Yes*	Yes***	Yes***	Yes, if $\ell \geq d$
Optimality	Yes**	No	No	No
Time	$\mathcal{O}(b^d)$	$\mathcal{O}(b^m)$	$\mathcal{O}(b^m)$	$\mathcal{O}(b^\ell)$
Space	$\mathcal{O}(b^d)$	$\mathcal{O}(b^m)$	$\mathcal{O}(bm)$	$\mathcal{O}(b\ell)$

* If b is finite

** If the path cost is a nondecreasing function of the depth of the node (e.g., all actions have the same cost)

*** If the search space is finite (also, loopy paths are removed)

References

Russell, A. S., and Norvig, P., *Artificial Intelligence A Modern Approach*, 4^{th} Edition. Prentice Hall.

- Chapter 3 – "Solving Problems by Searching", Section 3.1 "Problem-Solving Agents" up to and including Section 3.4.6 "Comparing Uninformed Search Algorithms"
- Appendix A – "Mathematical Background", Section A.1 "Complexity Analysis and $\mathcal{O}()$ Notation

Aims of the Session

You should now be able to:

- Understand the concept of asymptotic analysis
- Formulate a search problem
- Explain the steps involved in Breadth-First Search (BFS), Depth-First Search (DFS) and its variations
- Apply the algorithms to solve search problems

Thank you!