AI1 & AIML - Uninformed Search

Dr Leonardo Stella

Aims of the Session

This session aims to help you:

- Formulate a search problem
- Explain the steps involved in Breadth-First Search and be able to apply the algorithm to solve search problems
- Describe the concept of asymptotic analysis

Overview

- **Search Problem Formulation**
- Breadth-First Search
- Revision: Asymptotic Analysis

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

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.

Search Problem Formulation

- Many problems in AI, especially in certain contexts, e.g., games, can can be viewed as a search for a solution
- Definition: the **formulation of a search problem** is the process of formally define the search problem.

- To this end, we make the following assumptions about the environment:
  - **Observable**, i.e., the agent is able to know the current state
  - **Discrete**, i.e., there are only finitely many actions at any state
  - **Known**, i.e., the agent can determine which states are reached by which action
  - **Deterministic**, i.e., each action has exactly one outcome

Search Problem Formulation

- Under these assumptions, the solution to any problem is a fixed sequence of actions. More formally, the solution to a search problem is the sequence of actions from the initial state to the goal state

- The agent’s task is to find out how to act, now and in the future, in order to reach a goal state: namely to determine a sequence of actions

- The process of looking for a sequence of actions is called **search**

Search Problem Formulation

- Formally, a search problem is defined by the following five components:
  - **Initial state**, i.e., the state that the agent starts in
  - **Actions**, i.e., the set describing the actions that can be executed in any state *s*
  - **Transition model**, i.e., the states resulting from executing each action *a* from every state *s* (a description of what each action does)
  - **Goal test** to determine if a state is a goal state
  - **Path cost** function that assigns a value (cost) to each path

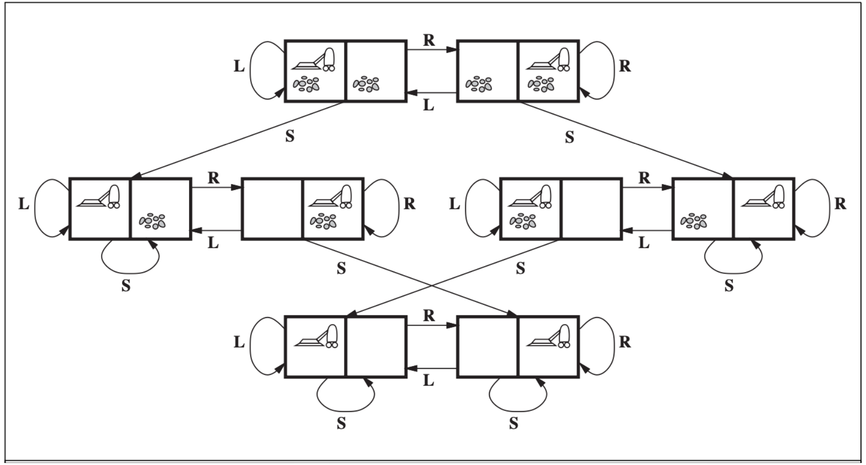
- The first three components together define the **state space** of the problem, in the form of a directed graph or network. A **path** in the state space is a sequence of states connected by a sequence of actions

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

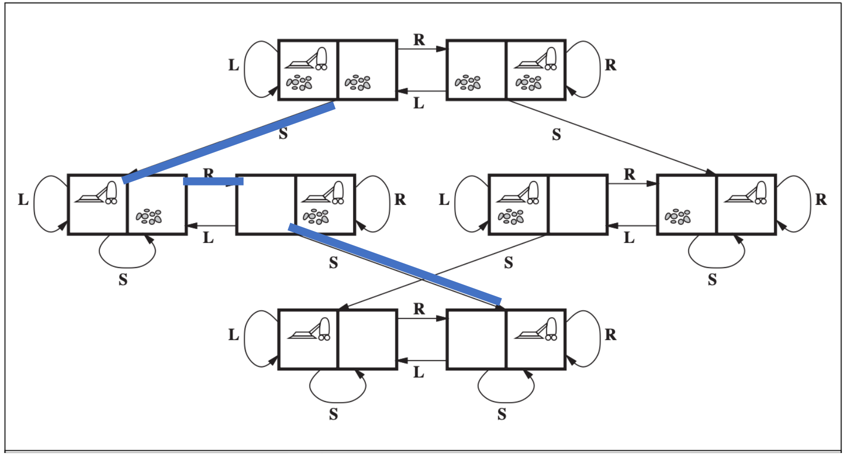
- On the contrary, the state space graph for the vacuum world example has a small number of states

- For example, the state space graph for chess would be very large



Notation

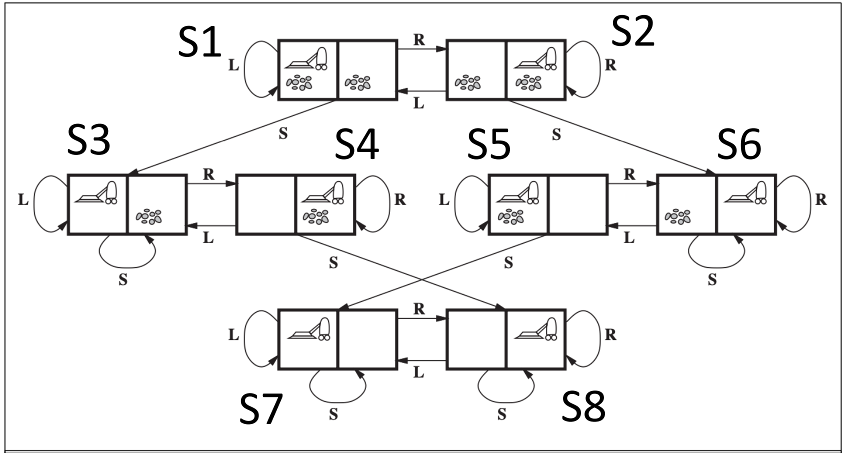
- A solution can be seen as a path in the state space graph



Notation

- A solution can be seen as a path in the state space graph

- Each state corresponds to a node in the state space graph



Summary

- A **problem-solving agent** is an agent that is able to search for a solution in a given problem (provided that one exists)

- **Search problem formulation** is the process of formally define a search problem, through five components: initial state, actions, transition model, goal test and path cost

Overview

- Search Problem Formulation
- **Breadth-First Search**
- Revision: Asymptotic Analysis

Searching for Solutions

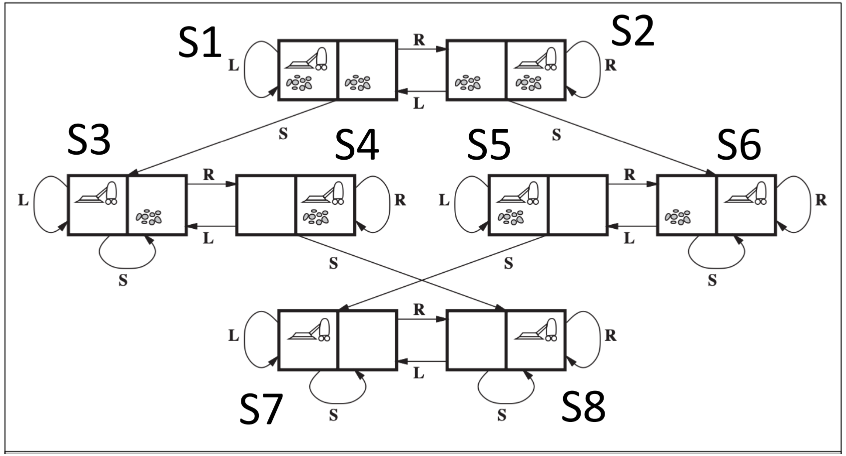
- A solution is an action sequence from an initial state to a goal state

- Possible action sequences form a **search tree** with initial state at the root; actions are the branches and nodes correspond to the state space

- The idea is to expand the current state by applying each possible action: this generates a new set of states

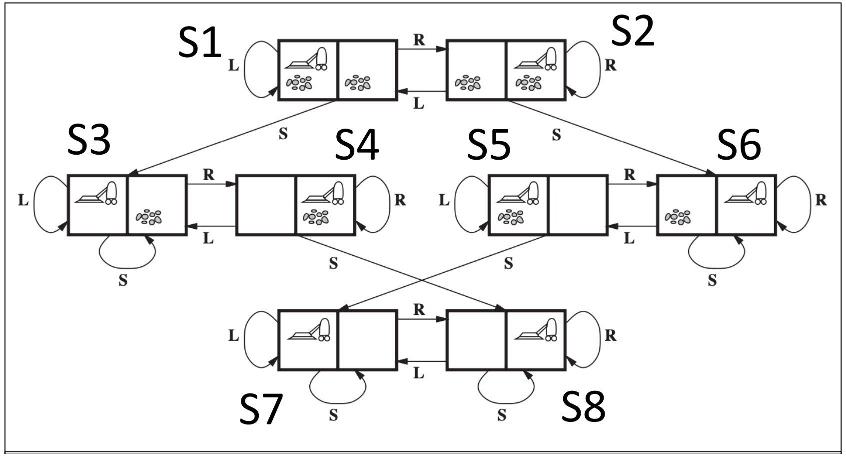
Searching for Solutions

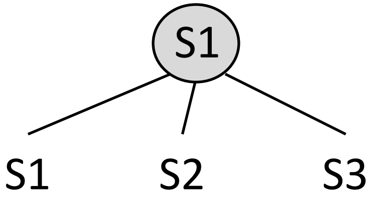
- Let us consider the example from before



Searching for Solutions

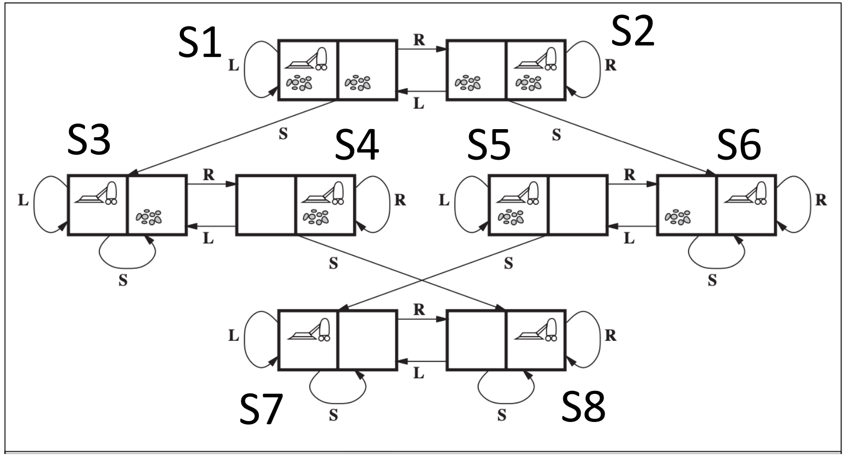
- Let us consider the example from before
- If S1 is the initial state and {S7, S8} is the set of goal states, the corresponding search tree after expanding the initial state is:

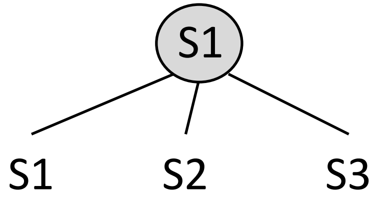




Searching for Solutions

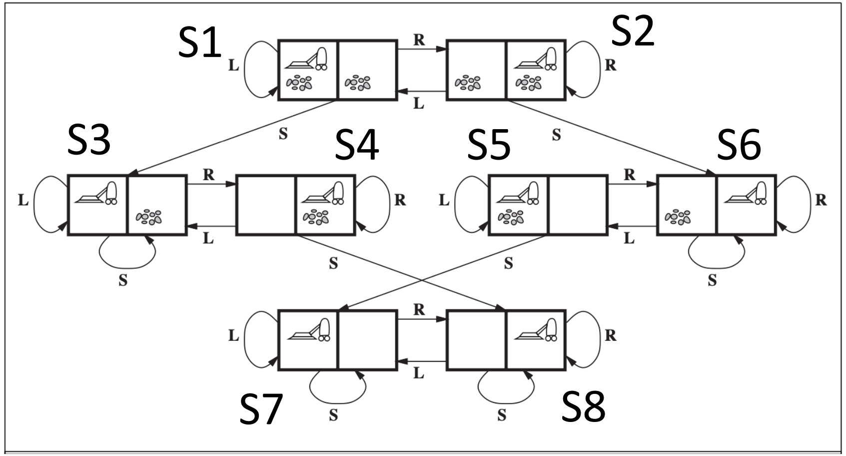
- Each of the three nodes resulting from the first expansion is a **leaf node**
- Definition: the set of all leaf nodes available for expansion at any given time is called the **frontier** (also sometimes called the **open list**).

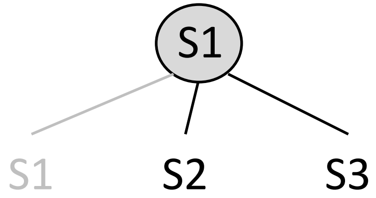




Searching for Solutions

- Each of the three nodes 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** (also sometimes called the **open list**).
- The path from S1 to S1 is a **loopy path** (or repeated state) and in general is not considered:





Uninformed Search Strategies

- Definition: **Uninformed Search** (also called **blind search**) is defined as the set of strategies having no additional information about states 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

- Using different words, the frontier nodes that are expanded belong to a given depth of the tree

- This is equivalent to expanding the **shallowest** unexpanded node in the frontier; we use a **queue** (**FIFO**) for expansion

Breadth-First Search

- Breadth-First Search algorithm (see Fig. 3.11):

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure

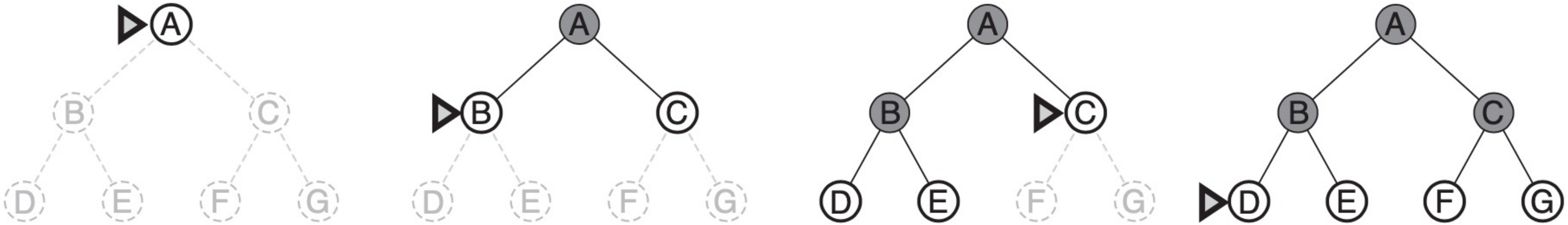
```
node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ← a FIFO queue with node as the only element
explored ← an empty set
loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)   /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
            if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
            frontier ← INSERT(child, frontier)
```

Breadth-First Search

- Breadth-First Search steps:
  - **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

Breadth-First Search

- Breadth-First Search steps (see Fig. 3.12):
  - **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:

- **Completeness**, i.e., whether the algorithm is guaranteed to find a solution provided that one exists
- **Optimality**, i.e., whether the strategy is able to find the optimal solution
- **Time complexity**, i.e., the time the algorithm takes to find a solution
- **Space complexity**, i.e., the memory needed to perform the search

Measuring Performance

We can evaluate the performance of an algorithm based on the following:

- **Completeness**, i.e., whether the algorithm is guaranteed to find a solution provided that one exists
- **Optimality**, i.e., whether the strategy is able to find the optimal solution
- **Time complexity**, i.e., the time the algorithm takes to find a solution
- **Space complexity**, i.e., the memory needed to perform the search

- To measure the performance, the size of the space graph is typically used, i.e., $|\mathcal{V}| + |\mathcal{E}|$, the set of vertices and set of edges, respectively

Measuring Performance

- 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 each 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

Summary

- Breadth-First Search is a search algorithm that expands the nodes in the frontier starting from the shallowest, similar to a queue (FIFO)

- This algorithm is complete (for finite b), optimal (if the path cost is nondecreasing), but has high time $O(b^d)$ and space complexity $O(b^d)$

Overview

- Search Problem Formulation
- Breadth-First Search
- **Revision: Asymptotic Analysis**

Asymptotic Analysis

- Computer scientists are often asked to determine the quality of an algorithm by comparing it with other ones and measure the speed and memory required
- **Benchmarking** is one approach:
  - We run the algorithms and we measure speed (in seconds) and memory consumption (in bytes)
  - Problem: this approach measures the performance of a specific program written in a particular language, on a given computer, with particular input data
- **Asymptotic analysis** is the second approach:
  - It is a mathematical abstraction over both the exact number of operations (by ignoring constant factors) and exact content of the input (by considering the size of the input, only)
  - It is independent of the particular implementation and input

Asymptotic Analysis

- The first step in the analysis is to abstract over the input. In practice, we characterise the size of the input, which we denote by n
- The second step is to abstract over the implementation. The idea is to find some measure that reflects the running time of the algorithm
- For asymptotic analysis, we typically use 3 notations:
  - Big O notation: $O(\cdot)$
  - Big Omega notation: $\Omega(\cdot)$
  - Big Theta notation: $\Theta(\cdot)$

Asymptotic Analysis: Big O

- We say that $f(n) \in O(g(n))$ when the following condition holds:

$$\exists k > 0\ \exists n_0 \forall n > n_0 \colon |f(n)| \leq k \cdot g(n)$$

- The above reads: “There exists a positive constant k, and a (positive) value n_0 such that for all $n > n_0$, $|f(n)| \leq k \cdot g(n)$”
- In simple terms, this is equivalent to saying that $|f|$ is bounded from above by a function g (up to a constant factor) asymptotically

Asymptotic Analysis: Big Theta and Big Omega

- We say that $f(n) \in \Omega(g(n))$ when the following condition holds:

$$\exists k > 0 \; \exists n_0 \forall n > n_0 \colon |f(n)| \geq k \cdot g(n)$$

- This is equivalent to saying that f is bounded from below by g asymptotically
- We say that $f(n) \in \Theta(g(n))$ when the following condition holds:

$$\exists k_1, k_2 > 0 \; \exists n_0 \forall n > n_0 \colon k_1 \cdot g(n) \leq |f(n)| \leq k_2 \cdot g(n)$$

- Or f is bounded both from above and from below by g asymptotically

Asymptotic Analysis: Example

- Consider the following algorithm (pseudocode):

function SUMMATION(*sequence*) **returns** a number
    sum ← 0
    for $i = 1$ **to** LENGTH(*sequence*) **do**
        sum ← *sum* + *sequence*[i]
    return *sum*

- Step 1: abstract over input, e.g., the length of the *sequence*
- Step 2: abstract over the implementation, e.g., total number of steps. If we call this characterisation $T(n)$ and we count lines of code, we have $T(n) = 2n + 2$

Asymptotic Analysis: Example

- Consider the following algorithm (pseudocode):

function SUMMATION(*sequence*) **returns** a number
  sum ← 0
  for $i = 1$ **to** LENGTH(*sequence*) **do**
    sum ← *sum* + *sequence*[i]
  return *sum*

- We say that the SUMMATION algorithm is $O(n)$, meaning that its measure is at most of constant times n with few possible exceptions
- $T(n) \in O(f(n))$ if $T(n) \leq k \cdot f(n)$ for some k, for all $n > n_0$
- For $T(n) = 2n + 2$, an example would be: $k = 3, n_0 = 2$

Summary

- Asymptotic analysis is a powerful tool to describe the speed and memory consumption of an algorithm
- It is useful as it is independent of a particular implementation and input
- It is an approximation as the input n approaches infinity and over the number of steps required
- Convenient to compare algorithms, e.g., an algorithm with time complexity of $O(n)$ runs faster than one with $O(n^2)$
- Other notations exist, such as $\Omega(n)$ and $\Theta(n)$

Aims of the Session

You should now be able to:

- Formulate a search problem
- Explain the steps involved in Breadth-First Search and be able to apply the algorithm to solve search problems
- Describe the concept of asymptotic analysis

References

- Russell, A. S., and Norvig, P. (2010), *Artificial Intelligence A Modern Approach*, 3rd Edition. Prentice Hall.
  - Chapter 3 – Solving Problems by Searching (Section 3.1 up to 3.2.1, Section 3.3 up to 3.4.1)
  - Appendix A – Mathematical Background (Section A.1)