Artificial Intelligence
CS-401

Chapter # 03

Solving Problems by Searching

Dr. Hafeez Ur Rehman

(Email: hafeez.urrehman@nu.edu.pk)

Chapter's Outline

- Problem Solving Agents
- Well Defined Problems and Solutions
- Example Problems
- Searching for Solutions
  - Infrastructure of Search Algorithms
  - Performance Metric
- Uninformed Search Strategies
- Informed (Heuristic) Search Strategies

Recall: Types of agents

Reflex agent

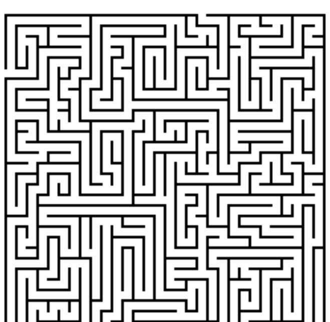
- Consider how the world IS
- Choose action based on current percept
- Do not consider the future consequences of actions

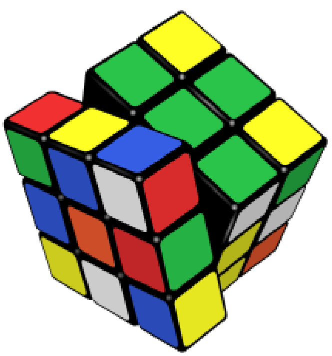
Planning agent

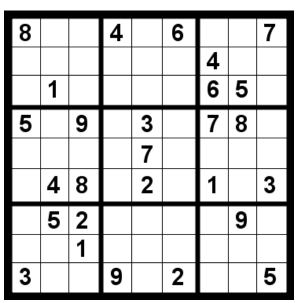
- Consider how the world WOULD BE
- Decisions based on (hypothesized) consequences of actions
- Must have a model of how the world evolves in response to actions
- Must formulate a goal

Initial Simplifying Assumptions

- Environment is **static**
  - no changes in environment while problem is being solved
- Environment is **observable**
- Environment and actions **are discrete**
  - (typically assumed, but we will see some exceptions)
- Environment is **deterministic**
- Environment is **known**

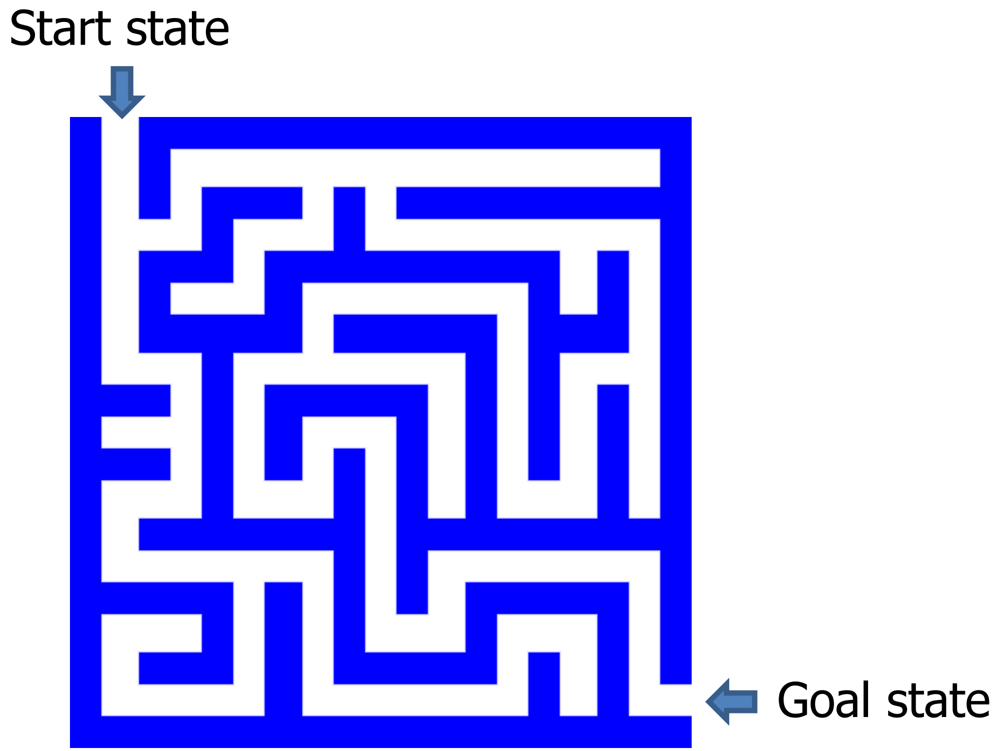






Search

- We will consider the problem of designing **goal-based agents** in **fully observable, deterministic, discrete, known** environments



Search

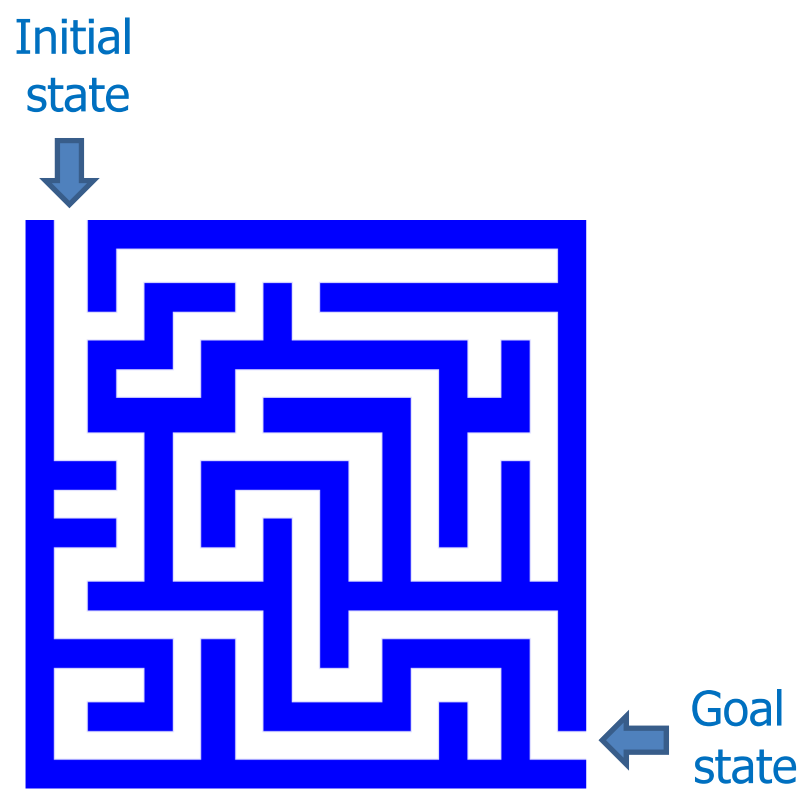
- We will consider the problem of designing **goal-based agents** in **fully observable, deterministic, discrete, known** environments
  - The agent must find a *sequence of actions* that reaches the goal
  - The **performance measure** is defined by
    - A. Reaching the goal and
    - B. How "**expensive**" the path to the goal is
  - We are focused on the process of finding the solution; while executing the solution, we assume that the agent can safely ignore (through abstraction) its percepts.

Abstraction

- **Definition**:
  Process of removing irrelevant detail to create an abstract representation: ``high-level”, ignores irrelevant details
- **Navigation Example: how do we define states and operators?**
  - First step is to abstract “the big picture”
    - i.e., solve a map problem
    - nodes = cities, links = freeways/roads (a high-level description)
    - this description is an abstraction of the real problem
  - Can later worry about details like scenery, refueling, etc
- **Abstraction is critical for automated problem solving**
  - must create an approximate, simplified, model of the world for the computer to deal with: real-world is too detailed to model exactly
  - **good abstractions retain all important details**

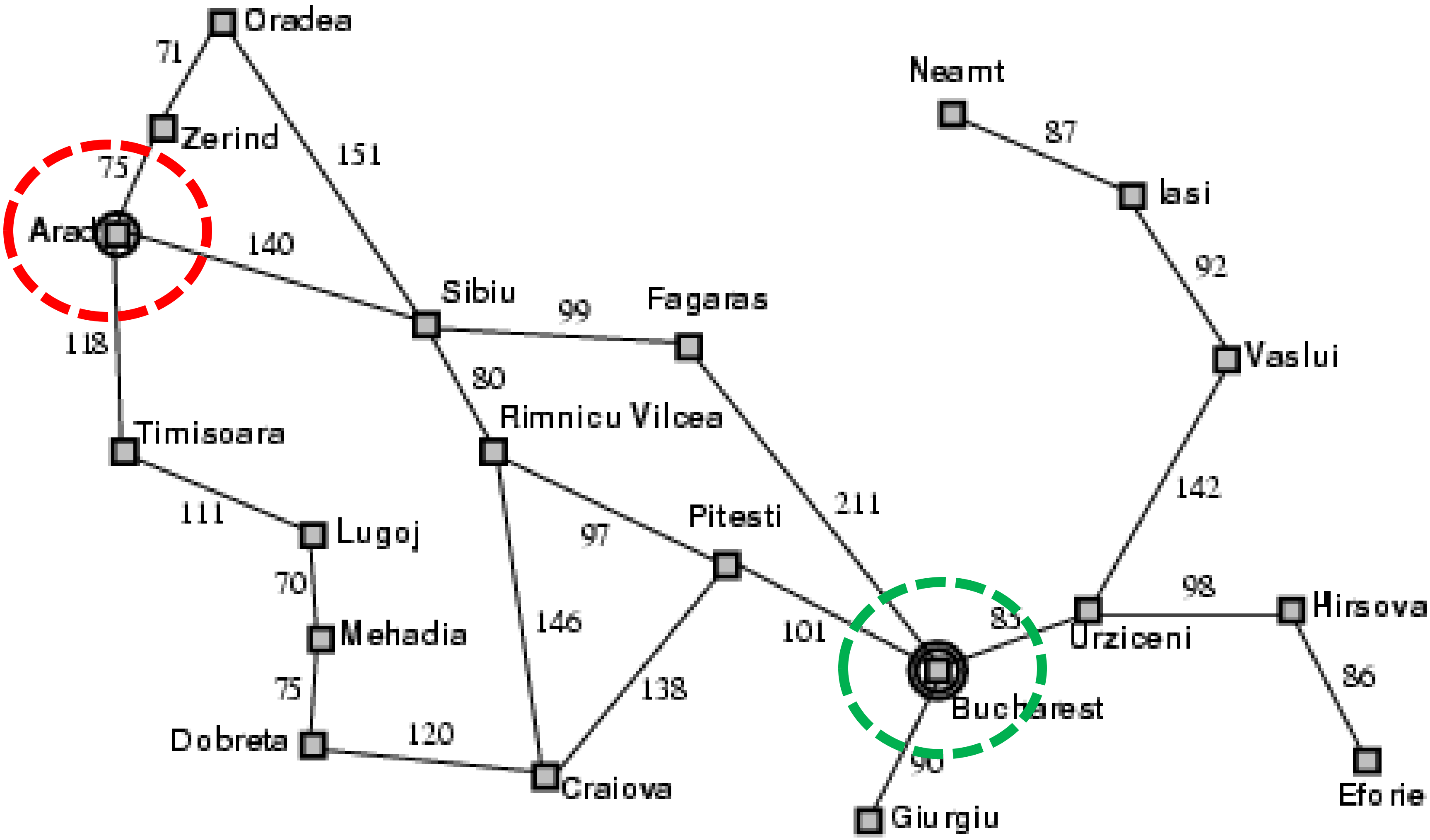
Search problem components

- **Initial state**
- **Actions**
- **Transition model**
  - What state results from performing a given action in a given state?
- **Goal state**
- **Path cost**
  - Assume that it is a sum of nonnegative *step costs*



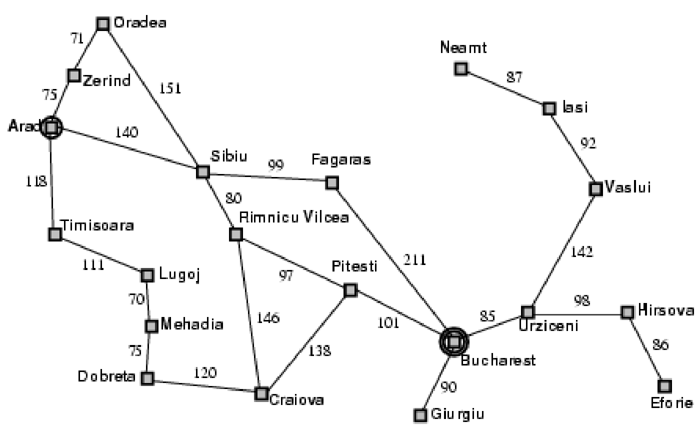
- The **optimal solution** is the sequence of actions that gives the *lowest* path cost for reaching the goal

Example: Traveling in Romania



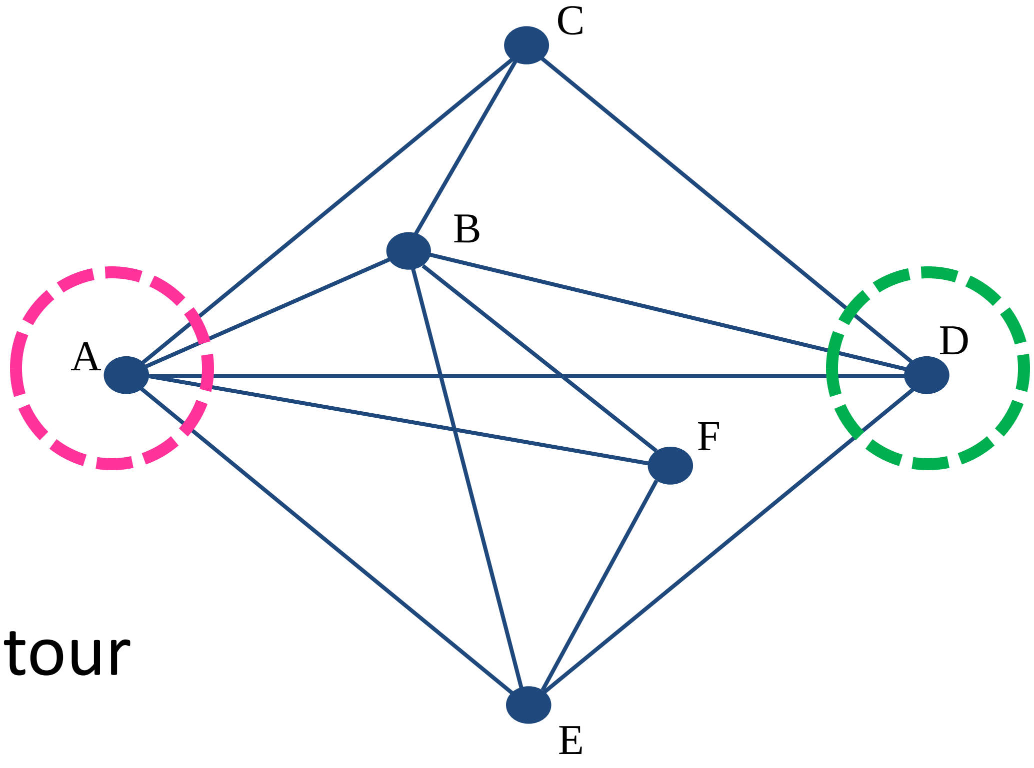
State space

- The initial state, actions, and transition model define the **state space** of the problem
  - The set of all reachable states from an initial state by any sequence of actions is called the state space
  - Can be represented as a **directed graph** where the nodes are states and links between nodes are actions
- What is the state space for the Romania problem?



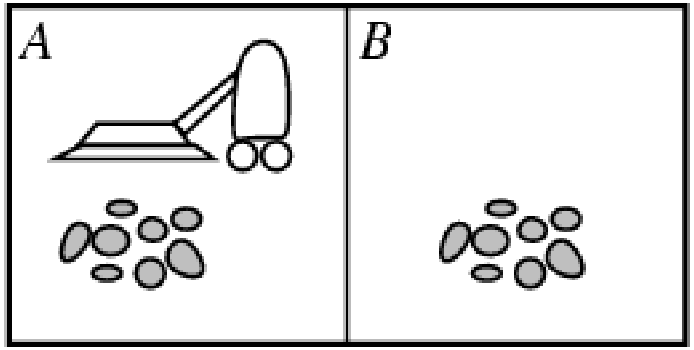
Example 1 : Traveling Salesperson Problem

- Find the shortest tour that visits all cities without visiting any city twice and return to starting point.
- State: sequence of cities visited
- $S_0 = A$



- G = a complete tour

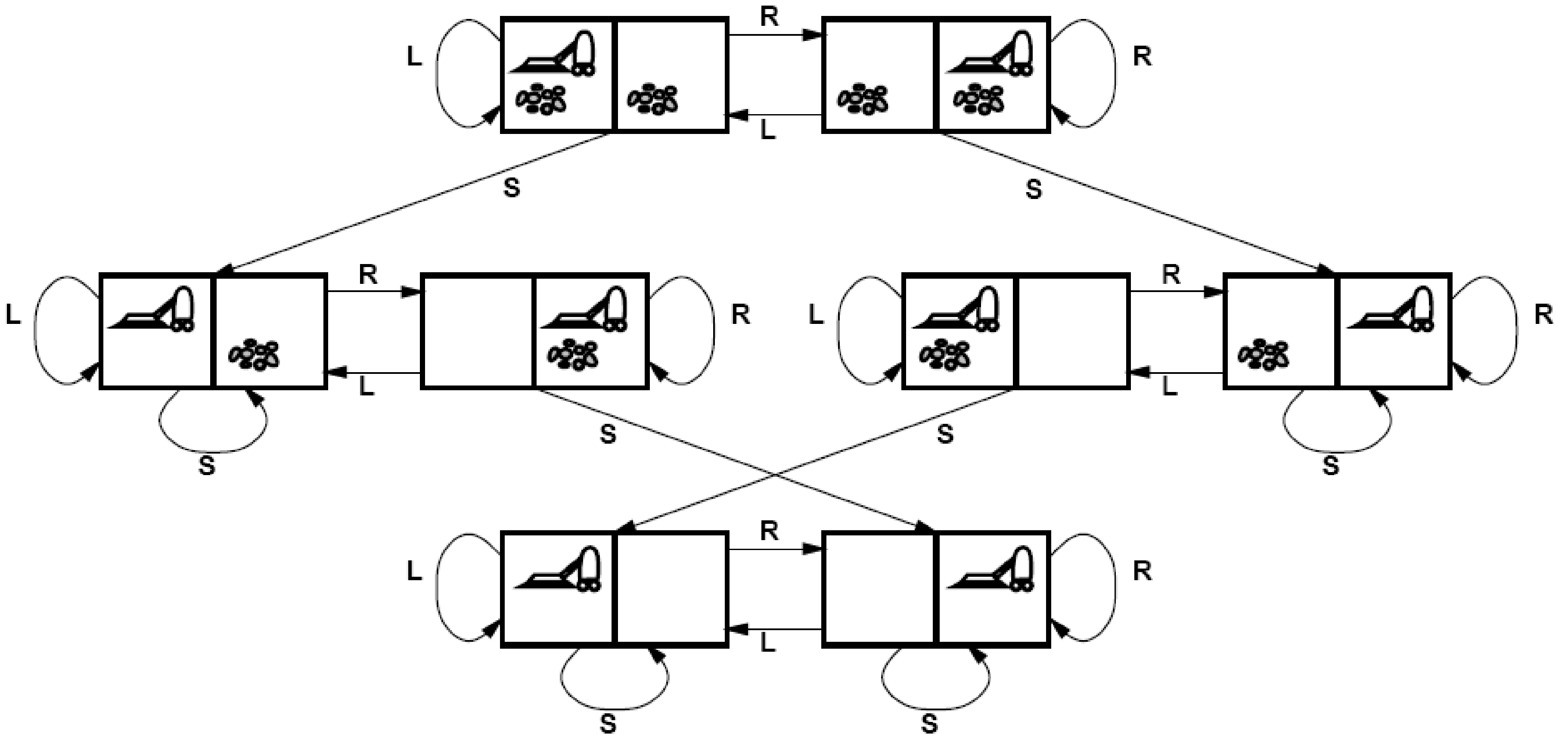
Example 2: Vacuum world



- **States**
  - Agent location and dirt location
  - How many possible states?
  - What if there are *n* possible locations?
    - The size of the state space grows exponentially with the "size" of the world!
- **Actions**
  - Left, right, suck

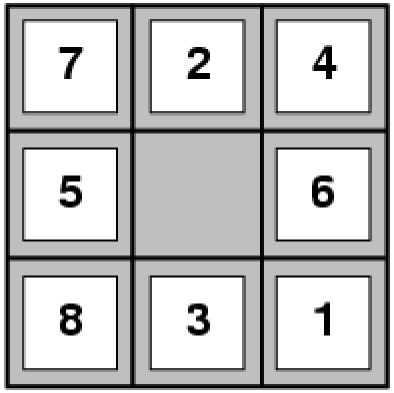
Vacuum world state space graph

- **Transition model**

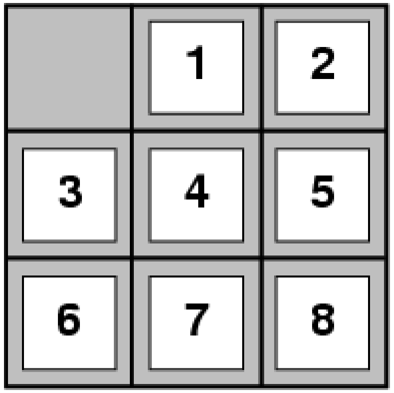


Example 3: The 8-puzzle

- **States**
  - Locations of tiles
    - 8-puzzle: 181,440 states (9!/2)
    - 15-puzzle: ~10 trillion states
    - 24-puzzle: ~10^{25} states
- **Actions**
  - Move blank left, right, up, down
- **Path cost**
  - 1 per move



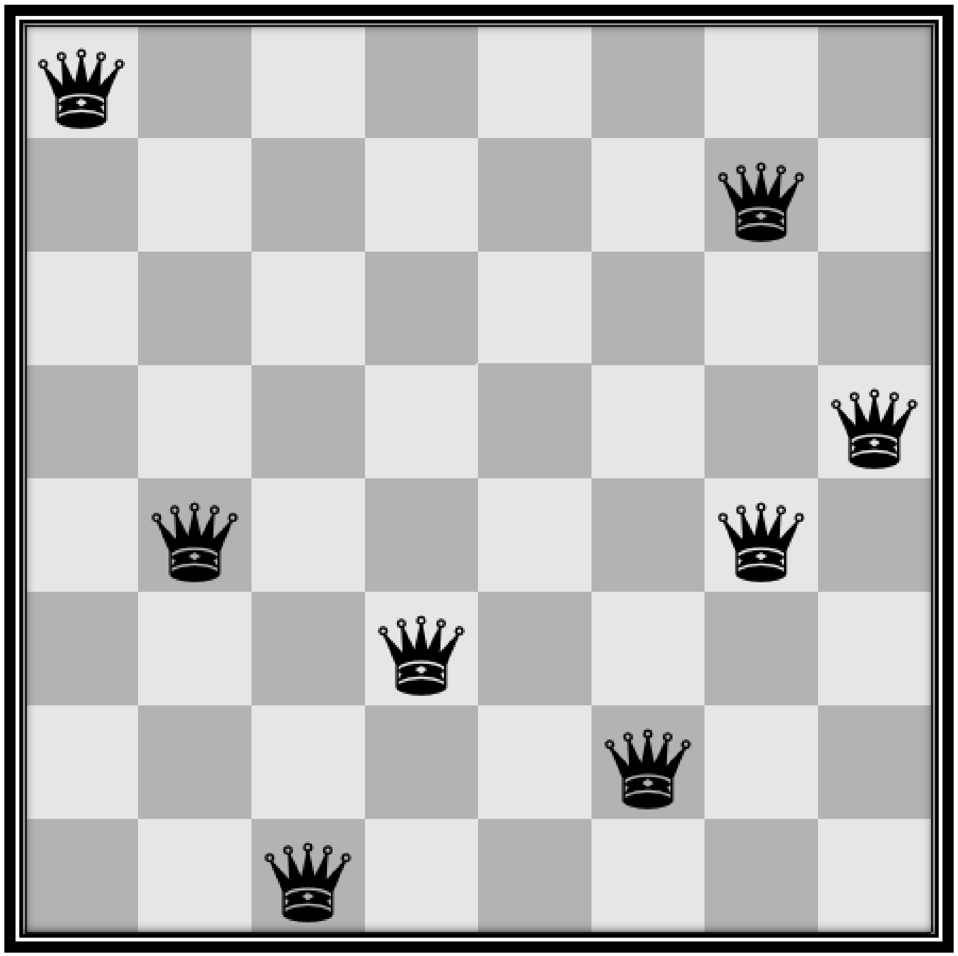
Start State



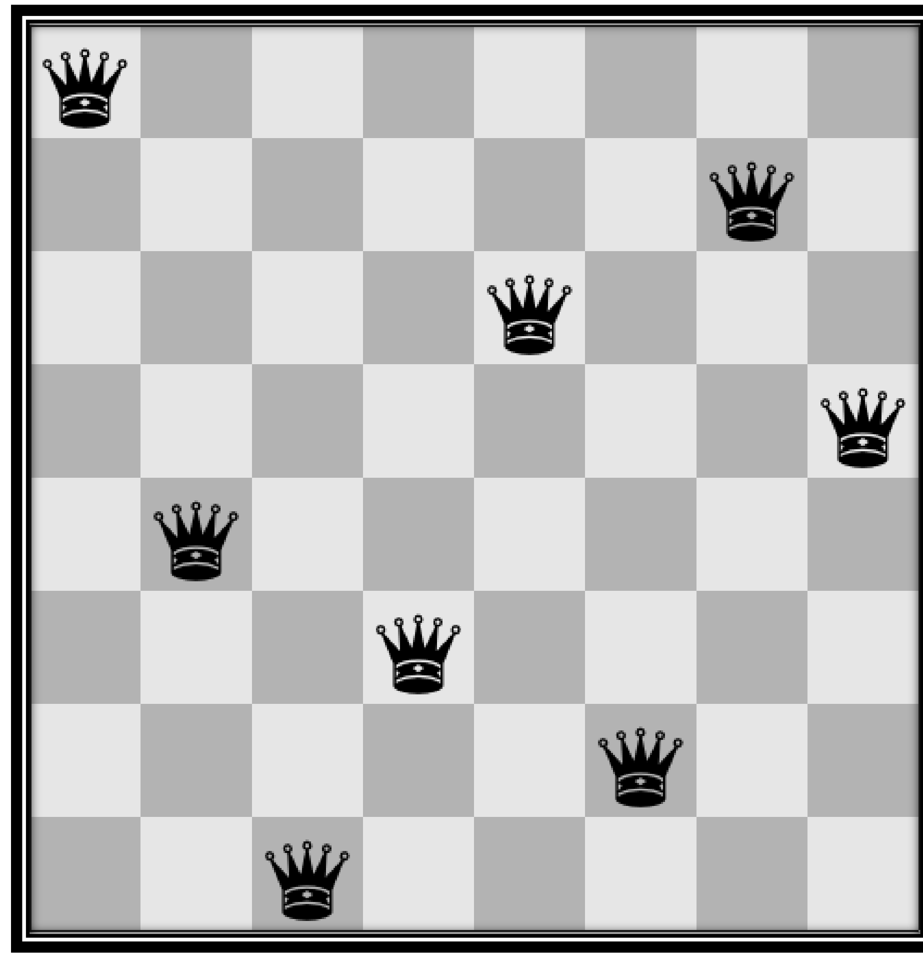
Goal State

Example 4: 8-queens problem

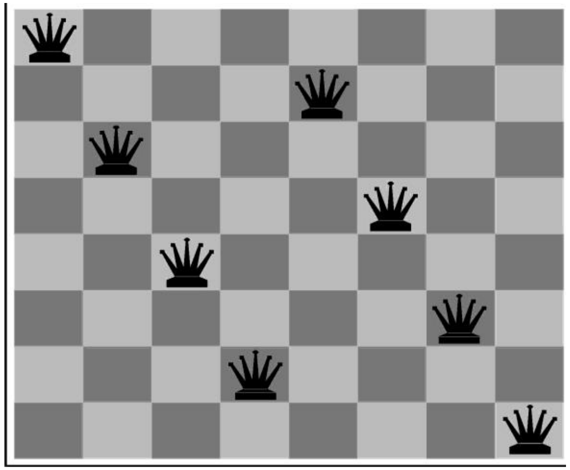
Problem



Solution



State-Space problem formulation



- states?
  -any arrangement of n<=8 queens
  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.
- initial state? no queens on the board
- actions?
  -add queen to any empty square
  -*or* add queen to leftmost empty square such that it is not attacked by other queens.
- goal test? 8 queens on the board, none attacked.
- path cost? 1 per move

Search

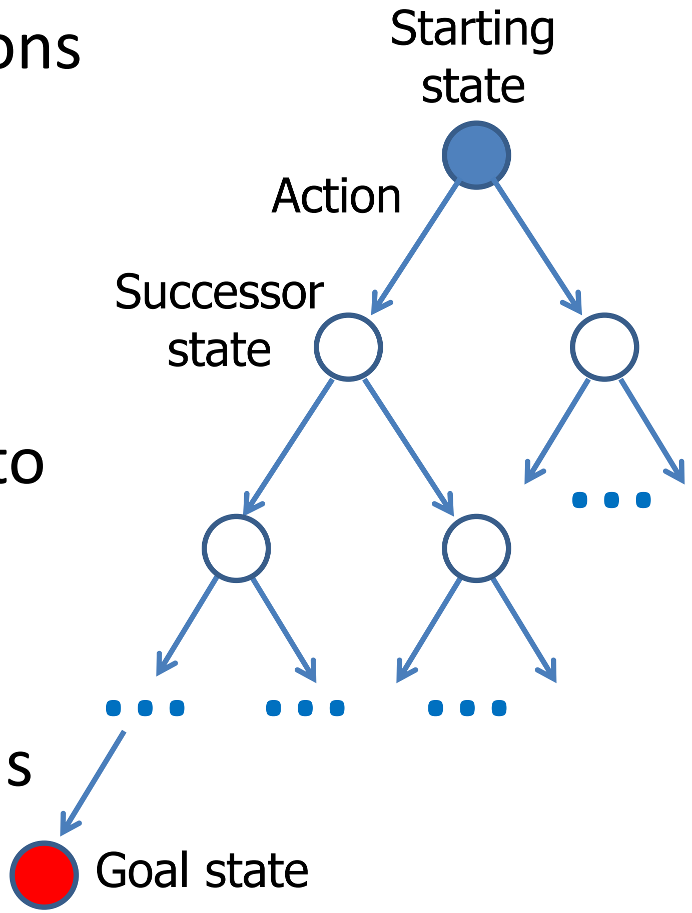
- Given:
  - **Initial state**
  - **Actions**
  - **Transition model**
  - **Goal state**
  - **Path cost**
- How do we find the **optimal solution**?
  - How about building the state space and then using Dijkstra's shortest path algorithm?
    - Complexity of Dijkstra's is $O(E + V \log V)$, where V is the size of the state space
    - For AI problems in particular the state space may be huge!

Search: Basic idea

- Let's begin at the start state and **expand** it by making a list of all possible successor states
- We NEVER build the ENTIRE state space at once
- Maintain a **frontier** or **a list of unexpanded states**
- At each step, pick a state from the frontier to expand
- Keep going until you reach a **goal** state
- Try to expand as few states as possible

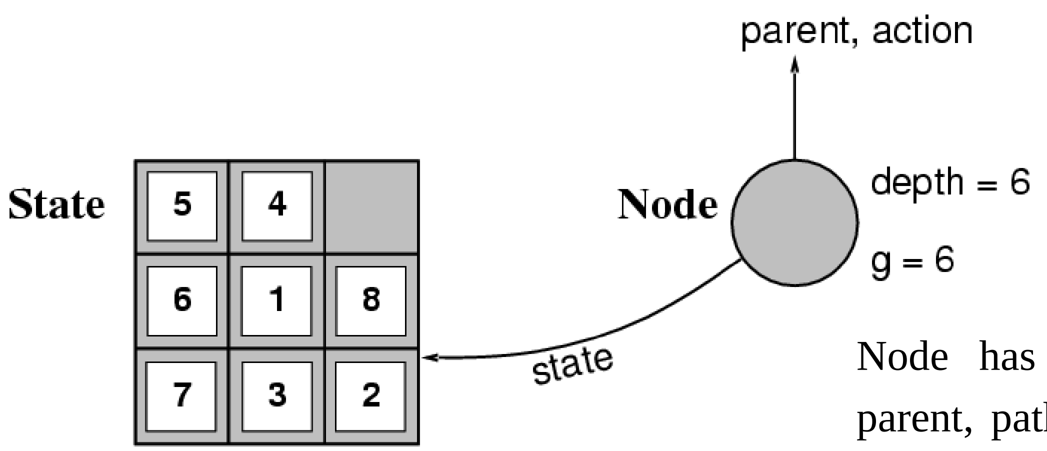
Search Graph to Tree

- "What if" tree of sequences of actions and outcomes
- The root node corresponds to the starting state
- The children of a node correspond to the **successor states** of that node's state
- A path through the tree corresponds to a sequence of actions
  - A solution is a path ending in the goal state



States versus Nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree contains info such as: state, parent node, action, path cost *g(x)*, depth



Node has to keep pointer to parent, path cost, possibly other info

- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

Search Tree for the 8 puzzle problem

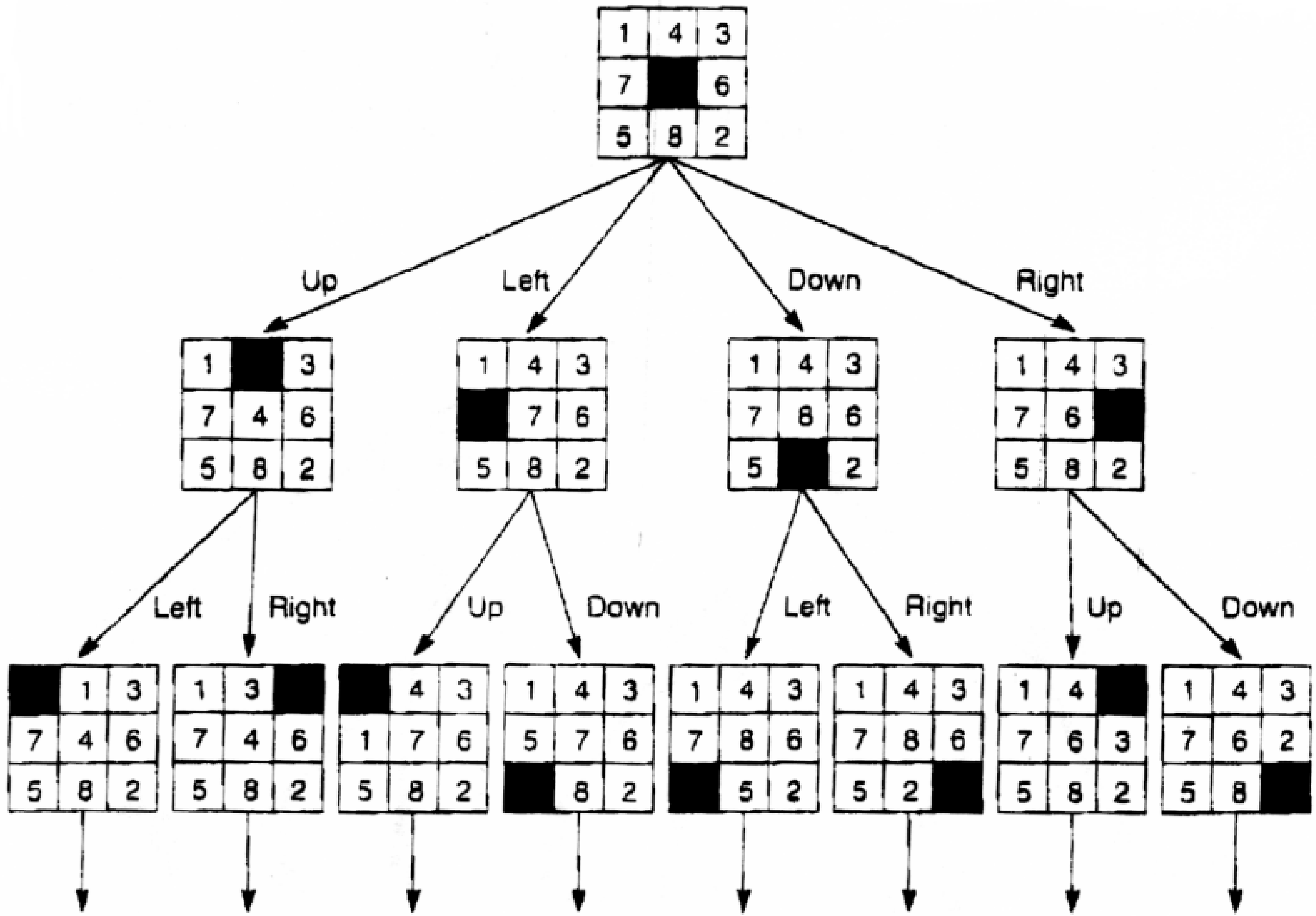
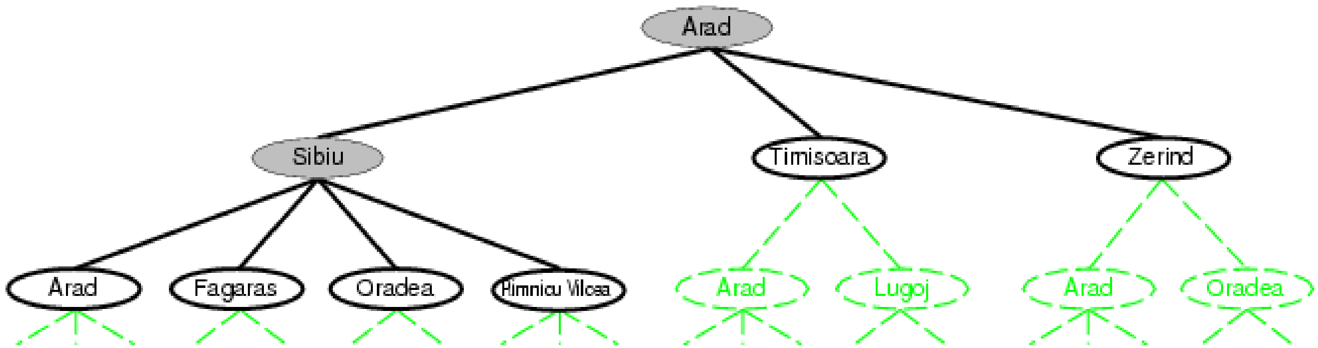


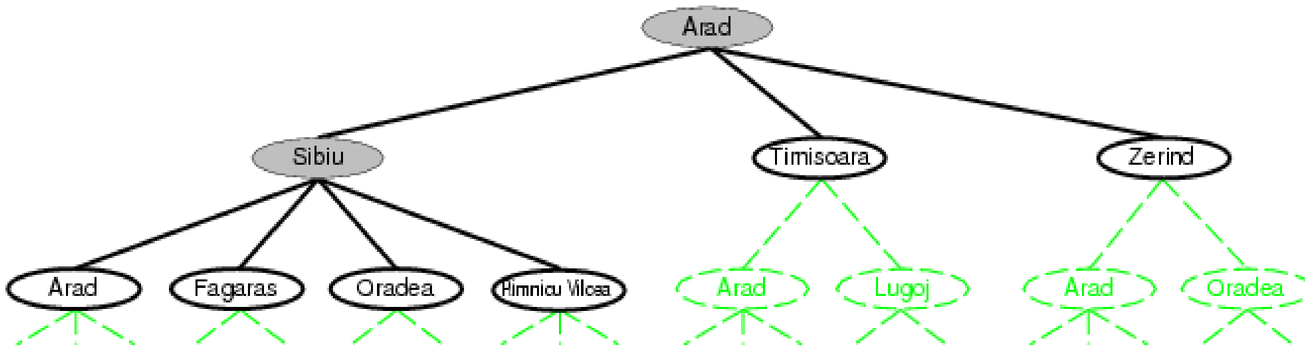
Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

Tree Search Algorithm



function TREE-SEARCH(*problem*, *strategy*) **returns** a solution, or failure
 initialize the search tree using the initial state of *problem*
 loop do
  if there are no candidates for expansion **then return** failure
  choose a leaf node for expansion according to *strategy*
  if the node contains a goal state **then return** the corresponding solution
  else expand the node and add the resulting nodes to the search tree

Tree Search Algorithm



This "strategy" is what differentiates different search algorithms

function TREE-SEARCH(*problem, strategy*) **returns** a solution failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose a leaf node for expansion according to *strategy*
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add the resulting nodes to the search tree

Search Strategies

- A search strategy is defined by picking the order of node expansion e.g. Breadth First Search
- Strategies are evaluated along the following dimensions:
  1. **Completeness**
     - Guarantees finding a solution whenever one exists
  2. **Optimality/Admissibility**
     - If a solution is found, is it **guaranteed to be an optimal** one? For example, is it the one with minimum cost?
  3. **Time Complexity**
     - How long (worst or average case) does it take to find a solution? Usually measured in terms of the **number of nodes expanded**
  4. **Space Complexity**
     - How much space is used by the algorithm? Usually measured in terms of the **maximum size that the “OPEN" list** becomes during the search

Evaluating Search Strategies

- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

The End