- Artificial Intelligence → Problem Solving (one of the aspect)
- Given – Desired
- Decision ?--- 1. knowledge based , 2. search
- Knowledge based –1. Memory Based , 2. Rule based
- Search – 1.

Module–II:
Problem Solving

Dr.K.Babu Rao,
Professor in CSE,
SoET, CMRU.

- The simplest reflex agents, which base their actions on a direct mapping from states to actions.
- Such agents cannot operate well in environments for which this mapping would be too large to store and would take too long to learn.
- Goal –based agents, on the other hand, consider future actions and the desirability of their outcomes.
- Another name of Goal-based is called a **problem-solving agent**.
- Problem-solving agents use **atomic** representations. i.e. states of the world are considered as wholes, with no internal structure visible to the problem-solving algorithms.
- Goal-based agents that use more advanced **factored** or **structured** representations are usually call **planning agents.**

- problem solving begins with precise definitions of **problems** and their **solutions** and give several examples to illustrate these definitions.
- Then describe several general-purpose search algorithms that can be used to solve these problems.
- **uninformed** search algorithms—algorithms that are given no information about the problem other than its definition.
- **Informed** search algorithms, on the other hand, can do quite well given some guidance on where to look for solutions.
- the simplest kind of task environment, for which the solution to a problem is always a fixed sequence of actions.
- The more general case—where the agent's future actions may vary depending on future percepts.

Problem –Solving Agents

- Intelligent agents are supposed to maximize their performance measure.
- Before an agent can start searching for solutions, a **goal** must be identified and a well-defined problem must be formulated.
- A general **TREE-SEARCH** algorithm considers all possible paths to find a solution, whereas a **GRAPH-SEARCH** algorithm avoids consideration of redundant paths.
- **Goal formulation**, based on the current situation and the agent's performance measure, is the first step in problem solving.
- **Problem formulation** is the process of deciding what actions and states to consider, given a goal.

Search framework

- State space search
  - Uninformed/Blind search
  - Informed / Heuristic search
- Problem reduction search (i.e. decomposed)
- Game tree search
- Advances
  - Memory bounded search
  - Multi-objective search
  - Learning how to search

State space search

- Basic search problem:
  - Given: [S, s, O, G] where
    - S is the (implicitly specified) set of states
    - s is the start state
    - O is the set of state transition operators
    - G is the set of goal states
  - To find a sequence of state transitions leading from s to a goal state.

Well-defined problems and solutions

- A **problem** can be defined formally by components:
  - The **initial state** that the agent starts in. For example, the initial state for our agent in Romania might be described as In(Arad).
  - A description of the possible **actions** available to the agent. Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s.
  - A description of what each action does; the formal name for this is the **transition model,** specified by a function RESULT(s, a) that returns the state that results from doing action a in state s.
  - **The goal test**, which determines whether a given state is a goal state. Sometimes there is an explicit set of possible goal states, and the test simply checks whether the given state is one of them.
  - A **path cost** function that assigns a numeric cost to each path

- A problem consists of five parts: the **initial state**, a set of **actions**, a **transition model** describing the results of those actions, a **goal test** function, and a **path cost** function.
- Together, the initial state, actions, and transition model implicitly define the **state space** of the problem—the set of all states reachable from the initial state by any sequence of actions.
- The state space forms a directed network or **graph** in which the nodes are states and the links between nodes are actions.
- A **path** in the state space is a sequence of states connected by a sequence of actions.
- The environment of the problem is represented by a **state space**. A **path** through the state space from the initial state to a goal state is a solution

Example Problems

- The problem-solving approach has been applied to a vast array of task environments.
- A **toy problem** is intended to illustrate or exercise various problem-solving methods. It can be given a concise, exact description and hence is usable by different researchers to compare the performance of algorithms.
- A **real-world problem** is one whose solutions people actually care about. Such problems tend not to have a single agreed-upon description, but we can give the general flavor of their formulations.

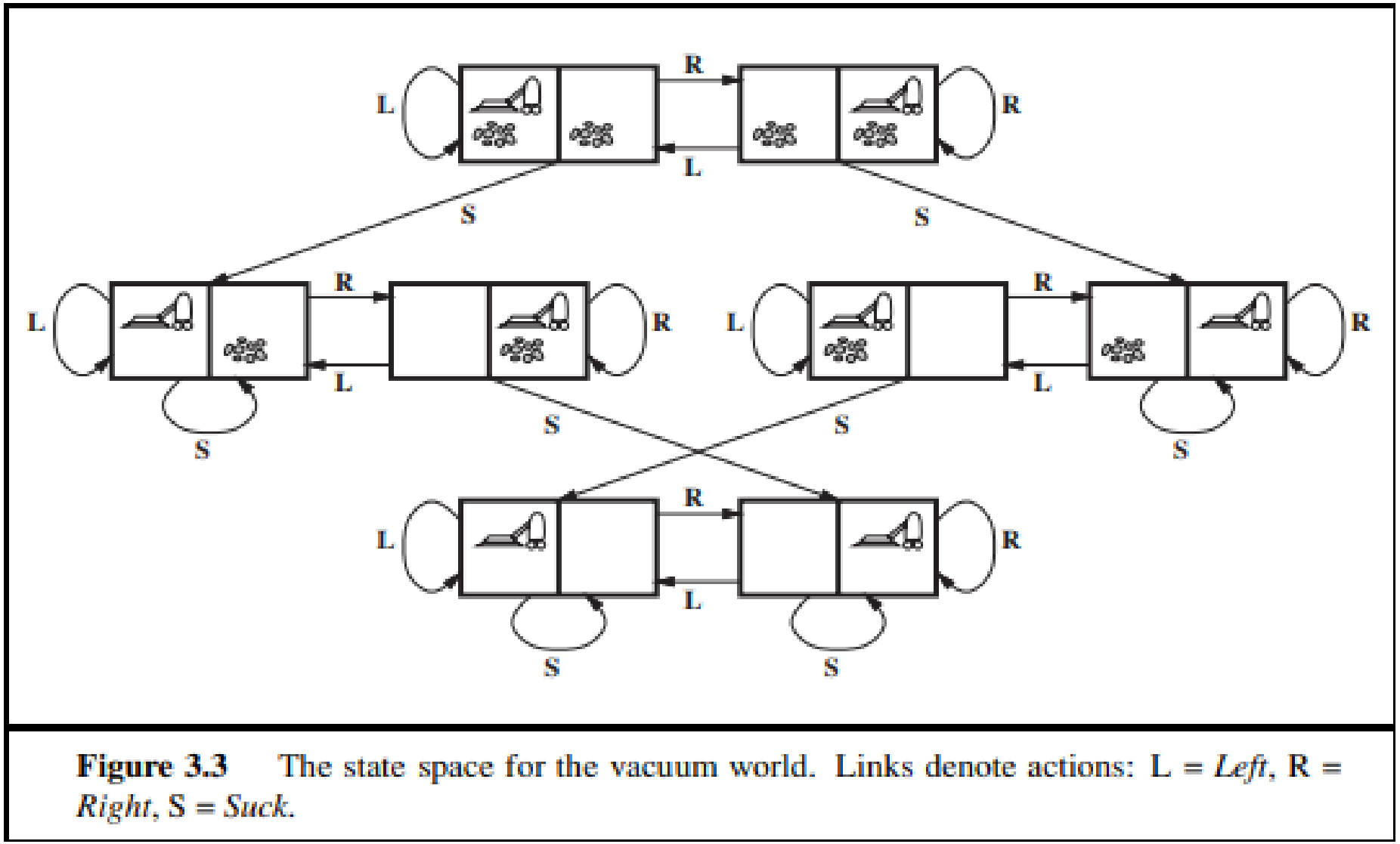

Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

3.2.1 Toy problems

The first example we examine is the **vacuum world** first introduced in Chapter 2. (See Figure 2.2.) This can be formulated as a problem as follows:

- **States**: The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus, there are $2 \times 2^2 = 8$ possible world states. A larger environment with n locations has $n \cdot 2^n$ states.
- **Initial state**: Any state can be designated as the initial state.
- **Actions**: In this simple environment, each state has just three actions: *Left*, *Right*, and *Suck*. Larger environments might also include *Up* and *Down*.
- **Transition model**: The actions have their expected effects, except that moving *Left* in the leftmost square, moving *Right* in the rightmost square, and *Sucking* in a clean square have no effect. The complete state space is shown in Figure 3.3.
- **Goal test**: This checks whether all the squares are clean.
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

8 puzzle problem

- State description (S)
  - Location of each of the eight tiles (and the blank)
- Start state (s)
  - The starting configuration (given)
- Operators (O)
  - Four operators, for moving the blank left, right, up or down
- Goals (G)
  - One or more goal configurations (given)

8-puzzle

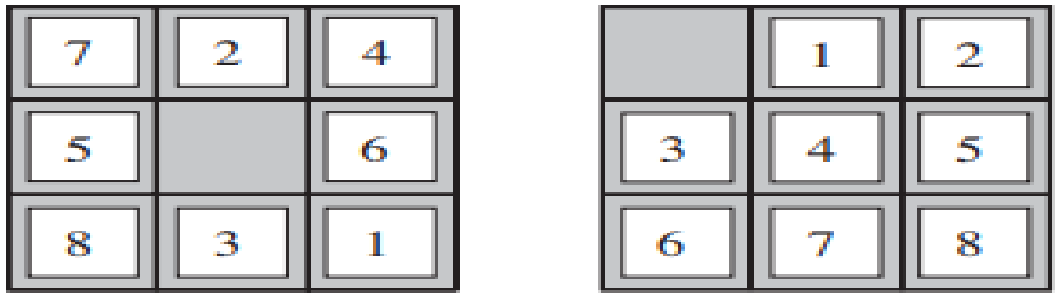

Figure 3.4 A typical instance of the 8-puzzle.

- **States**: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- **Initial state**: Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states (Exercise 3.4).
- **Actions**: The simplest formulation defines the actions as movements of the blank space *Left*, *Right*, *Up*, or *Down*. Different subsets of these are possible depending on where the blank is.
- **Transition model**: Given a state and action, this returns the resulting state; for example, if we apply *Left* to the start state in Figure 3.4, the resulting state has the 5 and the blank switched.
- **Goal test**: This checks whether the state matches the goal configuration shown in Figure 3.4. (Other goal configurations are possible.)
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

8-queens problem

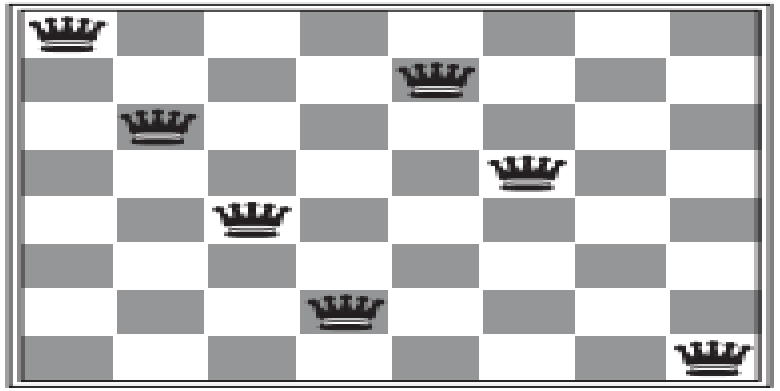

Figure 3.5 Almost a solution to the 8-queens problem. (Solution is left as an exercise.)

Although efficient special-purpose algorithms exist for this problem and for the whole n-queens family, it remains a useful test problem for search algorithms. There are two main kinds of formulation. An **incremental formulation** involves operators that *augment* the state description, starting with an empty state; for the 8-queens problem, this means that each action adds a queen to the state. A **complete-state formulation** starts with all 8 queens on the board and moves them around. In either case, the path cost is of no interest because only the final state counts. The first incremental formulation one might try is the following:

- **States**: Any arrangement of 0 to 8 queens on the board is a state.
- **Initial state**: No queens on the board.
- **Actions**: Add a queen to any empty square.
- **Transition model**: Returns the board with a queen added to the specified square.
- **Goal test**: 8 queens are on the board, none attacked.

In this formulation, we have $64 \cdot 63 \cdots 57 \approx 1.8 \times 10^{14}$ possible sequences to investigate. A better formulation would prohibit placing a queen in any square that is already attacked:

- **States**: All possible arrangements of n queens ($0 \leq n \leq 8$), one per column in the leftmost n columns, with no queen attacking another.
- **Actions**: Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.

8-queens problem

- Placing 8 queens on a chess board, so that none attacks the other
- Formulation – 1
  - A state is any arrangement of 0 to 8 queens on board
  - Operators add a queen to any square
- Formulation – 2
  - A state is any arrangement of 0-8 queens with none attacked
  - Operators place a queen in the left-most empty column

- Formulation – 3
  - A state is any arrangement of 8 queens, one in each column
  - Operators move an attacked queen to another square in the same column
- Formulation – 4

Real-world problems – Route finding problem

- **States**: Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and their status as domestic or international, the state must record extra information about these "historical" aspects.
- **Initial state**: This is specified by the user's query.
- **Actions**: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
- **Transition model**: The state resulting from taking a flight will have the flight's destination as the current location and the flight's arrival time as the current time.
- **Goal test**: Are we at the final destination specified by the user?
- **Path cost**: This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

Searching for Solutions

- A solution is an action sequence, so search algorithms work by considering various possible action sequences.
- The possible action sequences starting at the initial state form a **search tree** with the initial state at the root; the branches are actions and the **nodes** correspond to states in the state space of the problem.

Measuring problem-solving performance

- Evaluate an algorithm's performance in four ways:
  - **Completeness:** Is the algorithm guaranteed to find a solution when there is one?
  - **Optimality:** Does the strategy find the optimal solution
  - **Time complexity:** How long does it take to find a solution?
  - **Space complexity:** How much memory is needed to perform the search?

Uninformed Search or Blind Search Strategies

- **uninformed search** (also called **blind search**).
- The term means that the strategies have no additional information about states beyond that provided in the problem definition.
- All they can do is generate successors and distinguish a goal state from a non-goal state.
- All search strategies are distinguished by the order in which nodes are expanded.
- Strategies that know whether one non-goal state is "more promising" than another are called **informed search or heuristic** search strategies; they are covered in Section 3.5.

- • Uninformed search methods have access only to the problem definition. The basic algorithms are as follows:
  - **Breadth-first search** expands the shallowest nodes first; it is complete, optimal for unit step costs, but has exponential space complexity.
  - **Uniform-cost search** expands the node with lowest path cost, g(n), and is optimal for general step costs.
  - **Depth-first search** expands the deepest unexpanded node first. It is neither complete nor optimal, but has linear space complexity. Depth-limited search adds a depth bound.
  - **Iterative deepening search** calls depth-first search with increasing depth limits until a goal is found. It is complete, optimal for unit step costs, has time complexity comparable to breadth-first search, and has linear space complexity.
  - **Bidirectional search** can enormously reduce time complexity, but it is not always applicable and may require too much space.

Informed (Heuristic) Search Strategies

- Informed search strategy – one that uses problem-specific knowledge beyond the definition of the problem itself – can find solutions more efficiently that can an uniformed strategy.
- The general approach we consider is called best-first search. Best-first search is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which node is selected for expansion based on an evaluation function, f(n).

Heuristic Function