Artificial Intelligence
Unit 1
TYIT

By Prof Shaikh Bilal Naseem
KJ Somaiya College of Science and Commerce,
Somaiya Vidyavihar University, Mumbai 77.

What is Artificial Intelligence?

According to the father of Artificial Intelligence, John McCarthy, it is *"The science and engineering of making intelligent machines, especially intelligent computer programs"*.

Artificial Intelligence is a way of **making a computer, a computer-controlled robot, or a software think intelligently**, in the similar manner the intelligent humans think.

AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

Philosophy of AI

While exploiting the power of the computer systems, the curiosity of human, lead him to wonder, *"Can a machine think and behave like humans do?"*

Thus, the development of AI started with the intention of creating similar intelligence in machines that we find and regard high in humans.

Goals of AI

- **To Create Expert Systems** – The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.
- **To Implement Human Intelligence in Machines** – Creating systems that understand, think, learn, and behave like humans.

What Contributes to AI?

Artificial intelligence is a science and technology based on disciplines such as Computer Science, Biology, Psychology, Linguistics, Mathematics, and Engineering. A major thrust of AI is in the development of computer functions associated with human intelligence, such as reasoning, learning, and problem solving.

Out of the following areas, one or multiple areas can contribute to build an intelligent system.

Programming Without and With AI

The programming without and with AI is different in following ways –

Programming Without AI	Programming With AI
A computer program without AI can answer the **specific** questions it is meant to solve.	A computer program with AI can answer the **generic** questions it is meant to solve.
Modification in the program leads to change in its structure.	AI programs can absorb new modifications by putting highly independent pieces of information together. Hence you can modify even a minute piece of information of program without affecting its structure.
Modification is not quick and easy. It may lead to affecting the program adversely.	Quick and Easy program modification.

What is AI Technique?

In the real world, the knowledge has some unwelcomed properties −

- Its volume is huge, next to unimaginable.
- It is not well-organized or well-formatted.
- It keeps changing constantly.

AI Technique is a manner to organize and use the knowledge efficiently in such a way that −

- It should be perceivable by the people who provide it.
- It should be easily modifiable to correct errors.
- It should be useful in many situations though it is incomplete or inaccurate.

AI techniques elevate the speed of execution of the complex program it is equipped with.

Applications of AI

AI has been dominant in various fields such as –

- **Gaming** – AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
- **Natural Language Processing** – It is possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems** – There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
- **Vision Systems** – These systems understand, interpret, and comprehend visual input on the computer. For example,
  - A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.

- **Speech Recognition** – Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.

- **Handwriting Recognition** – The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.

- **Intelligent Robots** – Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

History of AI

Here is the history of AI during 20th century –

Year	Milestone / Innovation
1923	Karel Čapek play named “Rossum's Universal Robots” (RUR) opens in London, first use of the word "robot" in English.
1943	Foundations for neural networks laid.
1945	Isaac Asimov, a Columbia University alumni, coined the term *Robotics*.
1950	Alan Turing introduced Turing Test for evaluation of intelligence and published *Computing Machinery and Intelligence.* Claude Shannon published *Detailed Analysis of Chess Playing* as a search.
1956	John McCarthy coined the term *Artificial Intelligence*. Demonstration of the first running AI program at Carnegie Mellon University.
1958	John McCarthy invents LISP programming language for AI.

1964	Danny Bobrow's dissertation at MIT showed that computers can understand natural language well enough to solve algebra word problems correctly.
1965	Joseph Weizenbaum at MIT built *ELIZA*, an interactive problem that carries on a dialogue in English.
1969	Scientists at Stanford Research Institute Developed *Shakey*, a robot, equipped with locomotion, perception, and problem solving.
1973	The Assembly Robotics group at Edinburgh University built *Freddy*, the Famous Scottish Robot, capable of using vision to locate and assemble models.
1979	The first computer-controlled autonomous vehicle, Stanford Cart, was built.
1985	Harold Cohen created and demonstrated the drawing program, *Aaron*.

1990	Major advances in all areas of AI – ▪ Significant demonstrations in machine learning ▪ Case-based reasoning ▪ Multi-agent planning ▪ Scheduling ▪ Data mining, Web Crawler ▪ natural language understanding and translation ▪ Vision, Virtual Reality ▪ Games
1997	The Deep Blue Chess Program beats the then world chess champion, Garry Kasparov.
2000	Interactive robot pets become commercially available. MIT displays *Kismet*, a robot with a face that expresses emotions. The robot *Nomad* explores remote regions of Antarctica and locates meteorites.

What is Intelligence?

The ability of a system to calculate, reason, perceive relationships and analogies, learn from experience, store and retrieve information from memory, solve problems, comprehend complex ideas, use natural language fluently, classify, generalize, and adapt new situations.

Types of Intelligence

As described by Howard Gardner, an American developmental psychologist, the Intelligence comes in multifold –

Intelligence	Description	Example
Linguistic intelligence	The ability to speak, recognize, and use mechanisms of phonology (speech sounds), syntax (grammar), and semantics (meaning).	Narrators, Orators

Musical intelligence	The ability to create, communicate with, and understand meanings made of sound, understanding of pitch, rhythm.	Musicians, Singers, Composers
Logical-mathematical intelligence	The ability of use and understand relationships in the absence of action or objects. Understanding complex and abstract ideas.	Mathematicians, Scientists

Spatial intelligence	The ability to perceive visual or spatial information, change it, and re-create visual images without reference to the objects, construct 3D images, and to move and rotate them.	Map readers, Astronauts, Physicists
Bodily-Kinesthetic intelligence	The ability to use complete or part of the body to solve problems or fashion products, control over fine and coarse motor skills, and manipulate the objects.	Players, Dancers

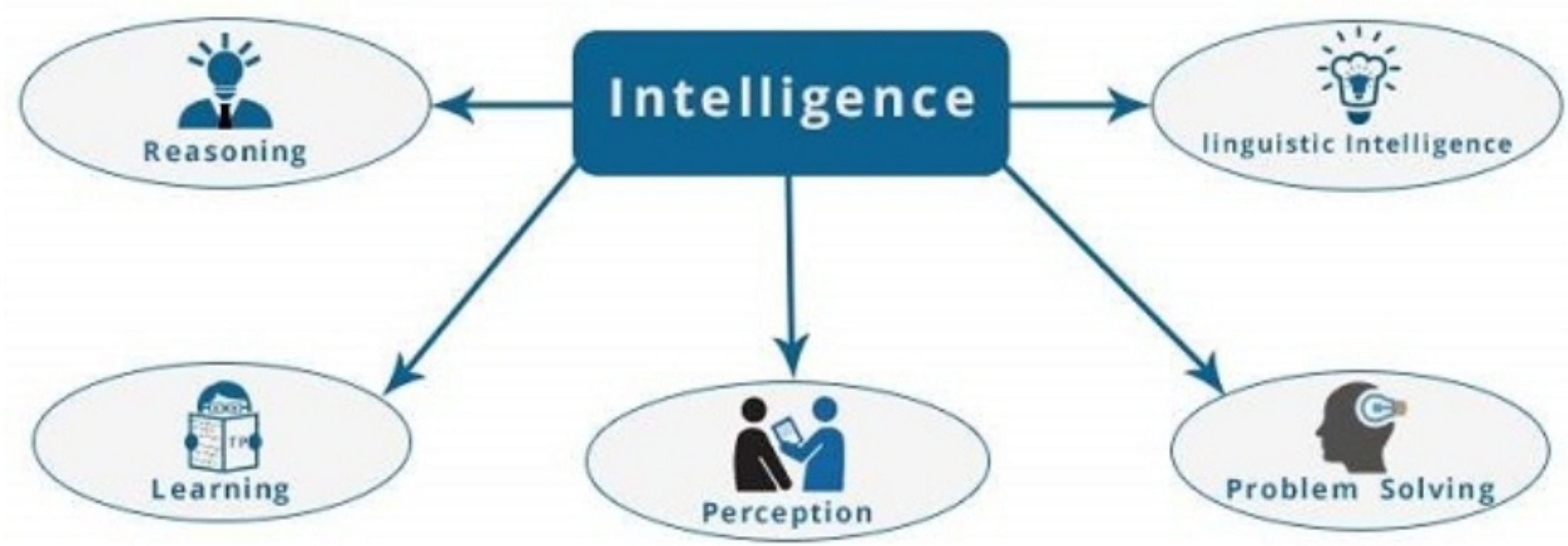
Intra-personal intelligence	The ability to distinguish among one's own feelings, intentions, and motivations.	Gautam Buddhha
Interpersonal intelligence	The ability to recognize and make distinctions among other people's feelings, beliefs, and intentions.	Mass Communicators, Interviewers

You can say a machine or a system is **artificially intelligent** when it is equipped with at least one and at most all intelligences in it.

What is Intelligence Composed of?

The intelligence is intangible. It is composed of −

- Reasoning
- Learning
- Problem Solving
- Perception
- Linguistic Intelligence



- **Reasoning** – It is the set of processes that enables us to provide basis for judgement, making decisions, and prediction. There are broadly two types –

Inductive Reasoning	Deductive Reasoning
It conducts specific observations to makes broad general statements.	It starts with a general statement and examines the possibilities to reach a specific, logical conclusion.
Even if all of the premises are true in a statement, inductive reasoning allows for the conclusion to be false.	If something is true of a class of things in general, it is also true for all members of that class.
Example – "Nita is a teacher. Nita is studious. Therefore, All teachers are studious."	Example – "All women of age above 60 years are grandmothers. Shalini is 65 years. Therefore, Shalini is a grandmother."

- **Learning** – It is the activity of gaining knowledge or skill by studying, practising, being taught, or experiencing something. Learning enhances the awareness of the subjects of the study.

  The ability of learning is possessed by humans, some animals, and AI-enabled systems. Learning is categorized as –

  - **Auditory Learning** – It is learning by listening and hearing. For example, students listening to recorded audio lectures.
  - **Episodic Learning** – To learn by remembering sequences of events that one has witnessed or experienced. This is linear and orderly.
  - **Motor Learning** – It is learning by precise movement of muscles. For example, picking objects, Writing, etc.

- **Observational Learning** – To learn by watching and imitating others. For example, child tries to learn by mimicking her parent.
- **Perceptual Learning** – It is learning to recognize stimuli that one has seen before. For example, identifying and classifying objects and situations.
- **Relational Learning** – It involves learning to differentiate among various stimuli on the basis of relational properties, rather than absolute properties. For Example, Adding 'little less' salt at the time of cooking potatoes that came up salty last time, when cooked with adding say a tablespoon of salt.
- **Spatial Learning** – It is learning through visual stimuli such as images, colors, maps, etc. For Example, A person can create roadmap in mind before actually following the road.

- **Stimulus-Response Learning** – It is learning to perform a particular behavior when a certain stimulus is present. For example, a dog raises its ear on hearing doorbell.

- **Problem Solving** – It is the process in which one perceives and tries to arrive at a desired solution from a present situation by taking some path, which is blocked by known or unknown hurdles.

  Problem solving also includes **decision making**, which is the process of selecting the best suitable alternative out of multiple alternatives to reach the desired goal are available.

- **Perception** – It is the process of acquiring, interpreting, selecting, and organizing sensory information.

  Perception presumes **sensing**. In humans, perception is aided by sensory organs. In the domain of AI, perception mechanism puts the data acquired by the sensors together in a meaningful manner.

- **Linguistic Intelligence** – It is one's ability to use, comprehend, speak, and write the verbal and written language. It is important in interpersonal communication.

Difference between Human and Machine Intelligence

- Humans perceive by patterns whereas the machines perceive by set of rules and data.
- Humans store and recall information by patterns, machines do it by searching algorithms. For example, the number 40404040 is easy to remember, store, and recall as its pattern is simple.
- Humans can figure out the complete object even if some part of it is missing or distorted; whereas the machines cannot do it correctly.

Real Life Applications of Research Areas

There is a large array of applications where AI is serving common people in their day-to-day lives –

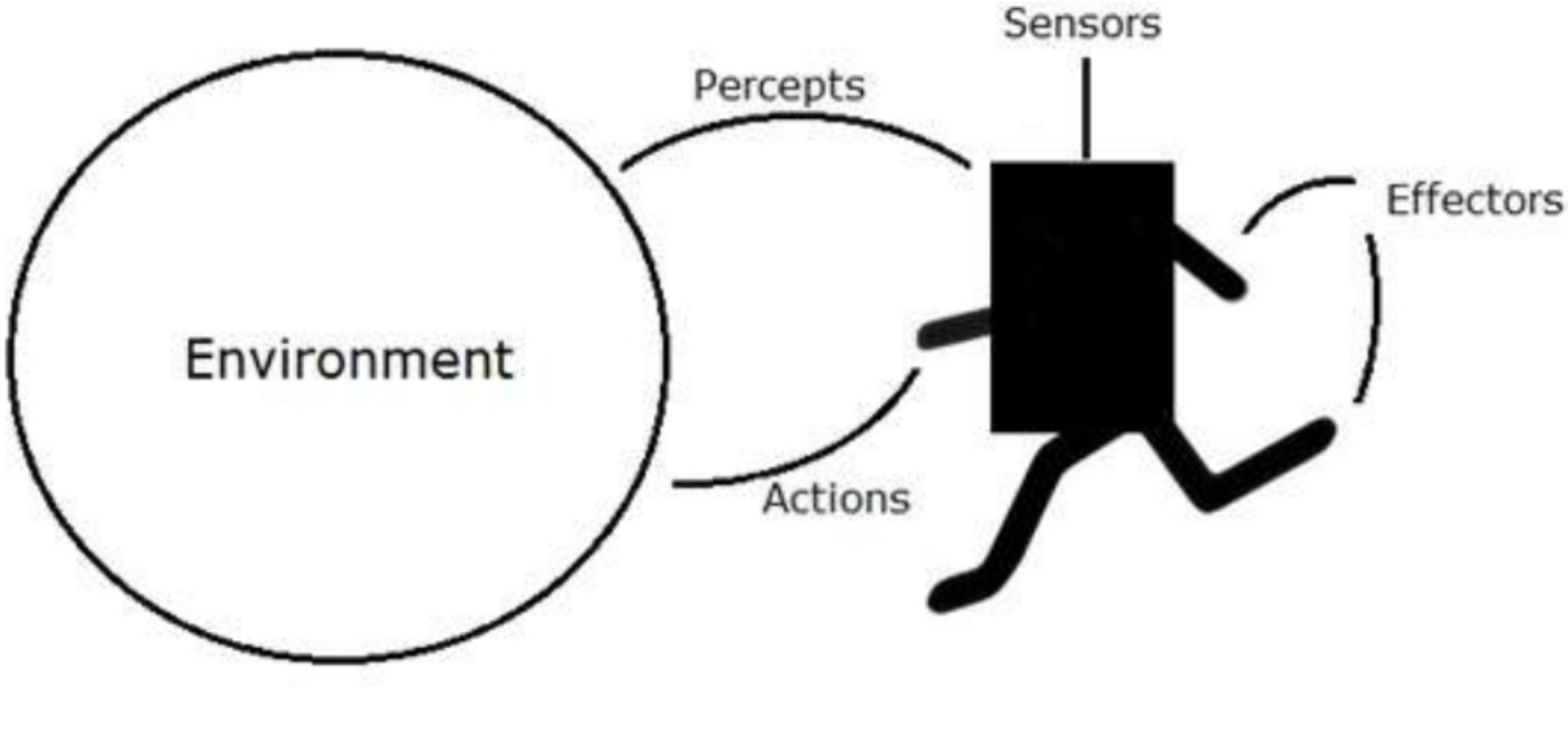
Sr.No.	Research Areas	Real Life Application
1	**Expert Systems** Examples – Flight-tracking systems, Clinical systems.	
2	**Natural Language Processing** Examples: Google Now feature, speech recognition, Automatic voice output.	
3	**Neural Networks** Examples – Pattern recognition systems such as face recognition, character recognition, handwriting recognition.	

4	**Robotics** Examples – Industrial robots for moving, spraying, painting, precision checking, drilling, cleaning, coating, carving, etc.	
5	**Fuzzy Logic Systems** Examples – Consumer electronics, automobiles, etc.	

What are Agent and Environment?

An **agent** is anything that can perceive its environment through **sensors** and acts upon that environment through **effectors.**

- A **human agent** has sensory organs such as eyes, ears, nose, tongue and skin parallel to the sensors, and other organs such as hands, legs, mouth, for effectors.
- A **robotic agent** replaces cameras and infrared range finders for the sensors, and various motors and actuators for effectors.
- A **software agent** has encoded bit strings as its programs and actions.



Agent Terminology

- **Performance Measure of Agent** – It is the criteria, which determines how successful an agent is.
- **Behavior of Agent** – It is the action that agent performs after any given sequence of percepts.
- **Percept** – It is agent's perceptual inputs at a given instance.
- **Percept Sequence** – It is the history of all that an agent has perceived till date.
- **Agent Function** – It is a map from the precept sequence to an action.

Rationality

Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.

Rationality is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of rationality.

What is Ideal Rational Agent?

An ideal rational agent is the one, which is capable of doing expected actions to maximize its performance measure, on the basis of –

- Its percept sequence
- Its built-in knowledge base

Rationality of an agent depends on the following –

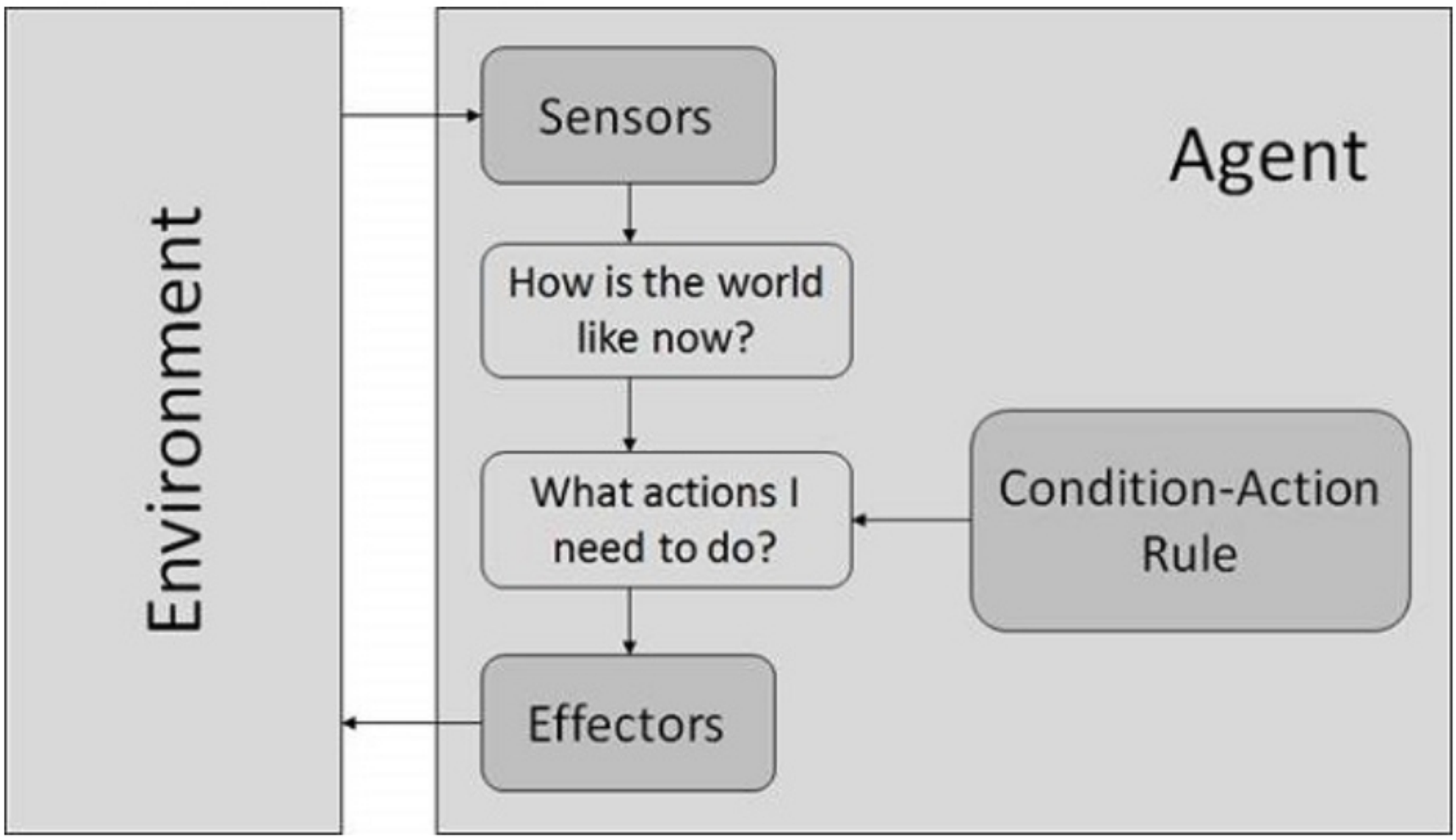
- The **performance measures**, which determine the degree of success.
- Agent's **Percept Sequence** till now.
- The agent's **prior knowledge about the environment**.
- The **actions** that the agent can carry out.

A rational agent always performs right action, where the right action means the action that causes the agent to be most successful in the given percept sequence. The problem the agent solves is characterized by Performance Measure, Environment, Actuators, and Sensors (PEAS).

Simple Reflex Agents

- They choose actions only based on the current percept.
- They are rational only if a correct decision is made only on the basis of current precept.
- Their environment is completely observable.

Condition-Action Rule – It is a rule that maps a state (condition) to an action.



Model Based Reflex Agents

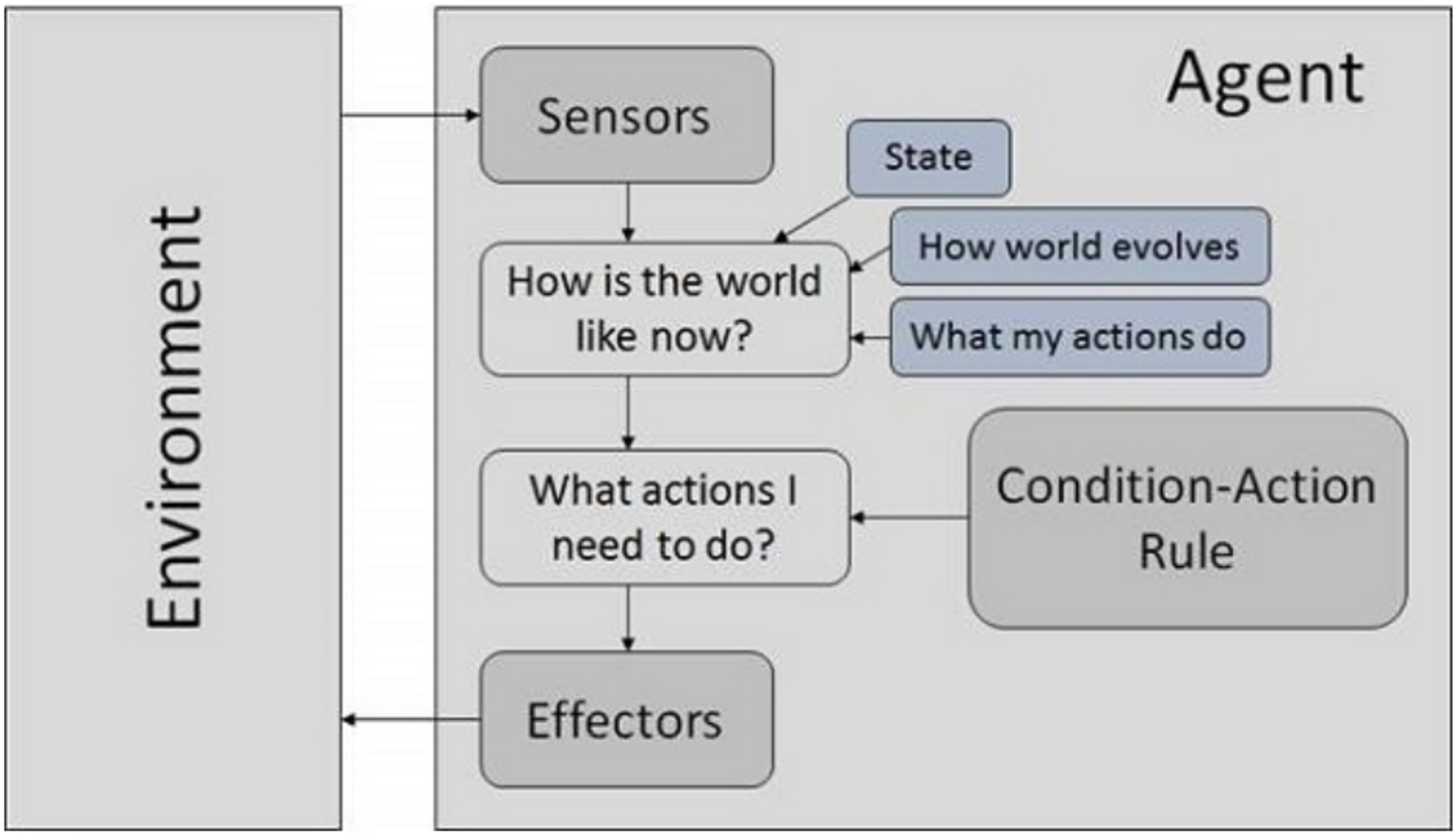
They use a model of the world to choose their actions. They maintain an internal state.

Model – knowledge about "how the things happen in the world".

Internal State – It is a representation of unobserved aspects of current state depending on percept history.

Updating the state requires the information about –

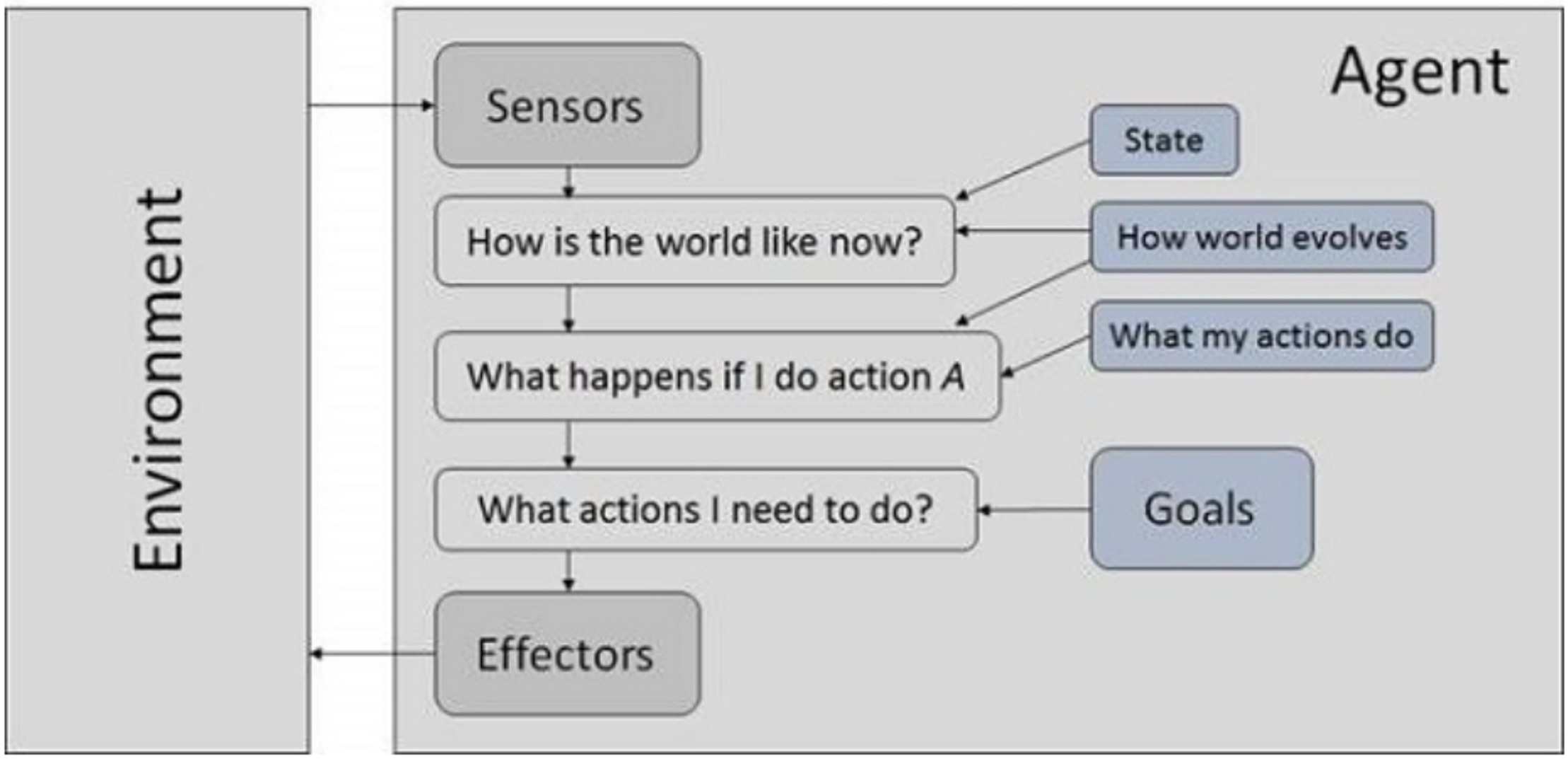
- How the world evolves.
- How the agent's actions affect the world.



Goal Based Agents

They choose their actions in order to achieve goals. Goal-based approach is more flexible than reflex agent since the knowledge supporting a decision is explicitly modeled, thereby allowing for modifications.

Goal – It is the description of desirable situations.

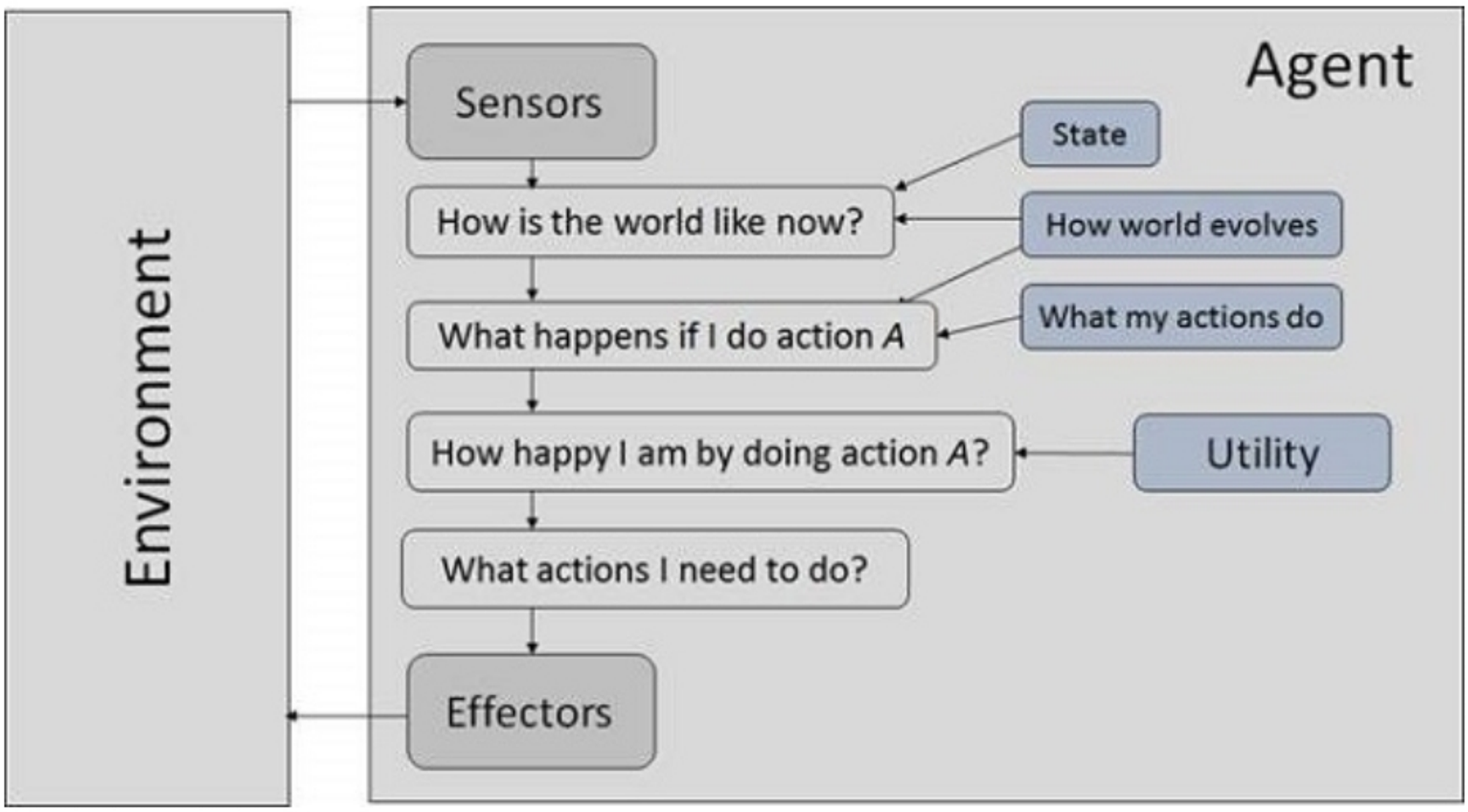


Utility Based Agents

They choose actions based on a preference (utility) for each state.

Goals are inadequate when −

- There are conflicting goals, out of which only few can be achieved.
- Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.



The Nature of Environments

Some programs operate in the entirely **artificial environment** confined to keyboard input, database, computer file systems and character output on a screen.

In contrast, some software agents (software robots or softbots) exist in rich, unlimited softbots domains. The simulator has a **very detailed, complex environment**. The software agent needs to choose from a long array of actions in real time. A softbot designed to scan the online preferences of the customer and show interesting items to the customer works in the **real** as well as an **artificial** environment.

The most famous **artificial environment** is the **Turing Test environment**, in which one real and other artificial agents are tested on equal ground. This is a very challenging environment as it is highly difficult for a software agent to perform as well as a human.

Turing Test

The success of an intelligent behavior of a system can be measured with Turing Test.

Two persons and a machine to be evaluated participate in the test. Out of the two persons, one plays the role of the tester. Each of them sits in different rooms. The tester is unaware of who is machine and who is a human. He interrogates the questions by typing and sending them to both intelligences, to which he receives typed responses.

This test aims at fooling the tester. If the tester fails to determine machine's response from the human response, then the machine is said to be intelligent.

Properties of Environment

The environment has multifold properties –

- **Discrete / Continuous** – If there are a limited number of distinct, clearly defined, states of the environment, the environment is discrete (For example, chess); otherwise it is continuous (For example, driving).
- **Observable / Partially Observable** – If it is possible to determine the complete state of the environment at each time point from the percepts it is observable; otherwise it is only partially observable.
- **Static / Dynamic** – If the environment does not change while an agent is acting, then it is static; otherwise it is dynamic.
- **Single agent / Multiple agents** – The environment may contain other agents which may be of the same or different kind as that of the agent.

- **Accessible / Inaccessible** – If the agent's sensory apparatus can have access to the complete state of the environment, then the environment is accessible to that agent.

- **Deterministic / Non-deterministic** – If the next state of the environment is completely determined by the current state and the actions of the agent, then the environment is deterministic; otherwise it is non-deterministic.

- **Episodic / Non-episodic** – In an episodic environment, each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself. Subsequent episodes do not depend on the actions in the previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.

Artificial Intelligence
Chapter 3: Solving Problems by Searching

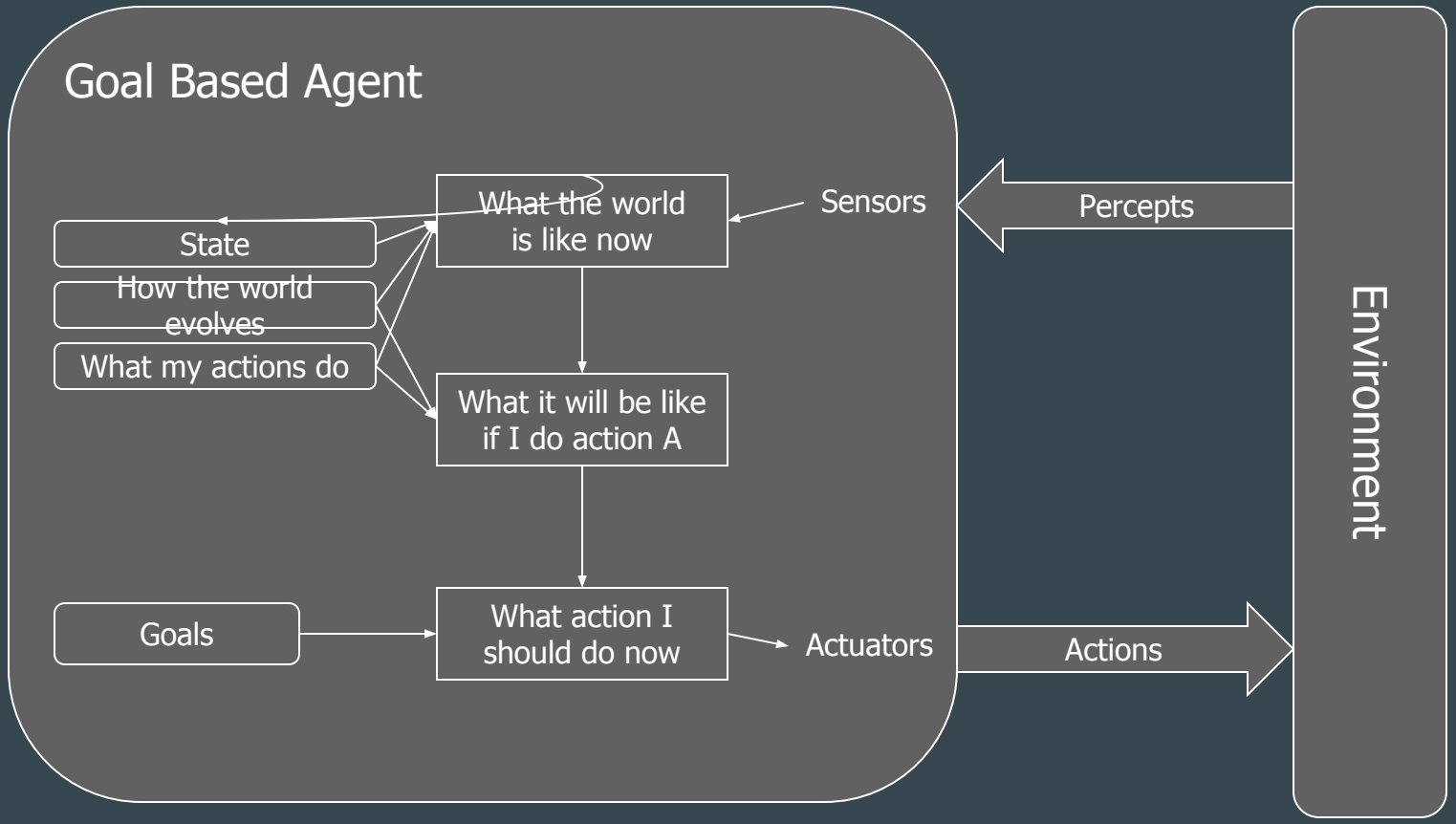
Prof. Shaikh Bilal Naseem
Department of CS/ IT
Somaiya Vidyavihar University

Problem Solving Agents

- Problem solving agent
  - A kind of "goal based" agent
  - Finds sequences of actions that lead to desirable states.

- The algorithms are uninformed
  - No extra information about the problem other than the definition
    - No extra information
    - No heuristics (rules)

Goal Based Agent



Goal Based Agent

```
Function Simple-Problem-Solving-Agent( percept ) returns action

Inputs: percept    a percept
Static: seq     an action sequence initially empty
        state      some description of the current world
        goal       a goal, initially null
        problem    a problem formulation

state <- UPDATE-STATE( state, percept )
if seq is empty then do
    goal <- FORMULATE-GOAL( state )
    problem <- FORMULATE-PROBLEM( state, goal )
    seq <- SEARCH( problem )                # SEARCH
action <- RECOMMENDATION ( seq )            # SOLUTION
seq <- REMAINDER( seq )
return action                          # EXECUTION
```

Goal Based Agents

- Assumes the problem environment is:
  - Static
    - The plan remains the same
  - Observable
    - Agent knows the initial state
  - Discrete
    - Agent can enumerate the choices
  - Deterministic
    - Agent can plan a sequence of actions such that each will lead to an intermediate state
- The agent carries out its plans with its eyes closed
  - Certain of what's going on
  - Open loop system

Well Defined Problems and Solutions

- A problem
  - Initial state
  - Actions and Successor Function
  - Goal test
  - Path cost

Example: Eight Puzzle

- States:
  - Description of the eight tiles and location of the blank tile
- Successor Function:
  - Generates the legal states from trying the four actions {*Left, Right, Up, Down*}
- Goal Test:
  - Checks whether the state matches the goal configuration
- Path Cost:
  - Each step costs 1

7	2	4	
5		6	
8	3	1	

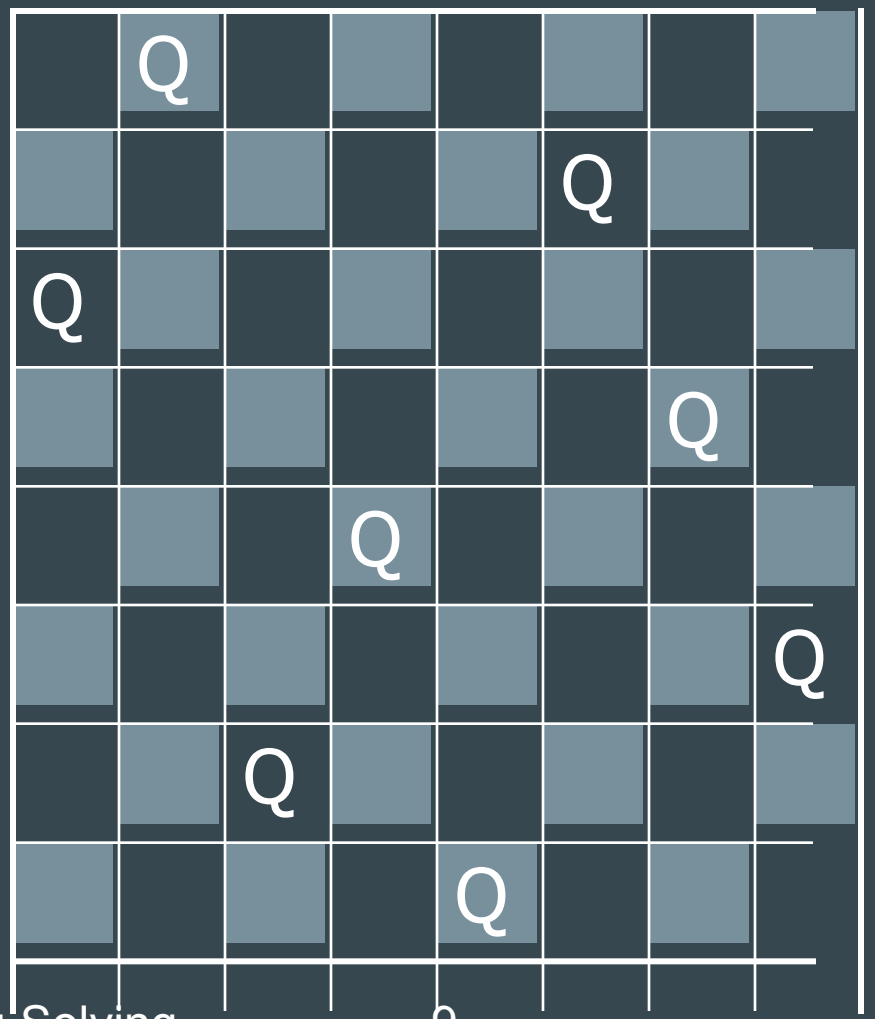
1	2	3	
4	5	6	
7	8		

Example: Eight Puzzle

- Eight puzzle is from a family of "sliding –block puzzles"
  - NP Complete
  - 8 puzzle has 9!/2 = 181440 states
  - 15 puzzle has approx. $1.3*10^{12}$ states
  - 24 puzzle has approx. $1*10^{25}$ states

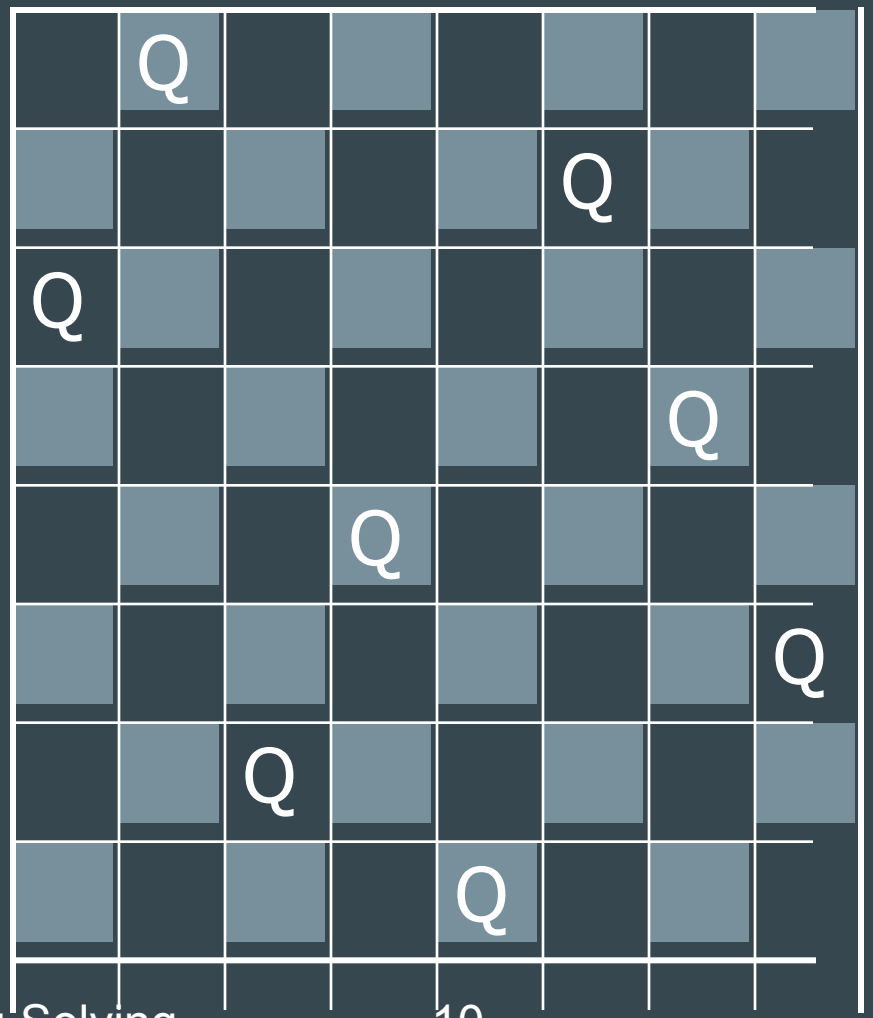
Example: Eight Queens

- Place eight queens on a chess board such that no queen can attack another queen
- No path cost because only the final state counts!
- Incremental formulations
- Complete state formulations



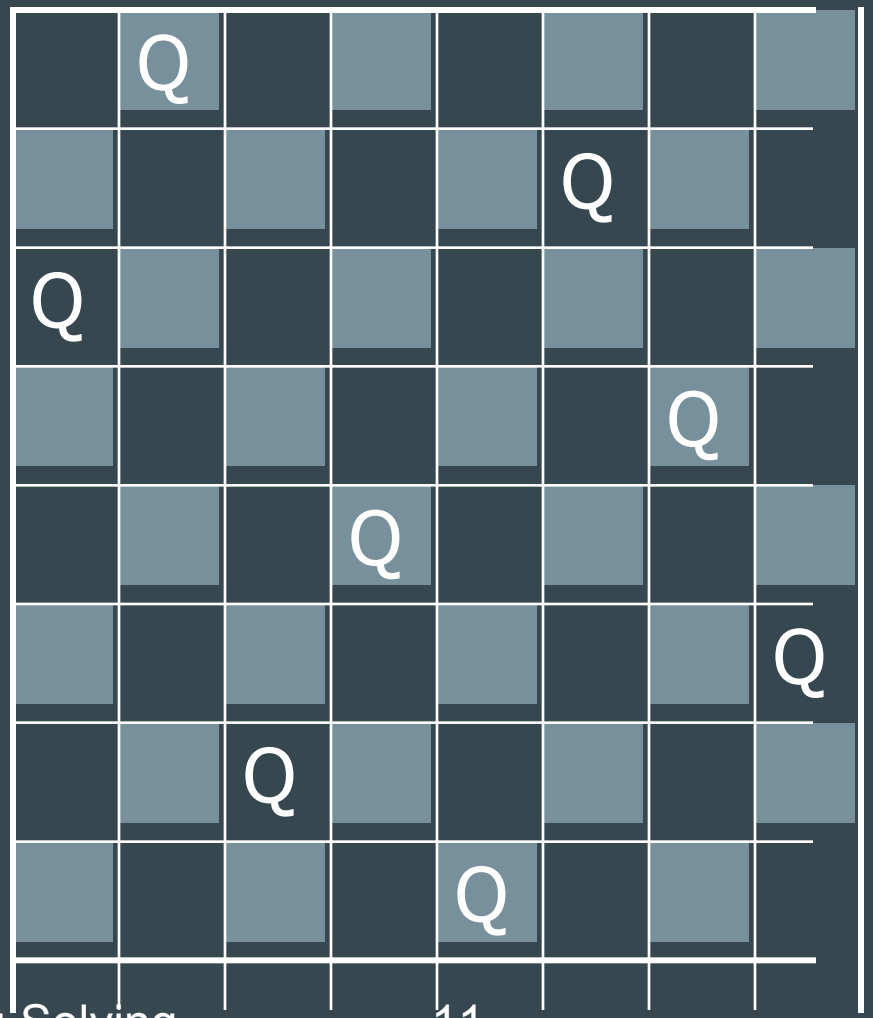
Example: Eight Queens

- States:
  - Any arrangement of 0 to 8 queens on the board
- Initial state:
  - No queens on the board
- Successor function:
  - Add a queen to an empty square
- Goal Test:
  - 8 queens on the board and none are attacked
- $64*63*\ldots*57 = 1.8*10^{14}$ possible sequences
  - Ouch!



Example: Eight Queens

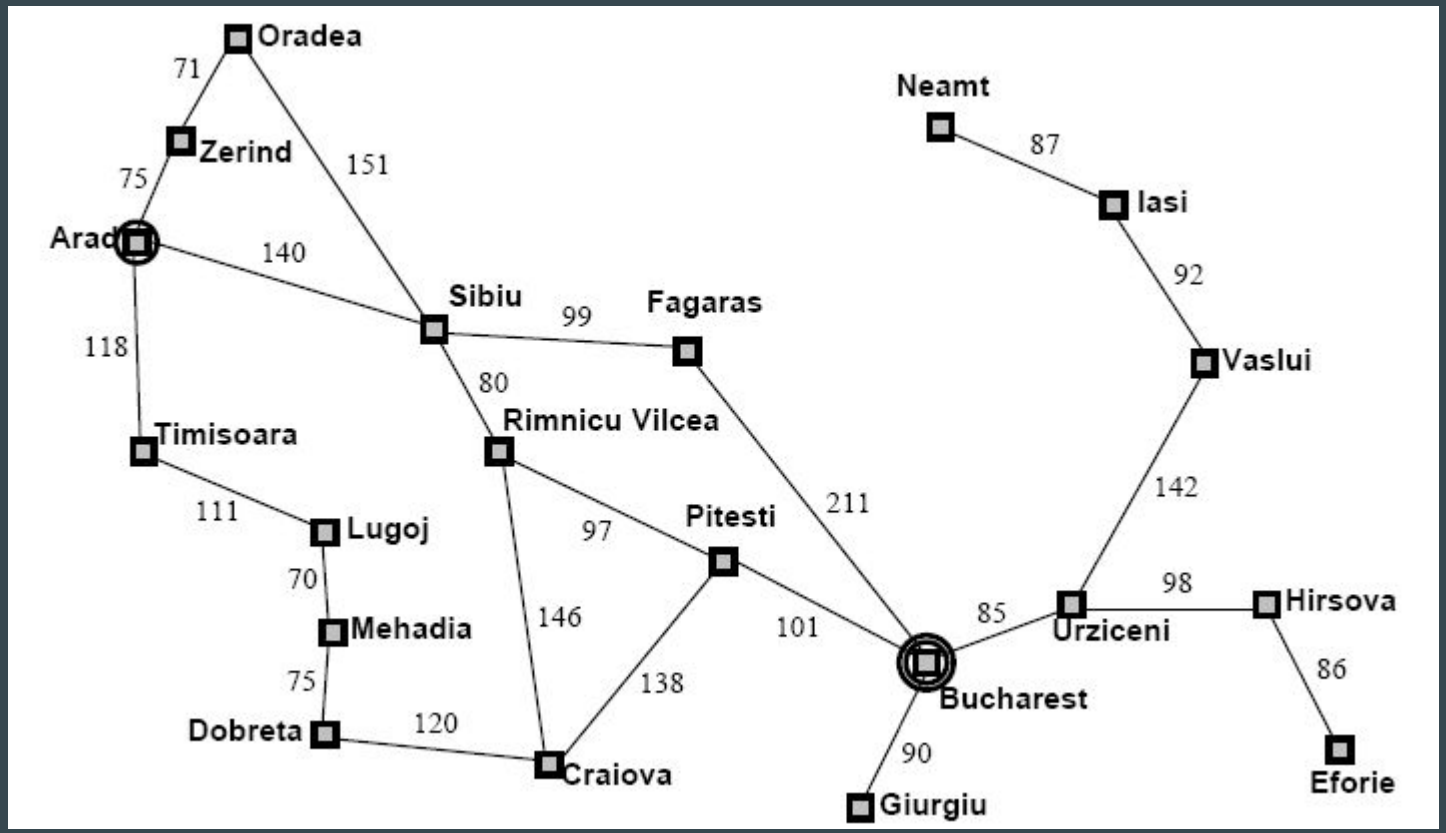
- States:
  - Arrangements of n queens, one per column in the leftmost n columns, with no queen attacking another are states
- Successor function:
  - Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.
- 2057 sequences to investigate



Other Toy Examples

- Another Example: Jug Fill
- Another Example: Black White Marbles
- Another Example: Row Boat Problem
- Another Example: Sliding Blocks
- Another Example: Triangle Tee

Example: Map Planning



Searching For Solutions

- Initial State
  - e.g. "At Arad"
- Successor Function
  - A set of action state pairs
  - S(Arad) = {(Arad->Zerind, Zerind), …}
- Goal Test
  - e.g. x = "at Bucharest"
- Path Cost
  - sum of the distances traveled

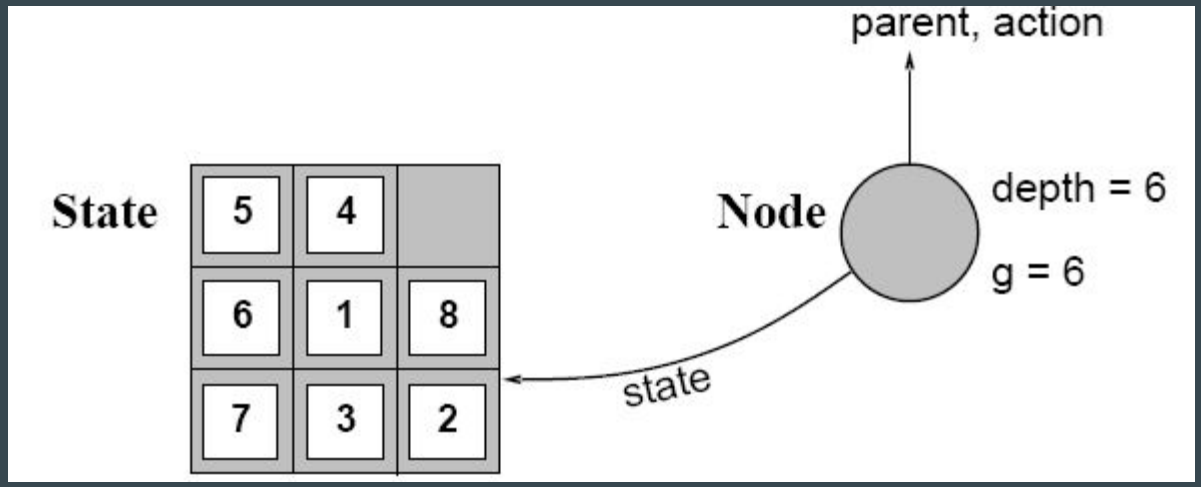
Searching For Solutions

- Having formulated some problems…how do we solve them?
- Search through a state space
- Use a search tree that is generated with an initial state and successor functions that define the state space

Searching For Solutions

- A ***state*** is (a representation of) a physical configuration
- A ***node*** is a data structure constituting part of a search tree
  - Includes parent, children, depth, path cost
- States do not have children, depth, or path cost
- The EXPAND function creates new nodes, filling in the various fields and using the SUCCESSOR function of the problem to create the corresponding states

Searching For Solutions



Searching For Solutions

```
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
    end
```

Searching For Solutions

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem] applied to STATE(node) succeeds return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Uninformed Search Strategies

- ***Uninformed*** strategies use only the information available in the problem definition
  - Also known as blind searching

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Comparing Uninformed Search Strategies

- Completeness
  - Will a solution always be found if one exists?
- Time
  - How long does it take to find the solution?
  - Often represented as the number of nodes searched
- Space
  - How much memory is needed to perform the search?
  - Often represented as the maximum number of nodes stored at once
- Optimal
  - Will the optimal (least cost) solution be found?

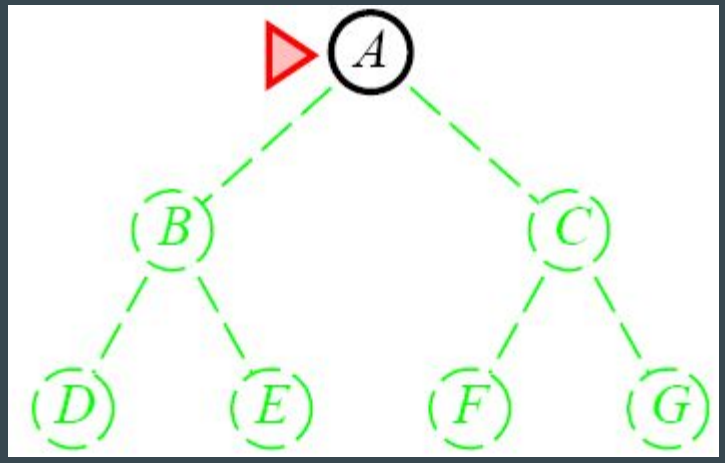
Comparing Uninformed Search Strategies

- Time and space complexity are measured in
  - b – maximum branching factor of the search tree
  - m – maximum depth of the state space
  - d – depth of the least cost solution

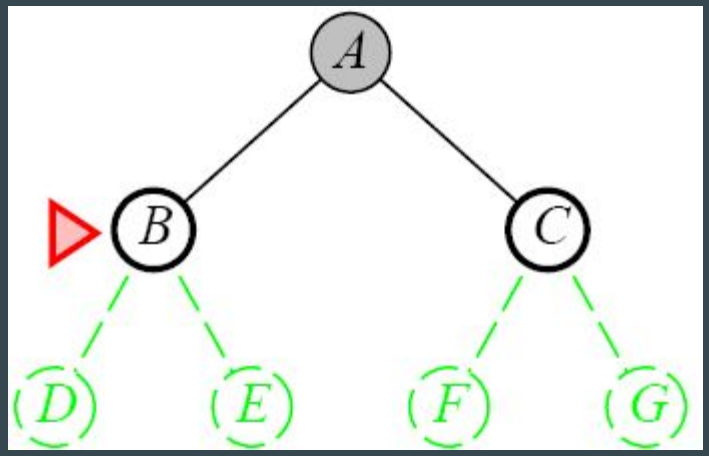
Breadth-First Search

- Recall from Data Structures the basic algorithm for a breadth-first search on a graph or tree
- Expand the ***shallowest*** unexpanded node
- Place all new successors at the end of a FIFO queue

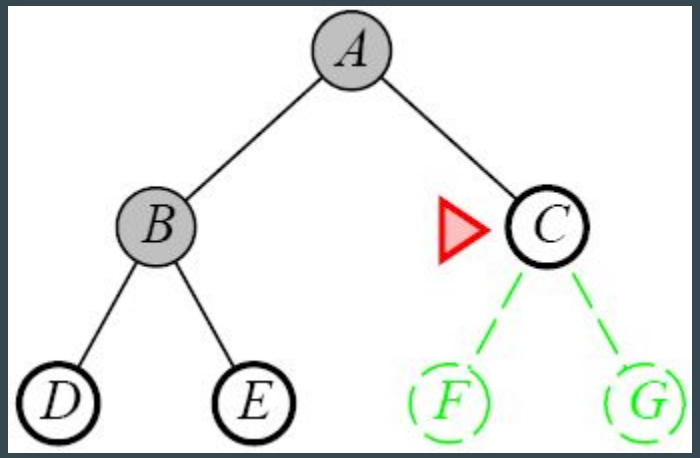
Breadth-First Search



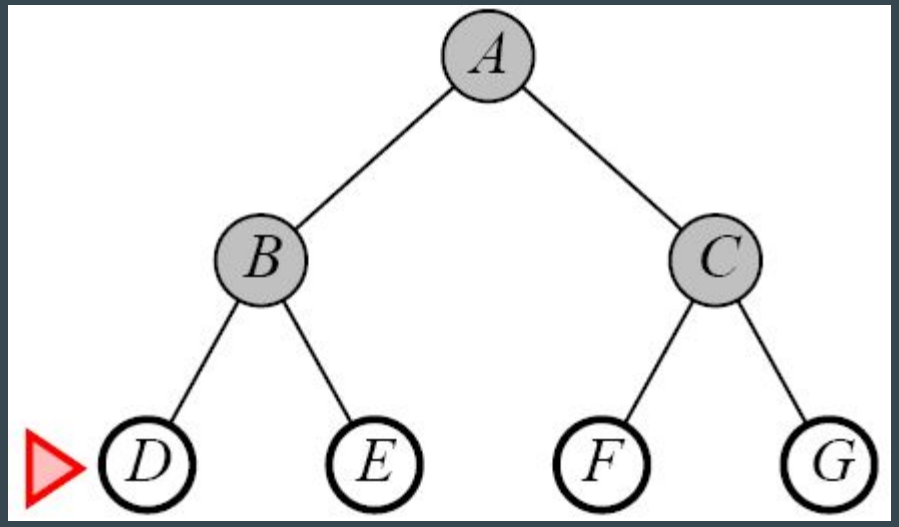
Breadth-First Search



Breadth-First Search



Breadth-First Search



Properties of Breadth-First Search

- Complete
  - Yes if b (max branching factor) is finite
- Time
  - $1 + b + b^2 + \ldots + b^d + b(b^d-1) = O(b^{d+1})$
  - exponential in d
- Space
  - $O(b^{d+1})$
  - Keeps every node in memory
  - This is the big problem; an agent that generates nodes at 10 MB/sec will produce 860 MB in 24 hours
- Optimal
  - Yes (if cost is 1 per step); not optimal in general

Lessons From Breadth First Search

- The memory requirements are a bigger problem for breadth-first search than is execution time

- Exponential-complexity search problems cannot be solved by uniformed methods for any but the smallest instances

Uniform-Cost Search

- Same idea as the algorithm for breadth-first search…but…
  - Expand the ***least-cost*** unexpanded node
  - FIFO queue is ordered by cost
  - Equivalent to regular breadth-first search if all step costs are equal

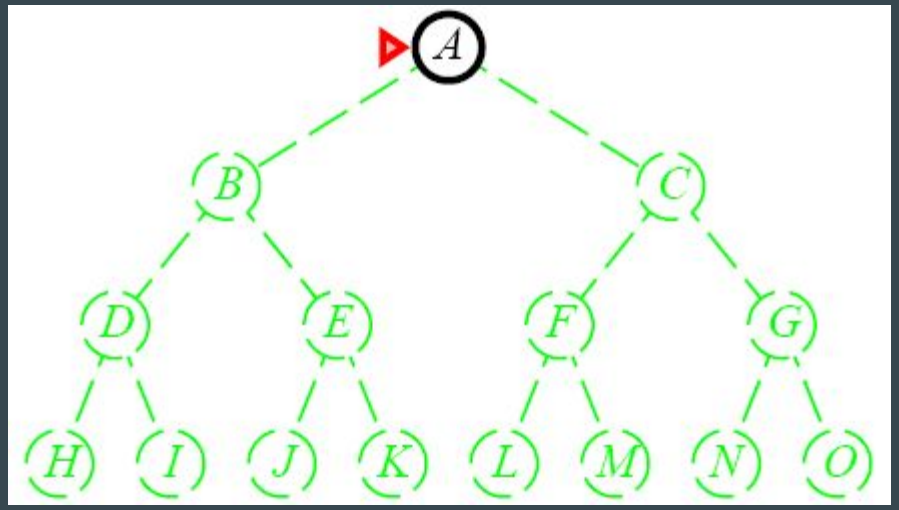
Uniform-Cost Search

- Complete
  - Yes if the cost is greater than some threshold
  - step cost >= ε
- Time
  - Complexity cannot be determined easily by d or d
  - Let C* be the cost of the optimal solution
  - $O(b^{ceil(C^*/ \varepsilon)})$
- Space
  - $O(b^{ceil(C^*/ \varepsilon)})$
- Optimal
  - Yes, Nodes are expanded in increasing order

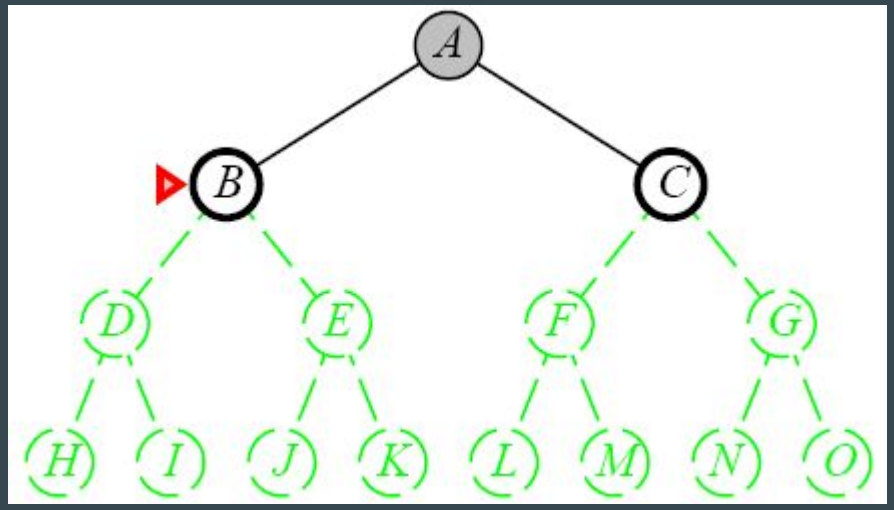
Depth-First Search

- Recall from Data Structures the basic algorithm for a depth-first search on a graph or tree
- Expand the ***deepest*** unexpanded node
- Unexplored successors are placed on a stack until fully explored

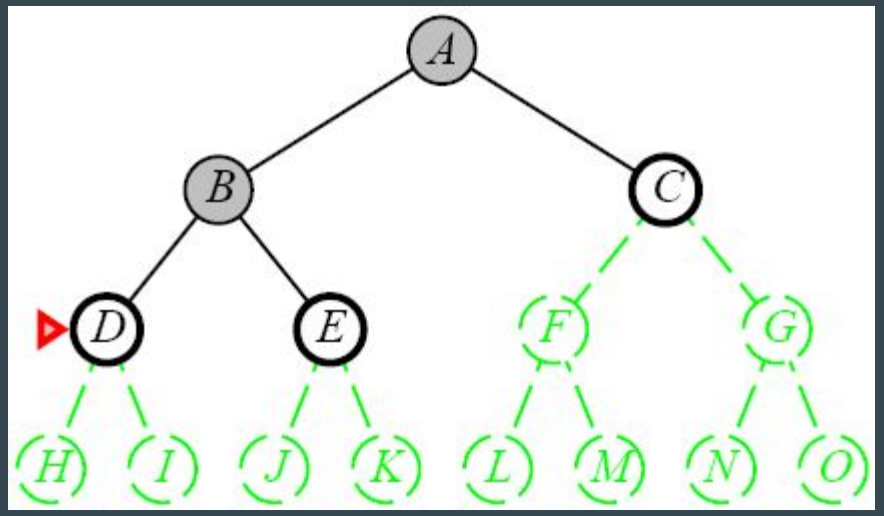
Depth-First Search



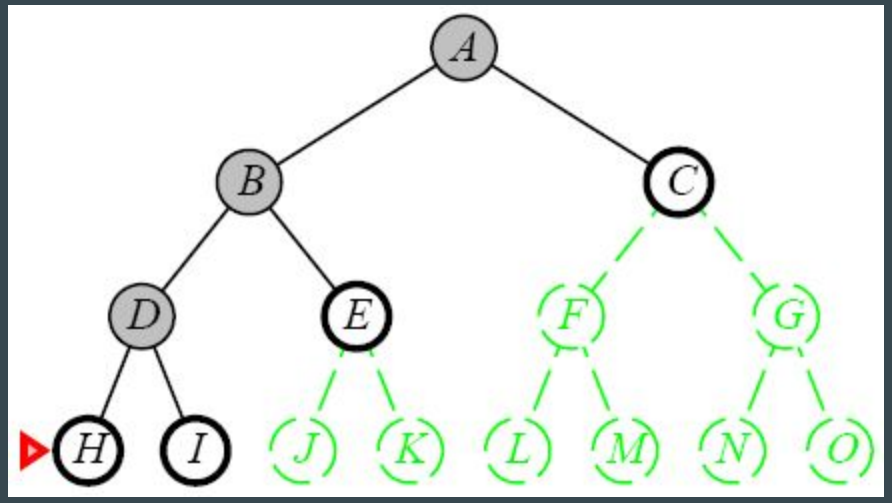
Depth-First Search



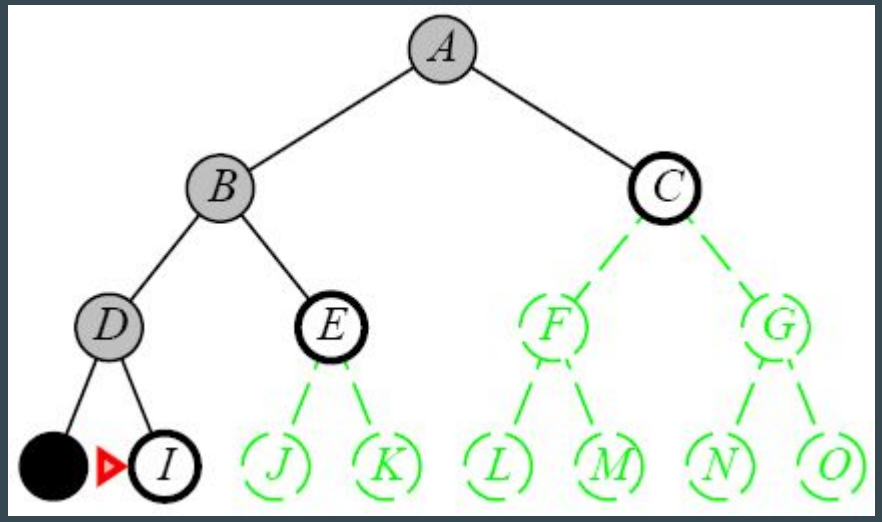
Depth-First Search



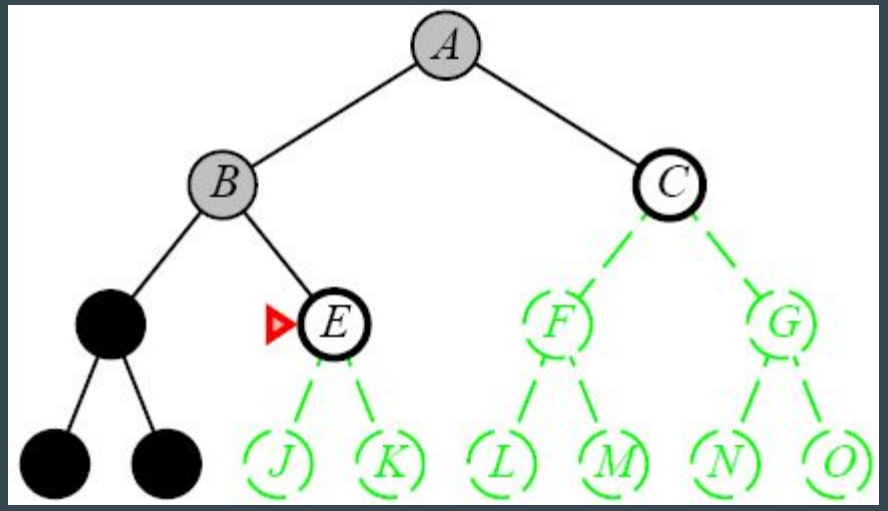
Depth-First Search



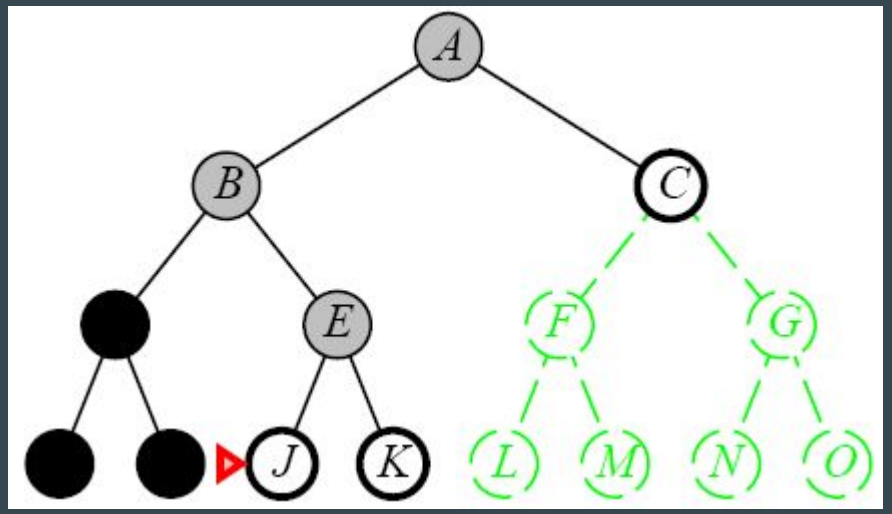
Depth-First Search



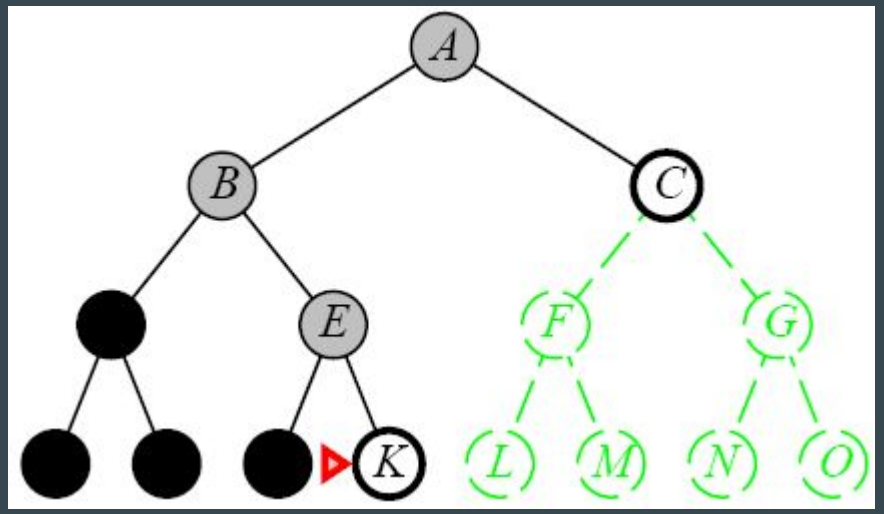
Depth-First Search



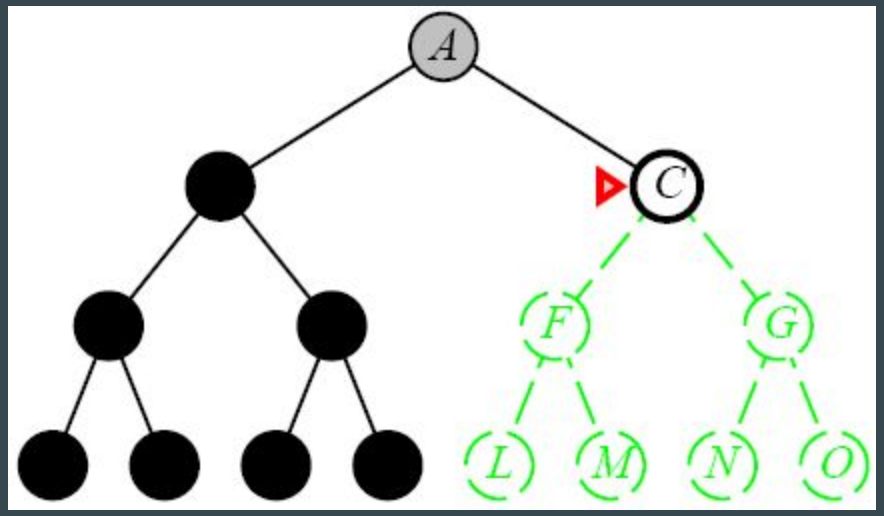
Depth-First Search



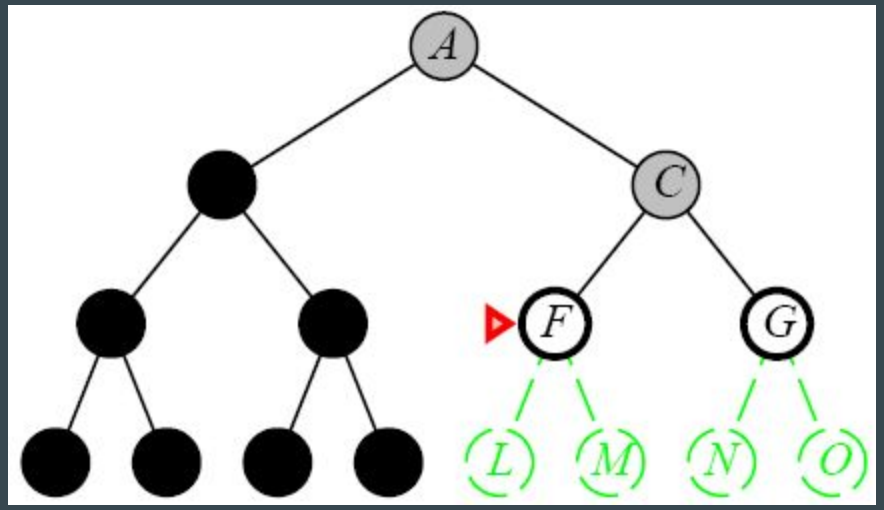
Depth-First Search



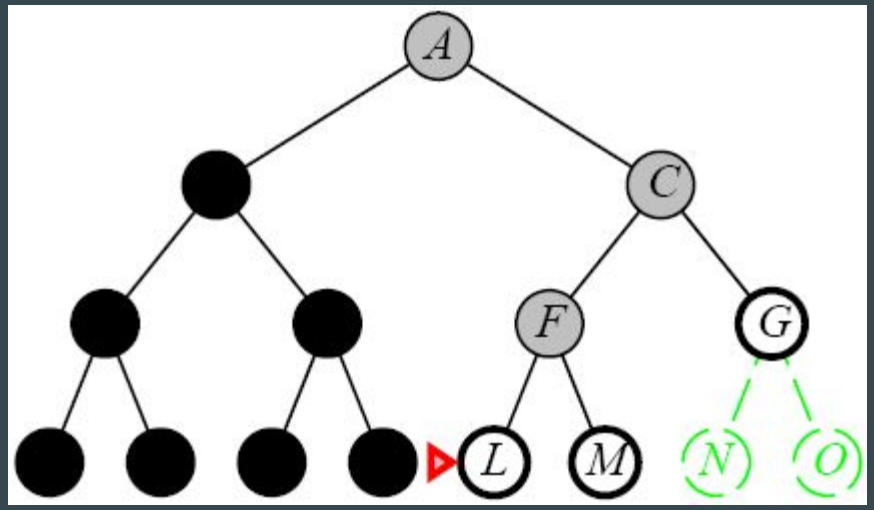
Depth-First Search



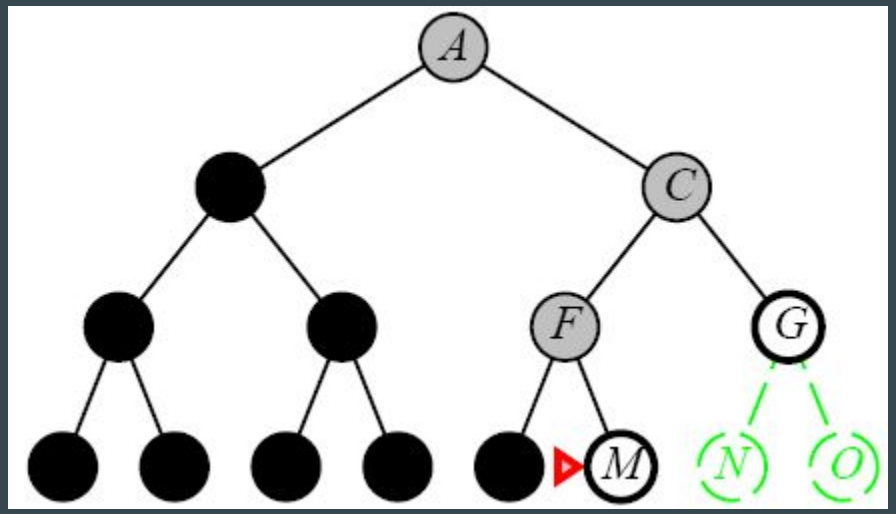
Depth-First Search



Depth-First Search



Depth-First Search



Depth-First Search

- Complete
  - No: fails in infinite-depth spaces, spaces with loops
    - Modify to avoid repeated spaces along path
  - Yes: in finite spaces
- Time
  - $O(b^m)$
  - Not great if m is much larger than d
  - But if the solutions are dense, this may be faster than breadth-first search
- Space
  - O(bm)…linear space
- Optimal
  - No

Depth-Limited Search

- A variation of depth-first search that uses a depth limit
  - Alleviates the problem of unbounded trees
  - Search to a predetermined depth **l** ("ell")
  - Nodes at depth l have no successors

- Same as depth-first search if $l = \infty$
- Can terminate for failure and cutoff

Depth-Limited Search

Recursive implementation:

```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST[problem](STATE[node]) then return node
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Depth-Limited Search

- Complete
  - Yes if l < d
- Time
  - $O(b^l)$
- Space
  - O(bl)
- Optimal
  - No if l > d

Iterative Deepening Search

- Iterative deepening depth-first search
  - Uses depth-first search
  - Finds the best depth limit
    - Gradually increases the depth limit; 0, 1, 2, … until a goal is found

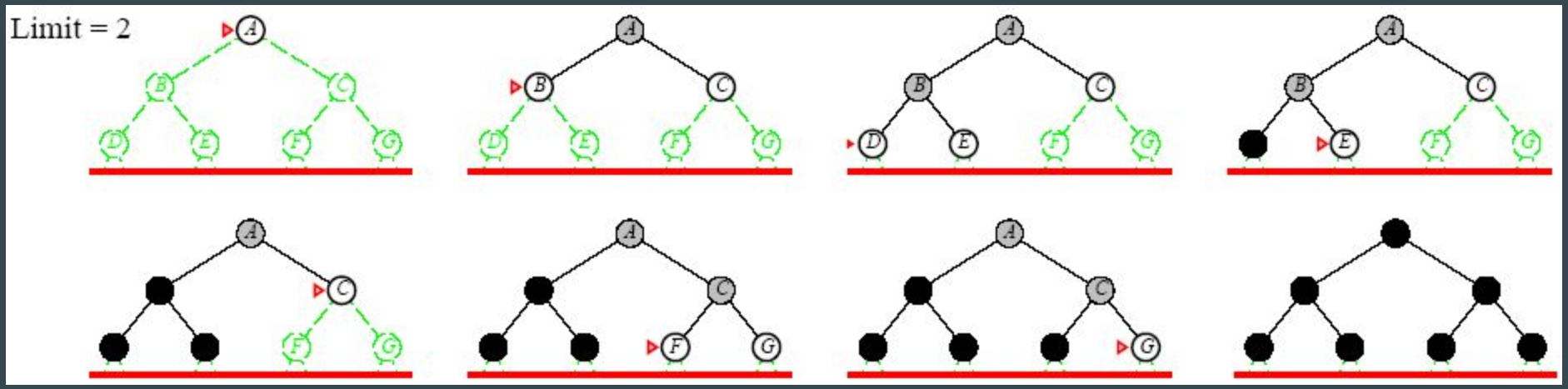
Iterative Deepening Search

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

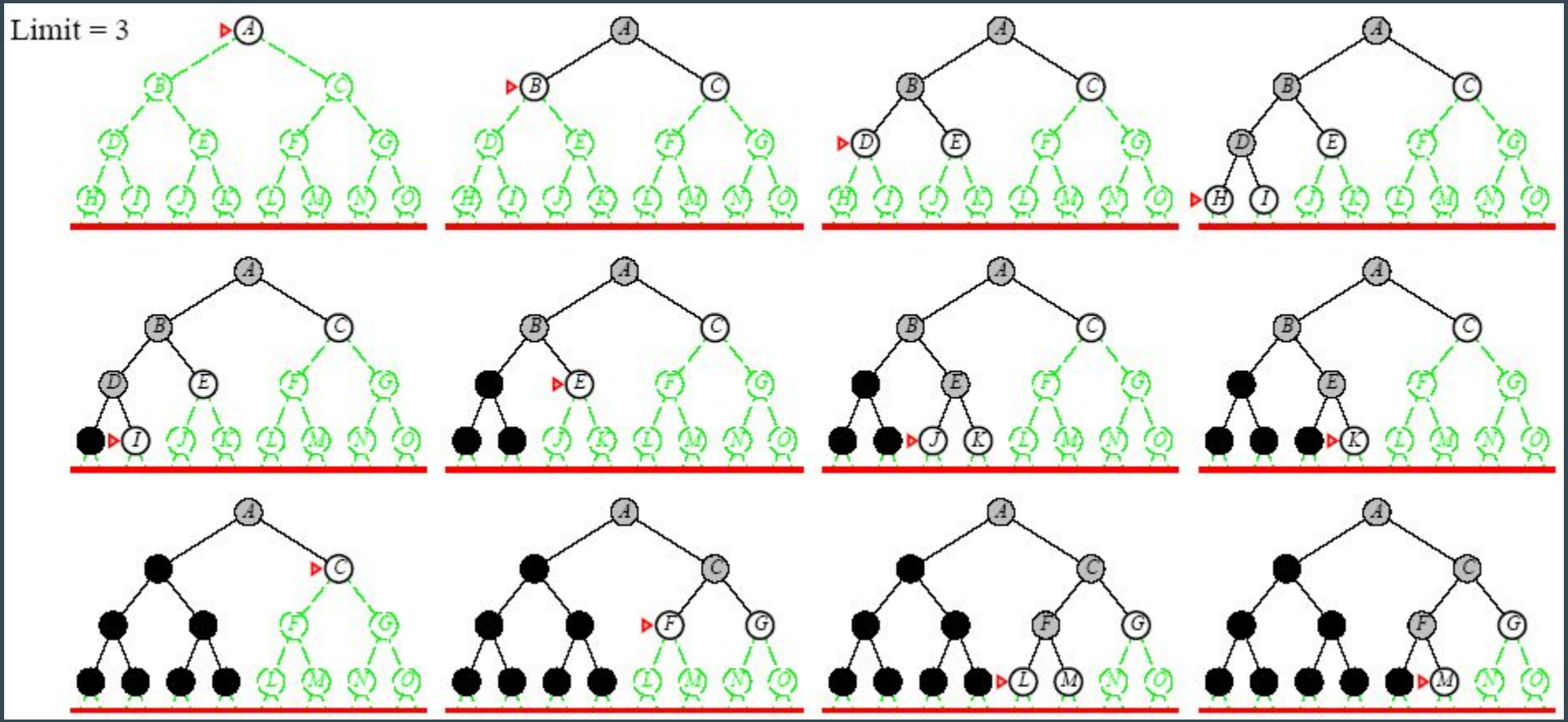
Iterative Deepening Search

Iterative Deepening Search

Iterative Deepening Search



Iterative Deepening Search



Iterative Deepening Search

- Complete
  - Yes
- Time
  - $O(b^d)$
- Space
  - O(bd)
- Optimal
  - Yes if step cost = 1
  - Can be modified to explore uniform cost tree

Lessons From Iterative Deepening Search

- Faster than BFS even though IDS generates repeated states
  - BFS generates nodes up to level d+1
  - IDS only generates nodes up to level d

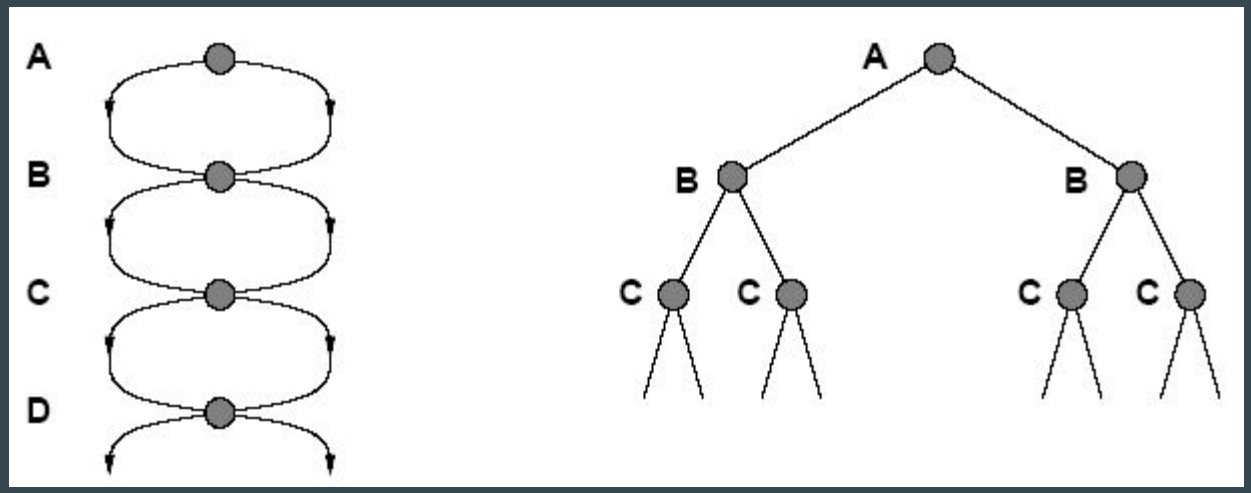
- In general, iterative deepening search is the preferred uninformed search method when there is a large search space and the depth of the solution is not known

Avoiding Repeated States

- Complication of wasting time by expanding states that have already been encountered and expanded before
  - Failure to detect repeated states can turn a linear problem into an exponential one

- Sometimes, repeated states are unavoidable
  - Problems where the actions are reversable
    - Route finding
    - Sliding blocks puzzles

Avoiding Repeated States



State Space

Search Tree

Avoiding Repeated States

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure

    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

CHAPTER 4

Beyond Classical Search

Prof. Shaikh Bilal Naseem
Department of CS/ IT
Somaiya Vidyavihar University

Search Algorithms So Far

- Designed to explore search space systematically:
  - keep one or more paths in memory
  - record which have been explored and which have not
  - a path to goal represents the solution

Local Search Algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

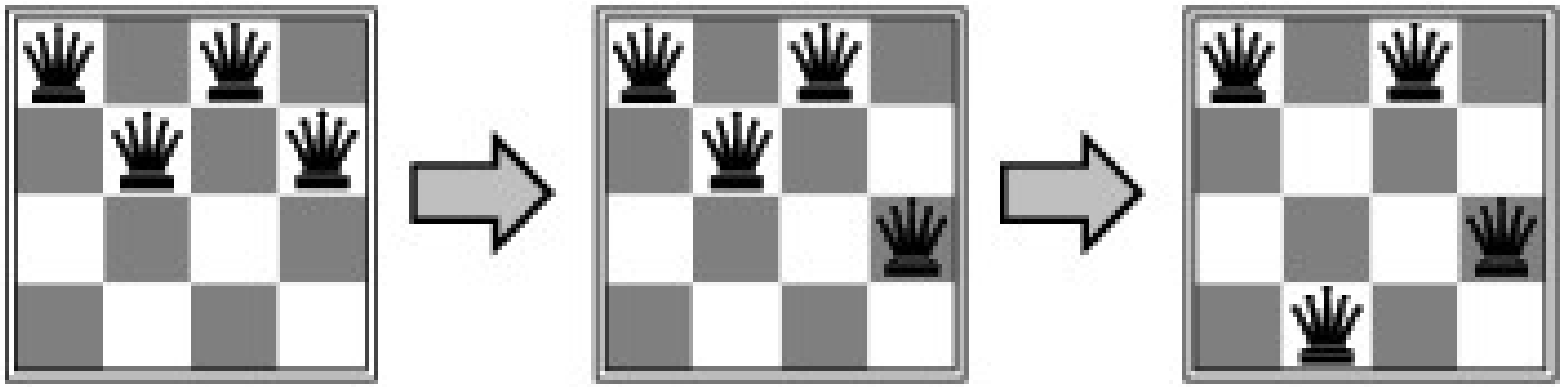
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens

- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

  - use very little memory – usually a constant amount
  - find reasonable solutions in large or infinite state spaces for which systematic solutions are unsuitable
  - useful for solving optimization problems, e.g. Darwinian evolution, no “goal test” or “path cost”

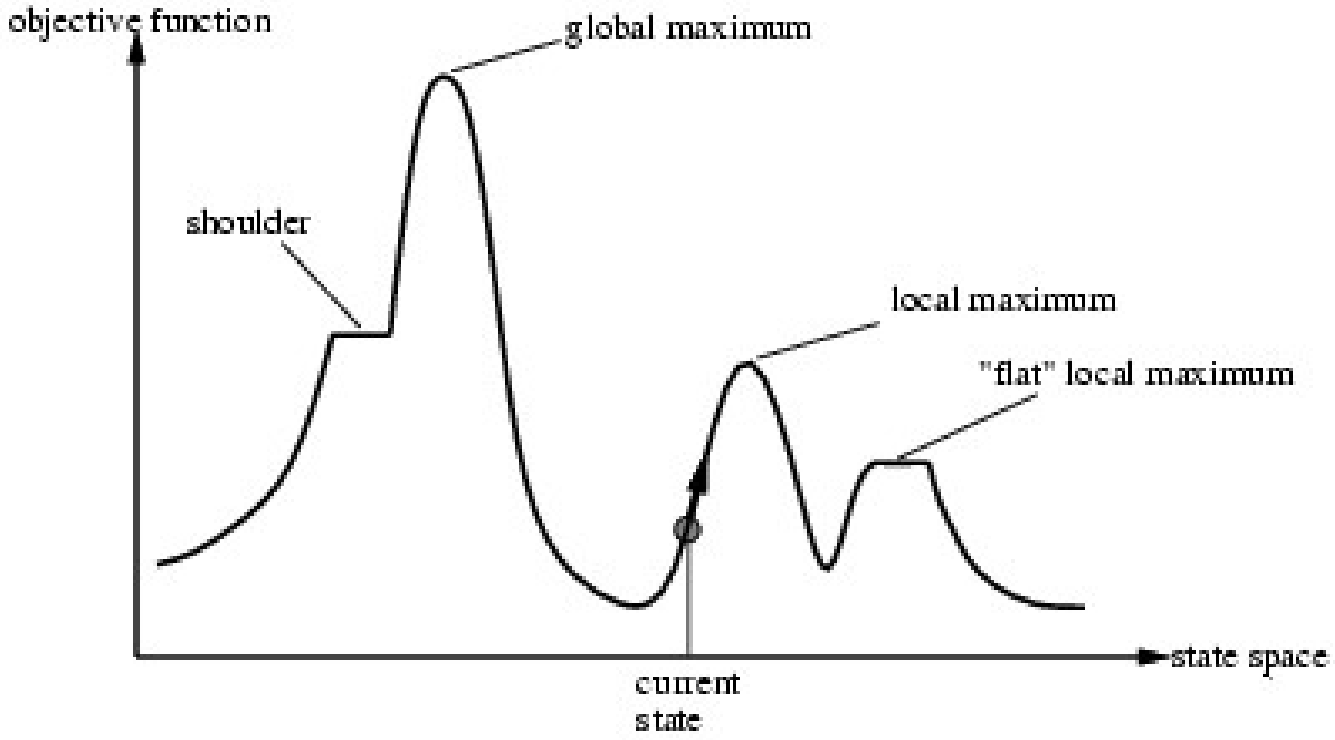
Example: n-Queen Problem

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal

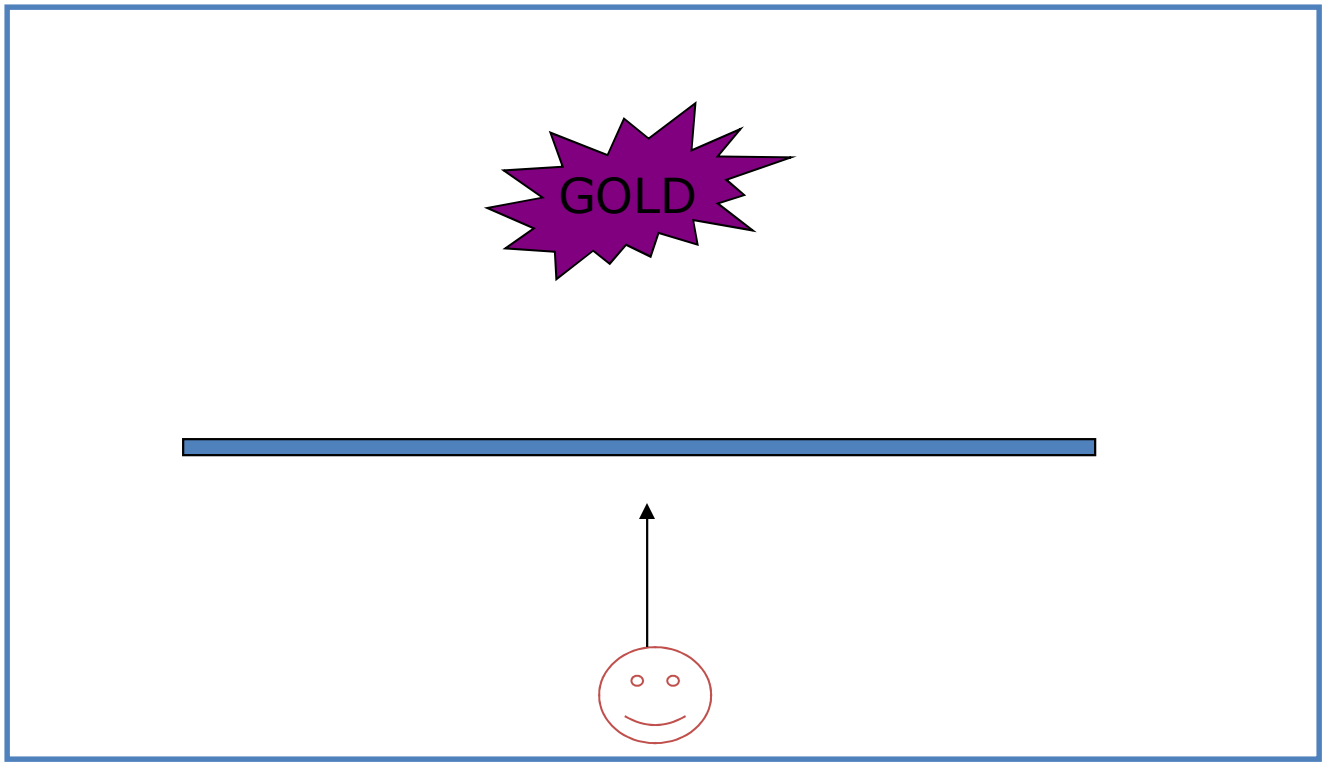


State Space Landscape

- Problem: depending on initial state, can get stuck in local maxima/minima

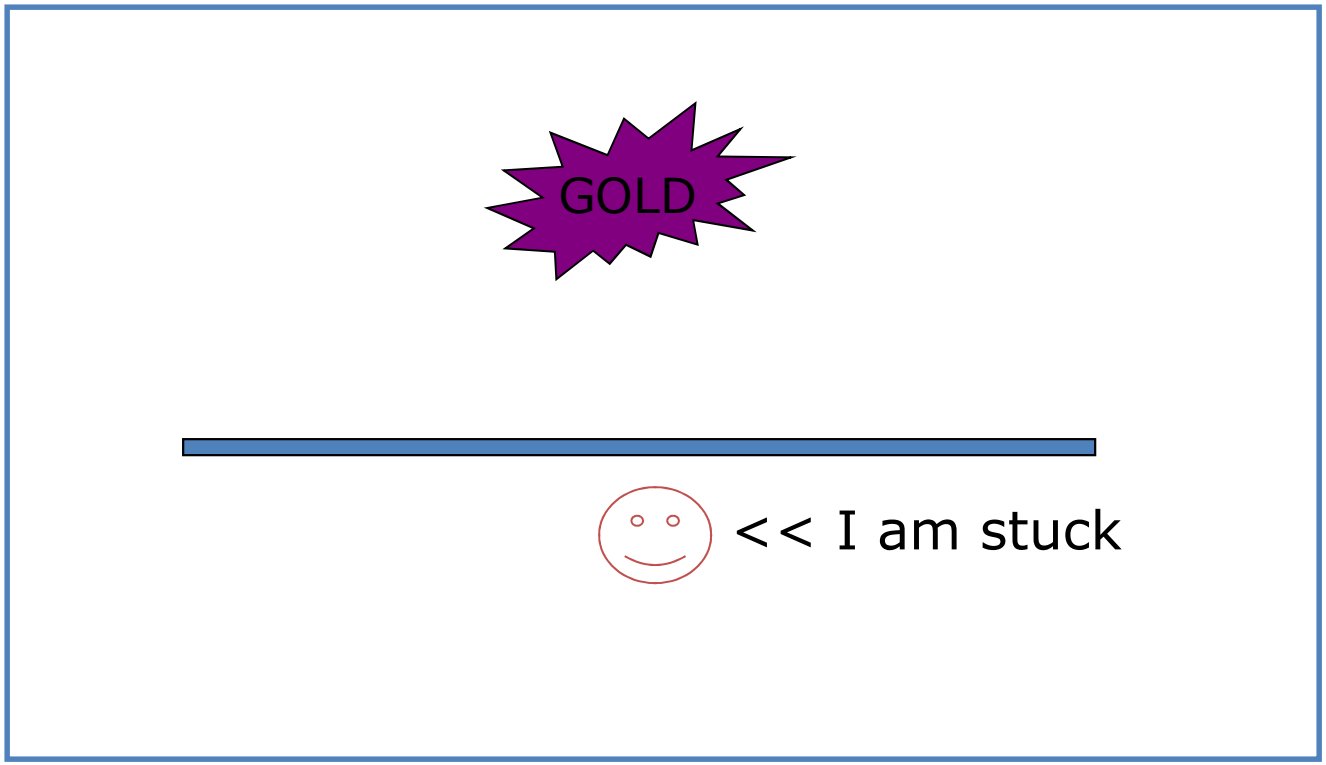


Example: Initial State



Assume the objective function measures the straight-line distance

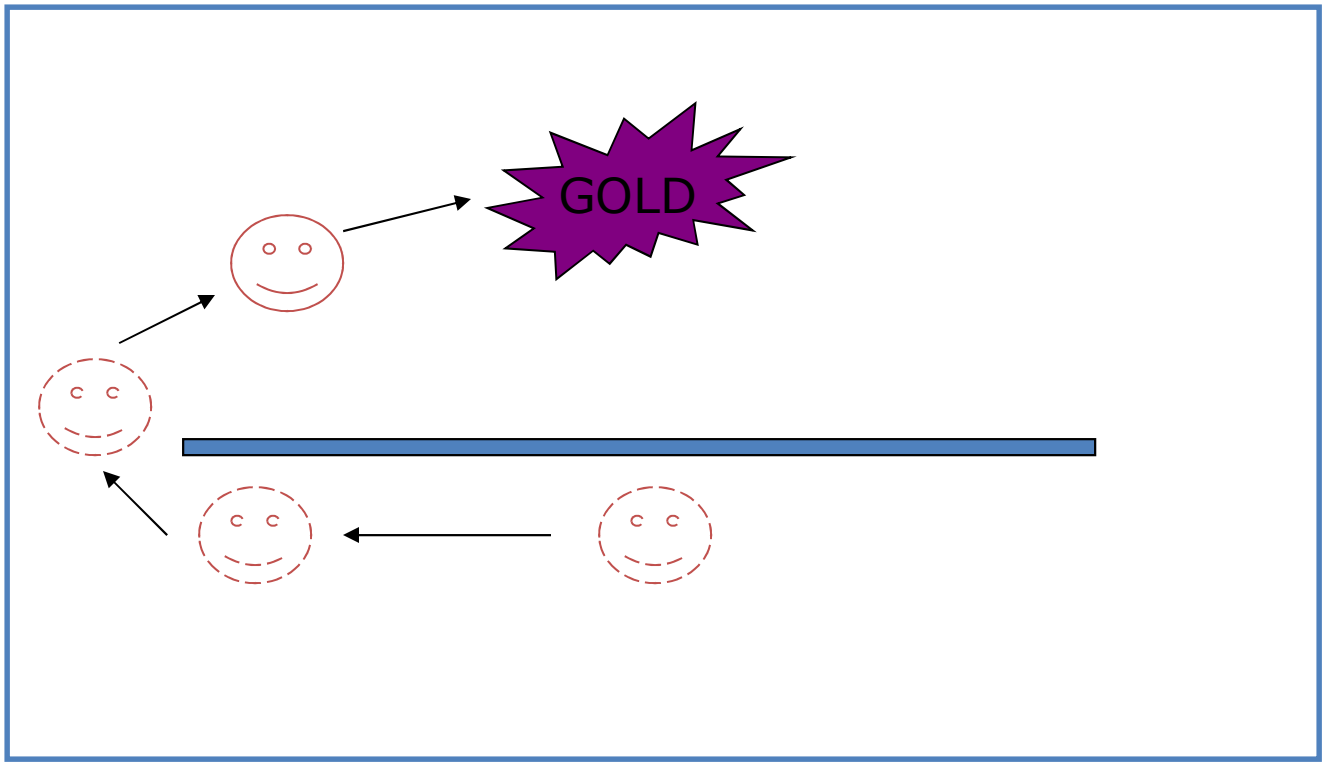
Example: Local Minima



Assume the objective function measures the straight-line distance

Example: A Plausible Solution

Making some "bad" choices is actually not that bad



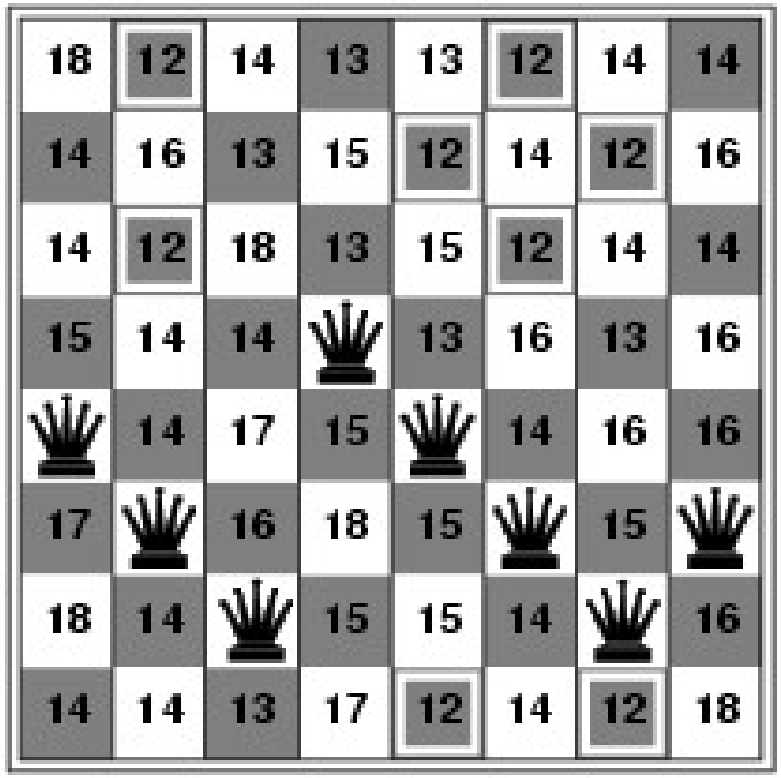
Assume the objective function measure the straight-line distance

Hill-Climbing Search

- "Like climbing Everest in thick fog with amnesia"

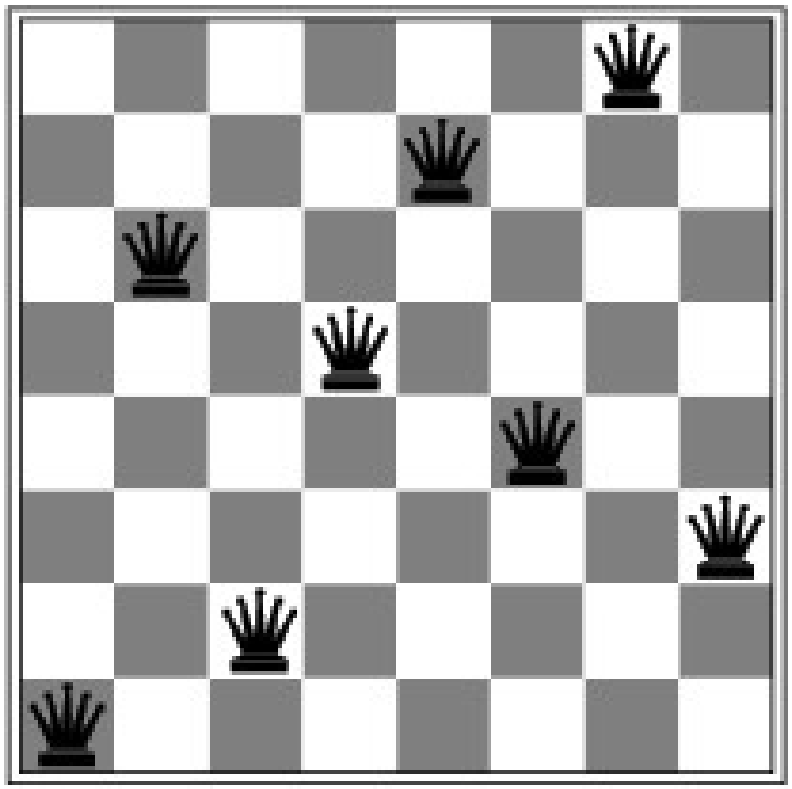
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Example: 8-Queen



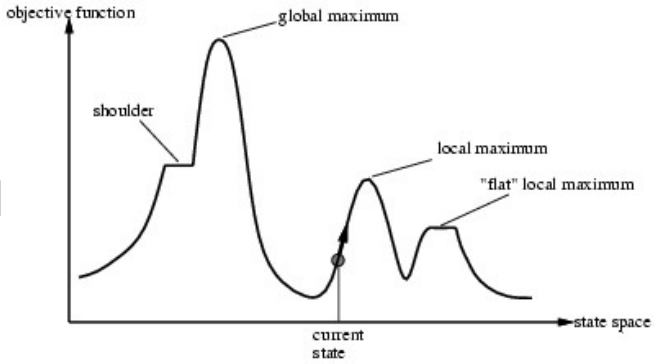
- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h* = 17 for the above state

Example: 8-Queen



- A local minimum with $h = 1$

More on Hill Clim



- Complete? Optimal?
- Hill climbing is sometimes called greedy local search
- Although greedy algorithms often perform well, hill climbing gets stuck when:
  - Local maxima/minima
  - Ridges
  - Plateau (shoulder or flat local maxima/minima)

- The steepest-ascent hill climbing solves only 14% of the randomly-generated 8-queen problems with an avg. of 4 steps
- Allowing sideways move raises the success rate to 94% with an avg. of 21 steps, and 64 steps for each failure

Variants of Hill Climbing

- Stochastic hill climbing:
  - chooses at random from among uphill moves
  - converges more slowly, but finds better solutions in some landscapes
- First-choice hill climbing:
  - generate successors randomly until one is better than the current
  - good when a state has many successors
- Random-restart hill climbing:
  - conducts a series of hill climbing searches from randomly generated initial states, stops when a goal is found
  - It's complete with probability approaching 1

More on Random-Restart Hill Climbing

- Assume each hill climbing search has a probability p of success, then the expected number of restarts required is 1/p
- For 8-queen problem, p = 14%, so we need roughly 7 iterations to find a goal
- Expected # of steps = cost_to_success + (1-p)/p * cost_to_failure

- Random-restart hill climbing is very effective for n-queen problem
- 3 million queens can be solved < 1 min

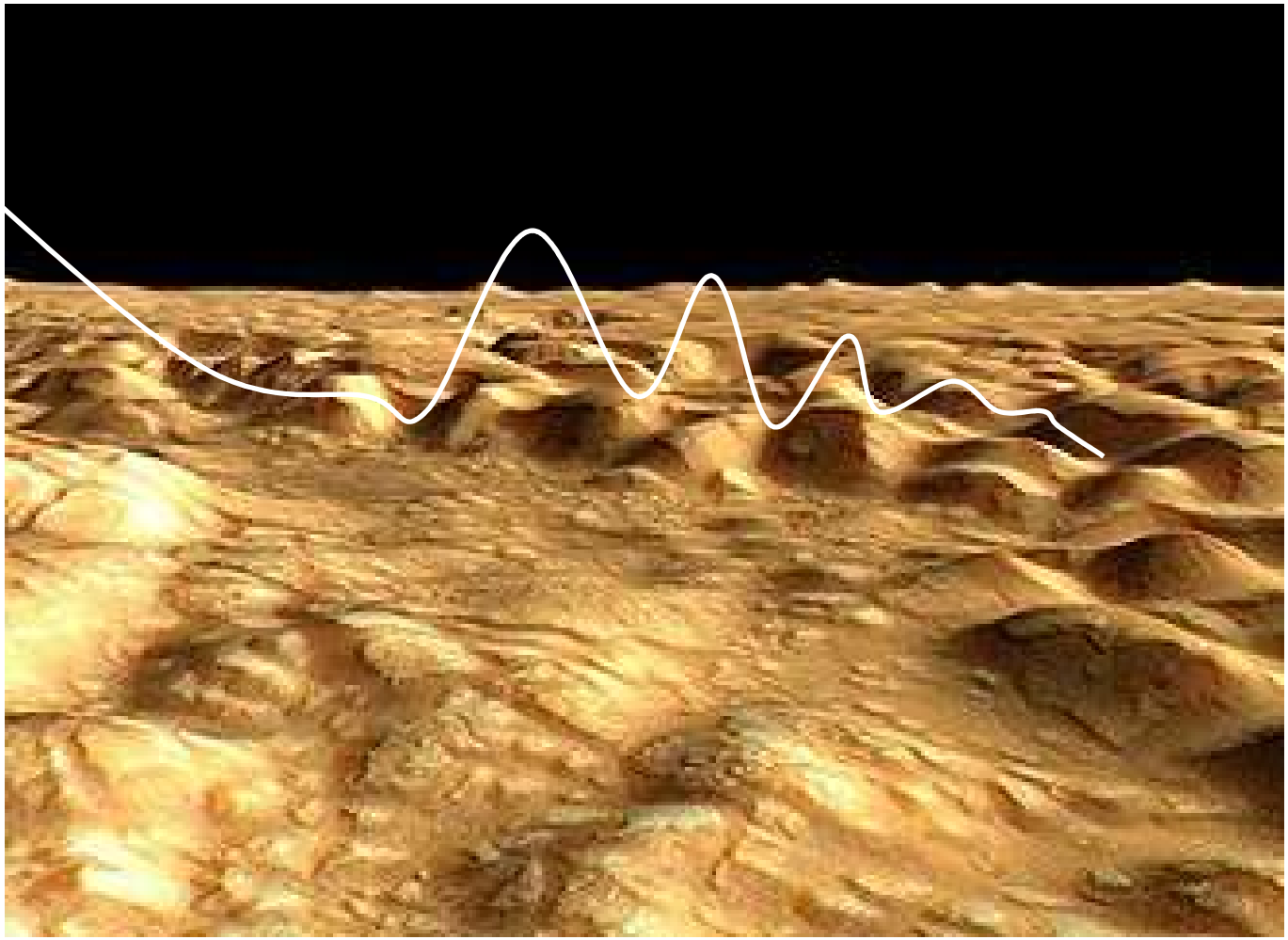
Some Thoughts

- NP-hard problems typically have an exponential number of local maxima/minima to get stuck on
- A hill climbing algorithm that never makes “downhill” (or “uphill”) moves is guaranteed to be incomplete
- A purely random walk – moving to a successor chosen uniformly at random – is complete, but extremely inefficient
- What should we do?
- Simulated annealing (P115)

What is Simulated Annealing?

- The process used to harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low-energy crystalline state

Ping-Pong Ball Example



Simulated Annealing

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

Analysis of Simulated Annealing

- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc.

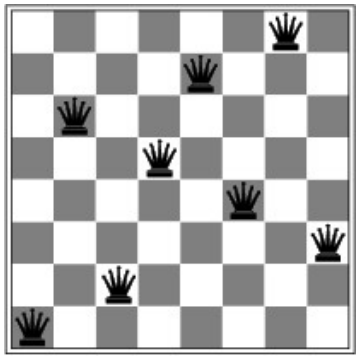
- An example: http://foghorn.cadlab.lafayette.edu/fp/fpIntro.html

Local Beam Search

- Idea:
  - Keep track of *k* states rather than just one
  - Start with *k* randomly generated states
  - At each iteration, all the successors of all *k* states are generated
  - If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat
- Is it the same as running k random-restart searches?
- Useful information is passed among the *k* parallel search threads
- Stochastic beam search: similar to natural selection, offspring of a organism populate the next generation according to its fitness

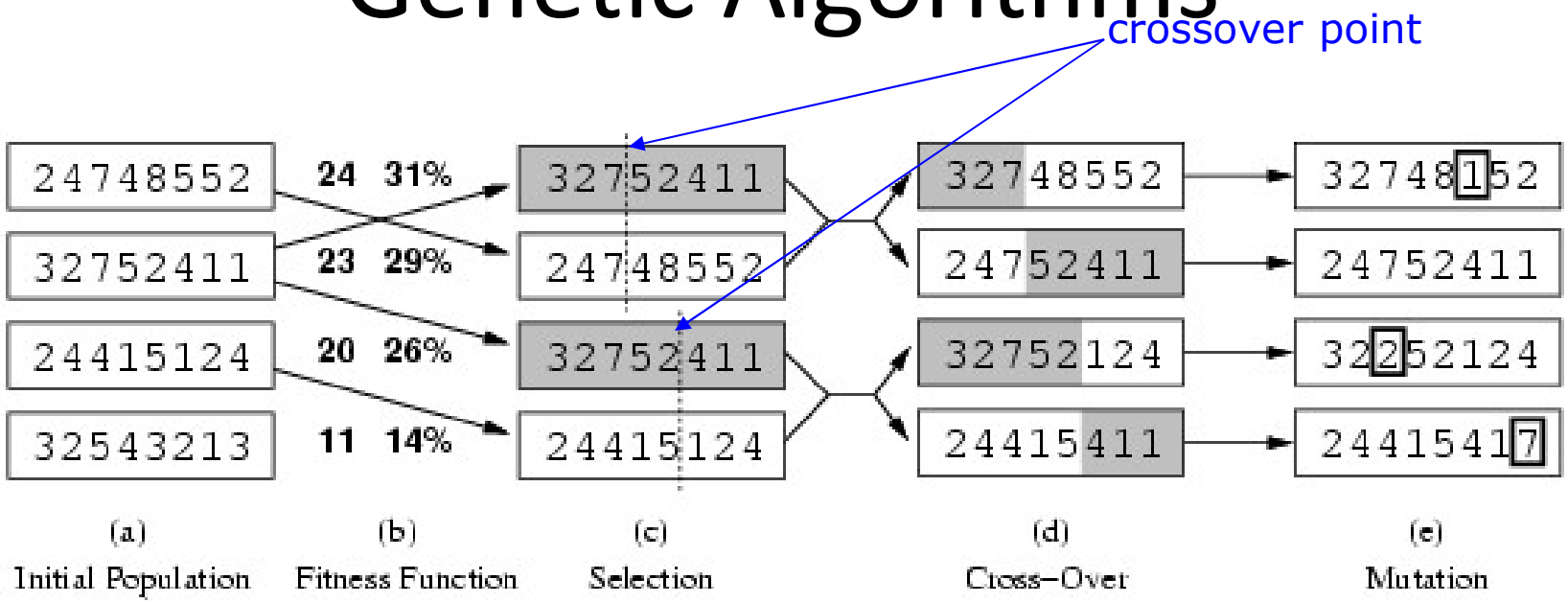
Genetic Algorithms

- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)



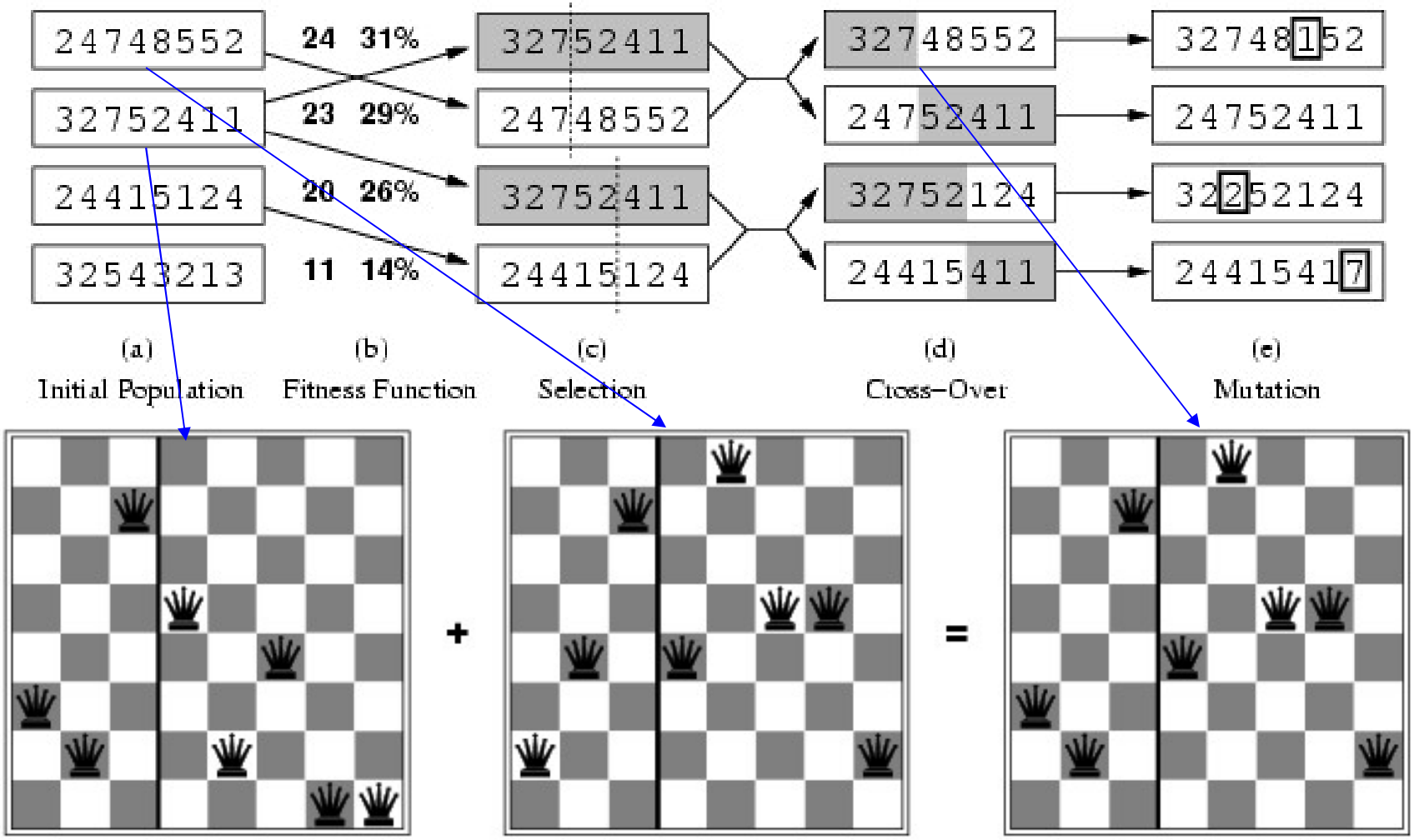
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s or digits) 16257483
- Evaluation function (fitness function). Higher values for better states
- Produce the next generation of states by selection, crossover, and mutation

Genetic Algorithms



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Example: 8-Queen



Example: TSA

- http://www.obitko.com/tutorials/genetic-algorithms/

More on Genetic Algorithms

- Genetic algorithms combine an uphill tendency with random exploration and exchange of information among parallel search threads
- Advantages come from “crossover”, which raise the level of granularity

A Genetic Algorithm

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    loop for i from 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Exercise

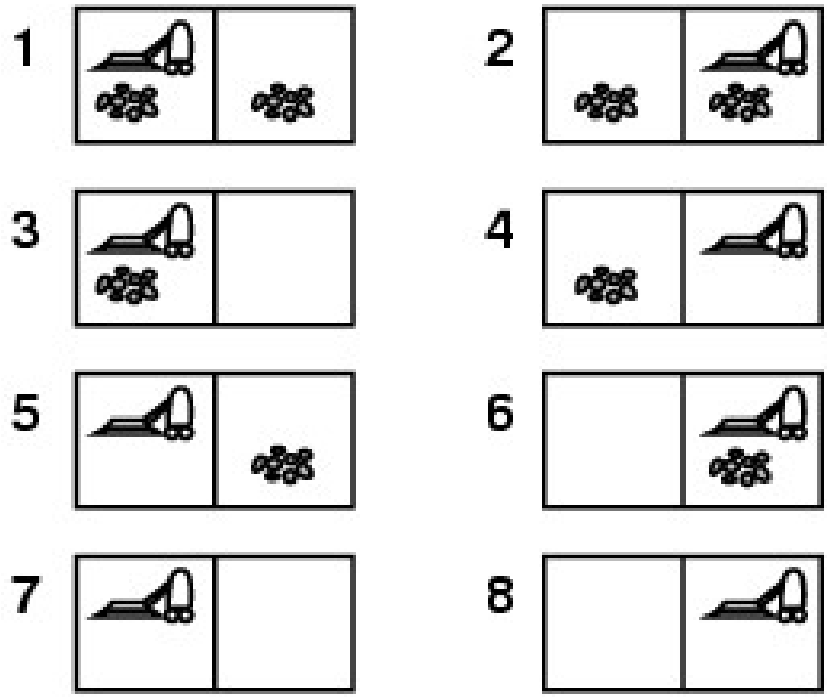
- 4.11 Give the name of the algorithm that results from each of the following special cases:
  - Local beam search with k = 1
  - Local beam search with one initial state and no limit on the number of states retained
  - Simulated annealing with T = 0 at all times (and omitting the termination test)
  - Genetic algorithm with population size N = 1

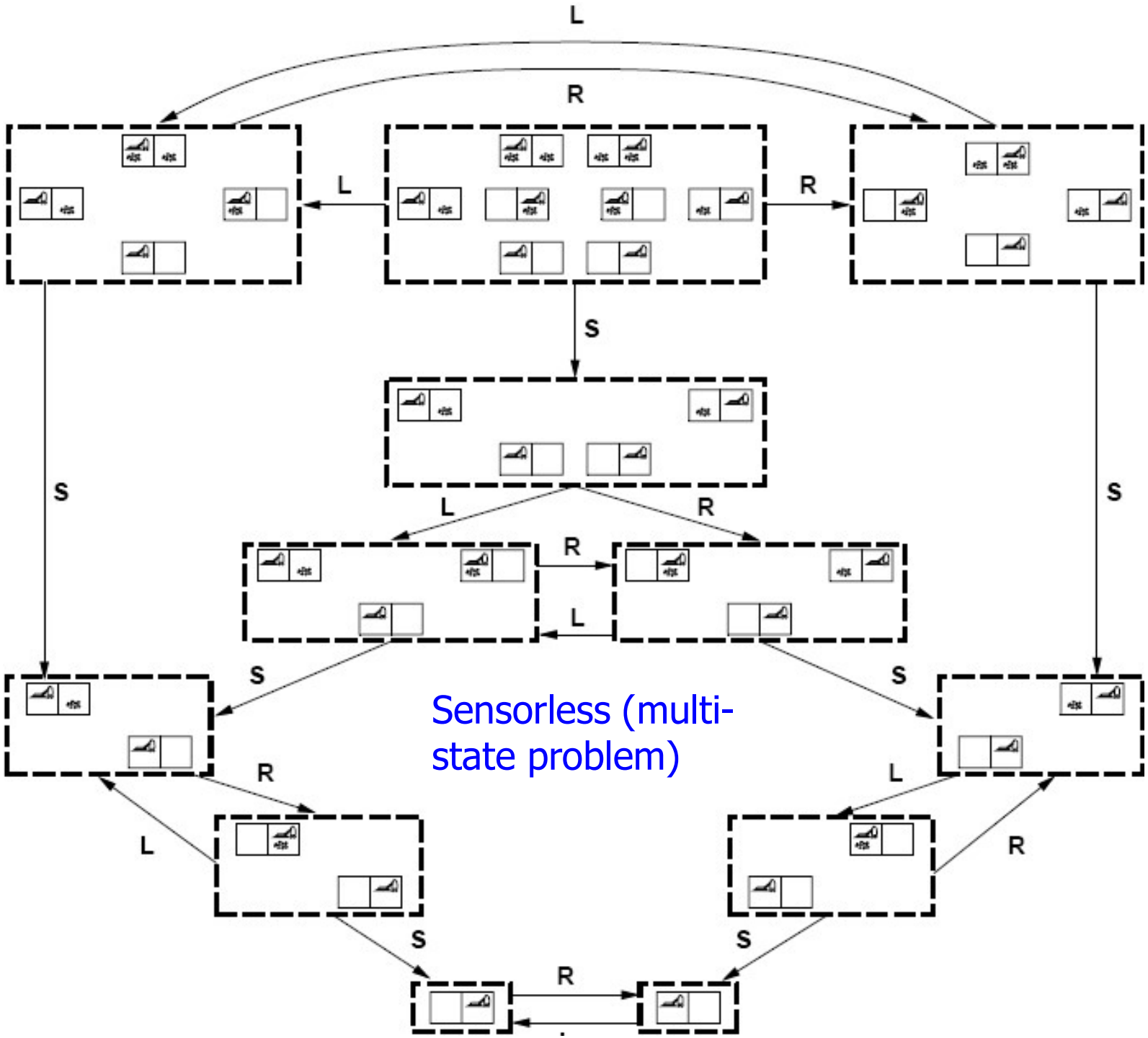
Searching With Partial Information

- We have covered: Deterministic, fully observable → single-state problem
  - agent knows exactly which state it will be in
  - solution is a sequence
- Deterministic, non-observable → multi-state problem
  - Also called sensorless problems (conformant problems)
  - agent may have no idea where it is
  - solution is a sequence
- Nondeterministic and/or partially observable → contingency problem
  - percepts provide new information about current state
  - often interleave search, execution
  - solution is a tree or policy
- Unknown state space → exploration problem ("online")
  - states and actions of the environment are unknown

Example: Vacuum World

- Single-state, start in #5. Solution?
  - [Right, Suck]
- Multi-state, start in #[1, 2, …, 8]. Solution?
  - [Right, Suck, Left, Suck]

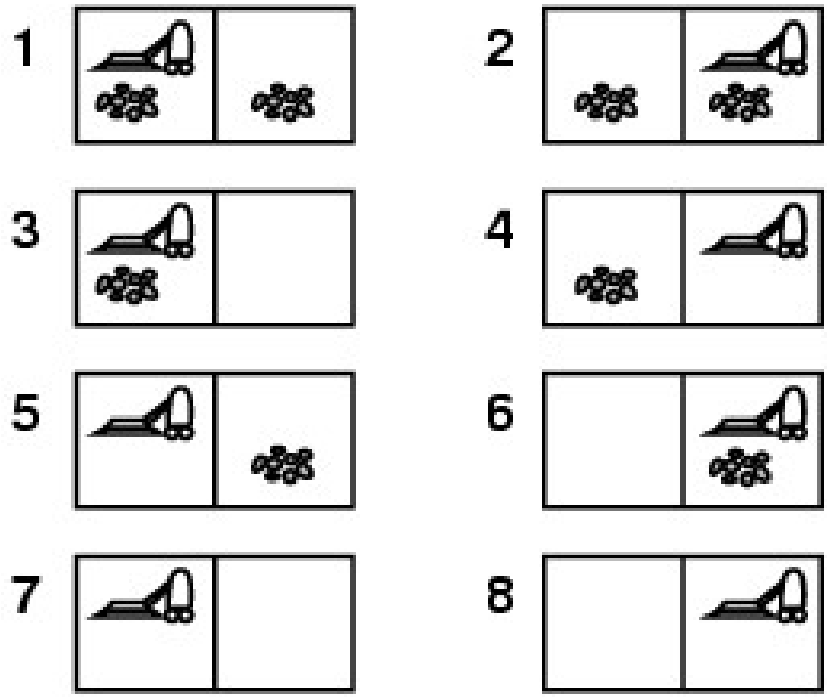




Sensorless (multi-state problem)

Contingency Problem

- Contingency, start in #5 & 7.
  - *Nondeterministic*: suck may dirty a clean carpet
  - *local sensing*: dirt, location only at current location
  - Solution?
  - Percept: [Left, Clean] → [Right, **if** dirty **then** Suck]



Exam 2

- Closed book and closed notes
- Coverage:
  - Informed Search
  - Local Search
- Format:
  - True/False
  - Short-Answer
  - Problem Solving

Informed Search

- Understand what is informed search
- Understand best-first, greedy, and A*
- Understand the optimality and completeness of each search
- Be able to draw search tree
- Understand the heuristic function in A*, under what condition will A* be optimal?
  - Admissible heuristic
  - Consistent heuristic

Local Search

- Hill climbing (HC)
  - Steepest HC, stochastic HC, 1st choice HC, random-restart HC
- HC with side-moves allowed
- Simulated annealing
- Local beam search
- Genetic algorithms

- Understand each search strategy
- Understand how to define the objective function for local search