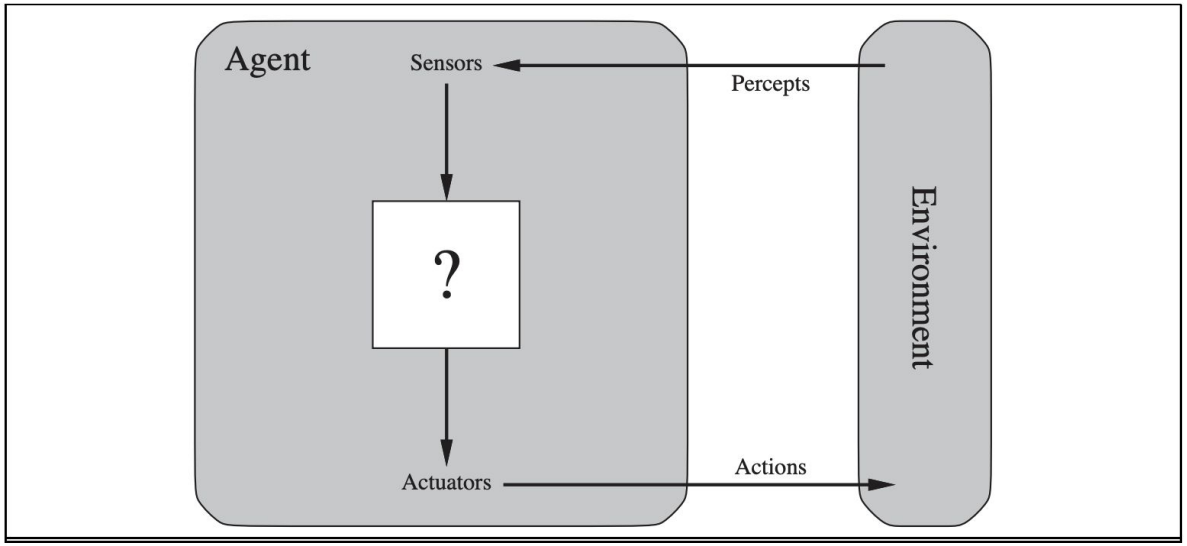
AI Course

Intelligent Agents and Search

Dr. Mürsel Taşgın

Agents

Intelligent Agents

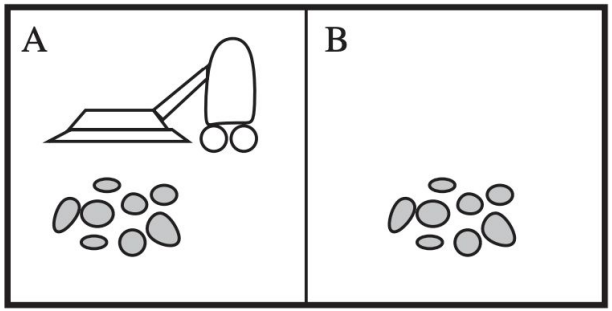


percept: the agent's perceptual inputs at any given instant

percept sequence: the complete history of everything the agent has ever perceived

agent function maps any given percept sequence to an action

Intelligent Agents - *Vacuum cleaner*



Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[B, Clean]$	*Left*
$[B, Dirty]$	*Suck*
$[A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Dirty]$	*Suck*
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Clean], [A, Dirty]$	*Suck*
⋮	⋮

A rational agent is one that does the right thing—conceptually speaking, every entry in the table for the agent function is filled out correctly. Obviously, doing the right thing is better than doing the wrong thing, but what does it mean to **do the right thing**?

Types of agents

Simple reflex agent

- No “state” or memory
- Reacts to current input according to its program (rules of the form “if condition then action”)

Model-based agent

- Uses an explicit knowledge base to model the environment
  - How does the environment evolve independently
  - How does the agent affect the environment
- Exhibits “understanding” of its input by relating it to prior knowledge
- Reacts according to rules
  - Conditions may be complex and require inference to evaluate

Types of agents

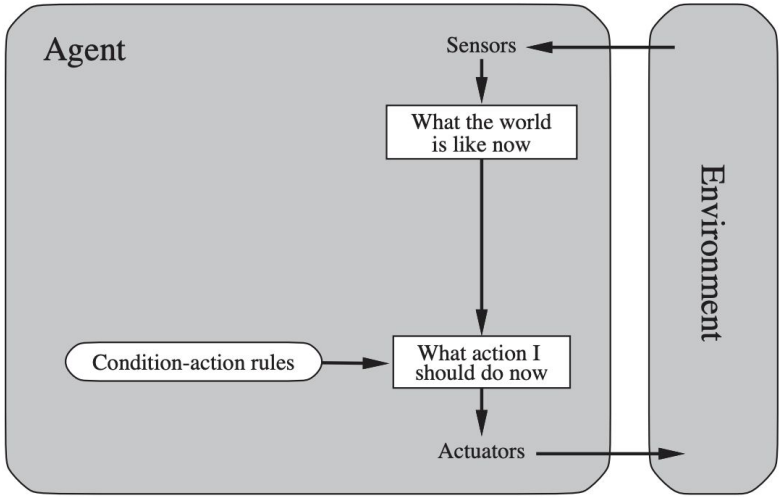
Planning agents *(goal-based and utility-based agents)*

- Explicitly represent their own goals and/or preferences (“utilities”) and can reason about them (i.e., planning)
- **Exhibit autonomy:** Actions do not follow directly from rule-based lookup

Learning agents

- **Supervised learning:** Learn from positive and negative examples
- **Reinforcement learning:** Learn from experience to improve its outcomes

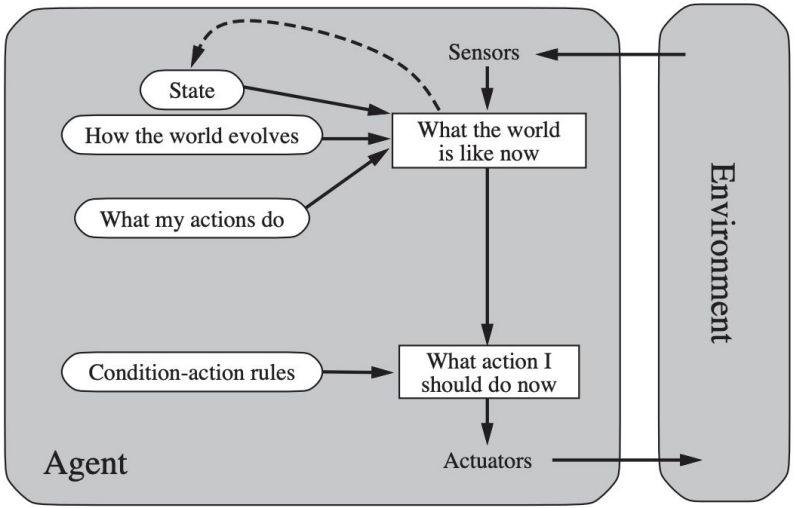
Declarative simple reflex agents



function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *rules*, a set of condition–action rules

  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

Model-based reflex agents

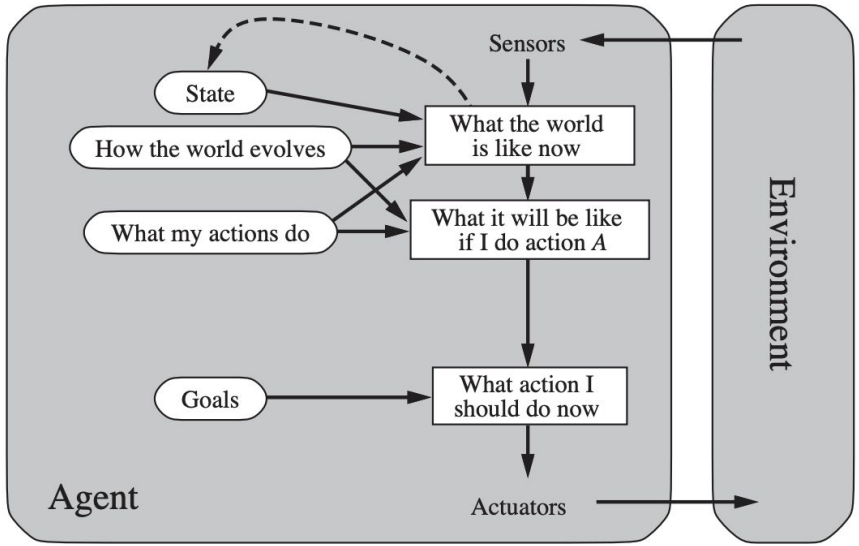


Model: The definition of the environment, e agent's model of how the world works.

function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *state*, the agent's current conception of the world state
    model, a description of how the next state depends on current state and action
    rules, a set of condition–action rules
    action, the most recent action, initially none

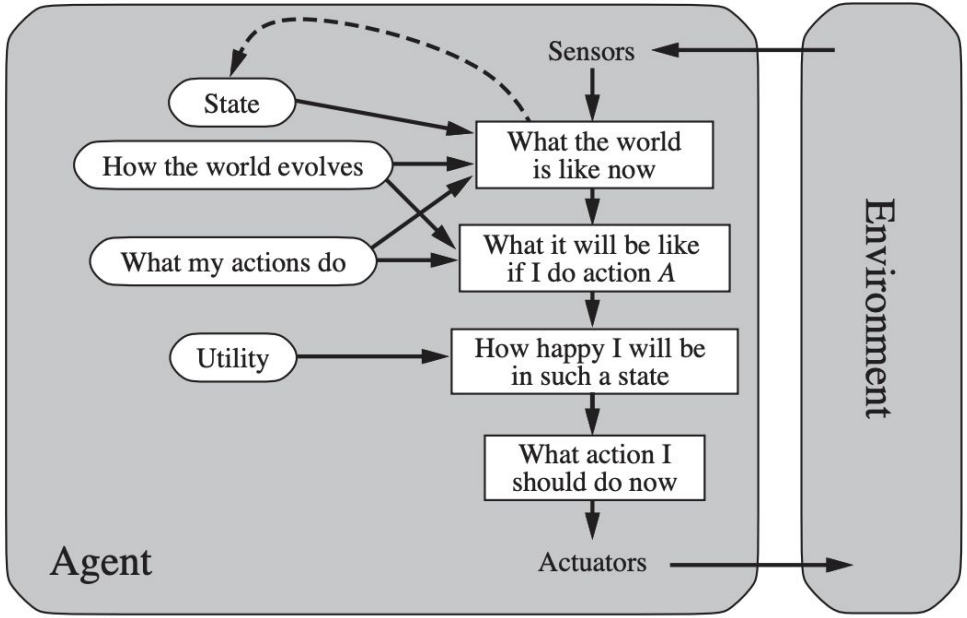
  state ← UPDATE-STATE(*state*, *action*, *percept*, *model*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

Goal-based agents



The agent needs some sort of goal information that describes situations that are desirable, for example, being at the passenger's destination.

Utility-based agents

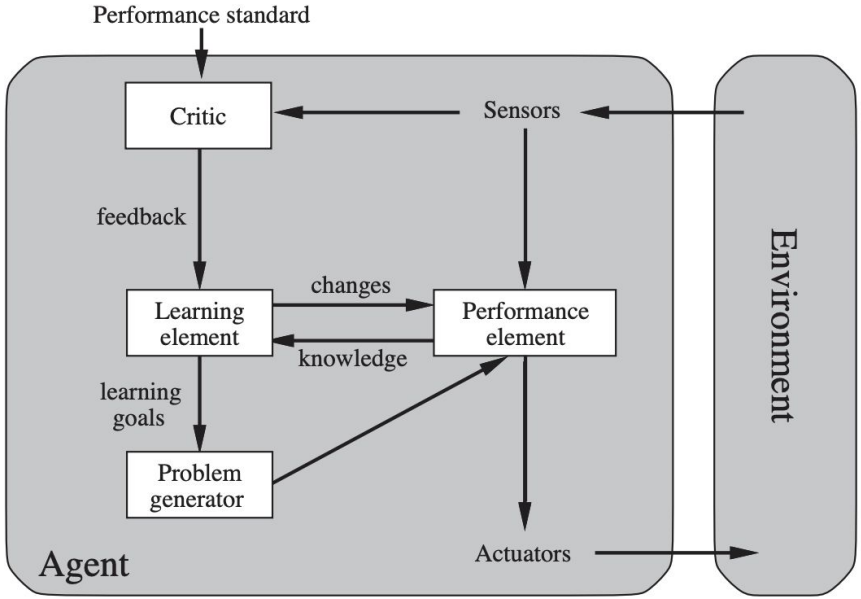


The agent's preferences are captured by a **utility function**, U(s), which assigns a single number to express desirability of a state.

Rational agent tries to select the best action to maximize the utility

Goals alone are not enough to generate high-quality behavior in most environments. For example, many action sequences will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others.
Goals just provide a crude binary distinction between "**happy**" and "**unhappy**" states. A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent. Because "happy" does not sound very scientific, economists and computer scientists use the term utility instead

Learning agents



A learning agent can be divided into four conceptual components, as shown in figure.

The most important distinction is between the learning element, which is responsible for making improvements, and the performance element, which is responsible for selecting external actions.

The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.

The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.

Drawbacks of simple reflex agents

- HUGE rule base, time consuming to build by hand
- What if more than one condition is satisfied?
- Inflexible (no adaptation or learning)

Representing agent knowledge

- **Q:** What formal language(s) can we use to represent
  1. Current facts about the state of the world?
  2. A model of how the world behaves?
  3. A model of the effects of actions that the agent can perform?
  4. The production rules that specify agent behavior?

- **A:** Formal logic
  - Syntax and semantics are well understood
  - Computational tractability known for important subsets (e.g., Horn clause logic)

How do we determine success of the agent?

1. Getting the “*right answer*”?
2. The Turing Test *(or modified versions)?*
3. Having a good outcome? *(using some “utility” function)*
4. Know it when we see it?

Analyzing agent performance

- **Rational agent** is one that does the "right" thing
  - Must define a **performance measure**
    - Costs (penalties) and rewards
  - Chooses an action that maximizes expected score
- Rationality depends on
  1. Success criterion defined by performance measure
  2. "Behavior" of the environment (e.g., can a clean square get dirty again?)
  3. Possible actions
  4. Percept sequence
- Autonomy
  - Rational agents require **learning** to compensate for incorrect or incomplete starting knowledge

Rational Agent

Without loss of generality, "goals" specifiable by performance measure defining a numerical value for any environment history

Rational action: whichever action maximizes the expected value of the performance measure given the percept sequence to date

Rational $\neq$ omniscient
Rational $\neq$ clairvoyant
Rational $\neq$ successful

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent's prior knowledge of the environment.
- The actions that the agent can perform.
- The agent's percept sequence to date.

Rational Agent, Autonomy, Exploration

- **Autonomy:** Learn and act (percepts)

- **Exploration**: Learning new things, trial

- **Exploitation:** Acting based on knowledge at hand

Specifying the task environment

PEAS (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors)

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Examples

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Environments

Environment examples

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Episodic: The next episode does not depend on the actions taken in previous episodes
Sequential: The current decision could affect all future decisions

Deterministic: The next state of the environment is completely determined by the current state and the action
Stochastic: Actions are characterized by their possible outcomes

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Accessible??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Semi	Semi	No
Discrete??	Yes	Yes	Yes	No

The environment type largely determines the agent design

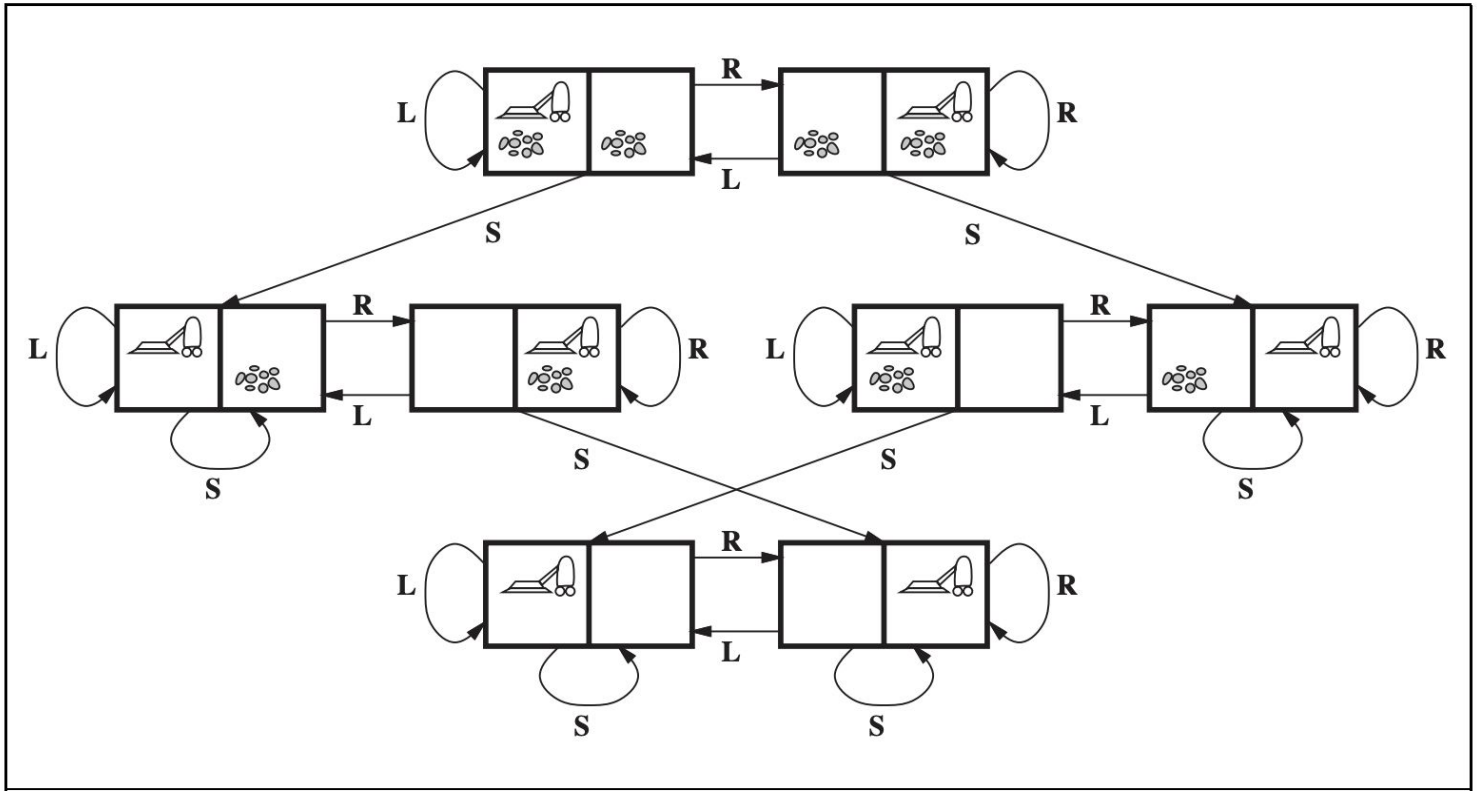
The real world is (of course) inaccessible, stochastic, sequential, dynamic, continuous

Solving problems by Search

Problem solving agents

- Goal formulation
- Problem formulation
- Environment
- Actions

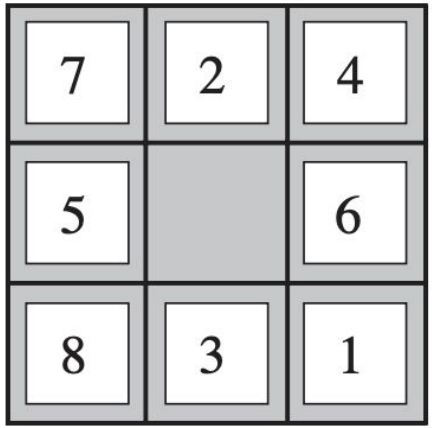
Example - *Vacuum cleaner*



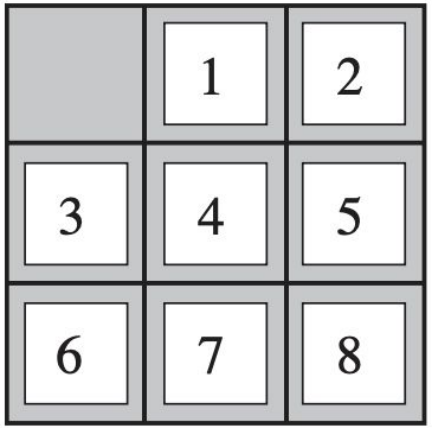
Vacuum cleaner - *Problem formulation*

- **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 *Suck*ing 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.

Example - *8-puzzle*



Start State

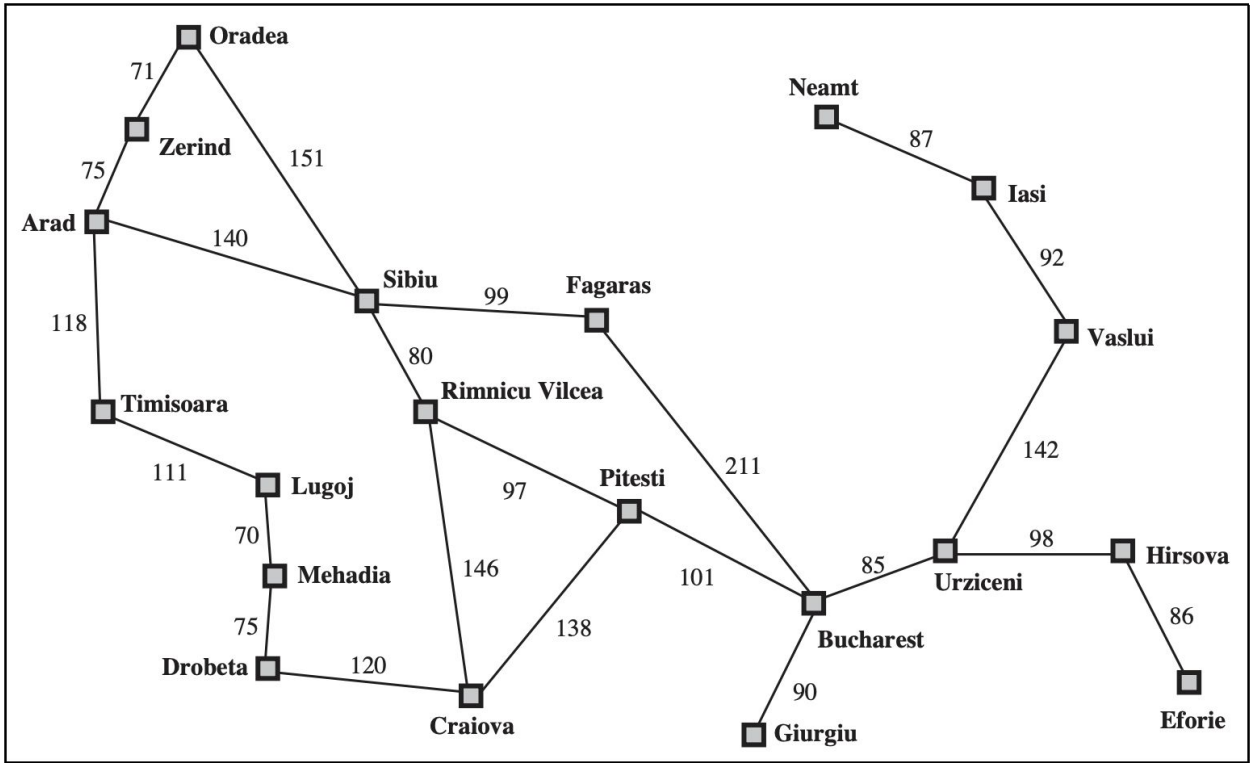


Goal State

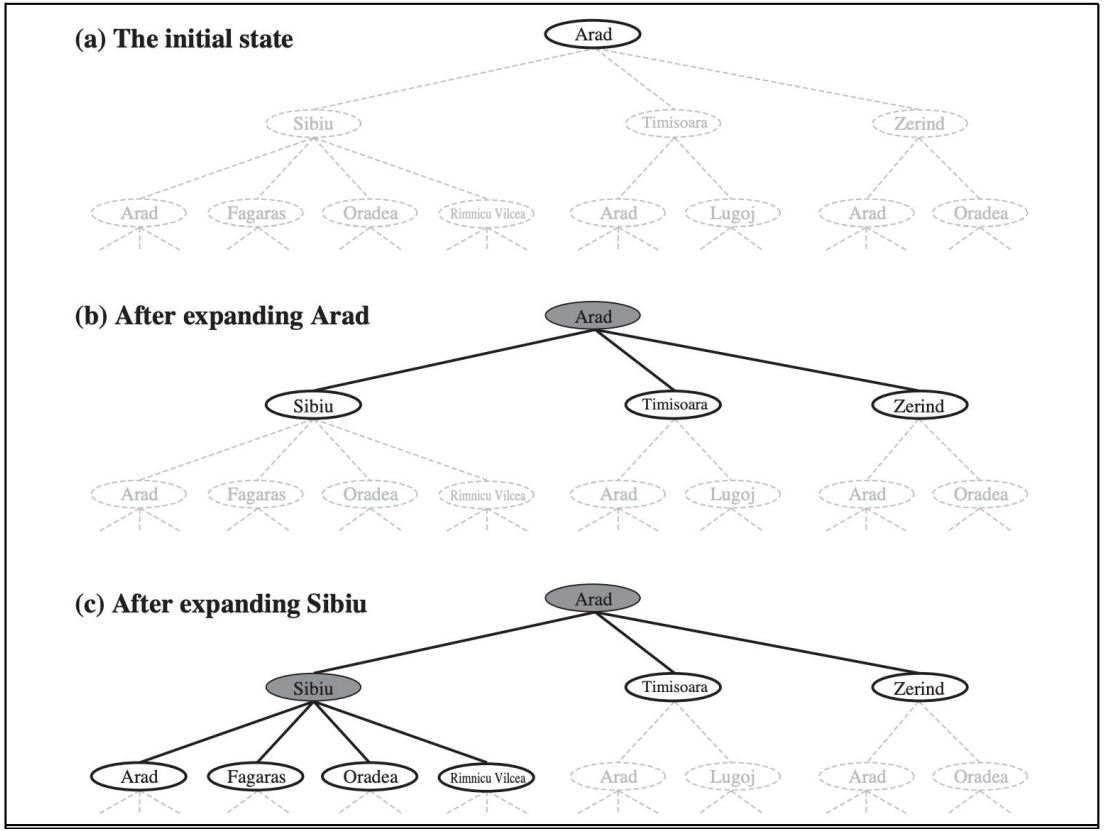
8-puzzle

- **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
- **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.

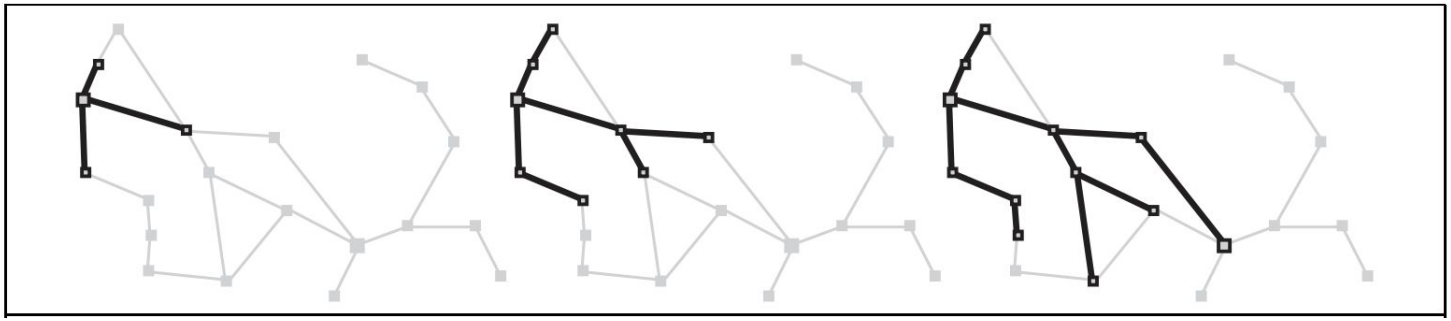
Example - Roadtrip from Arad to Bucharest



Roadtrip

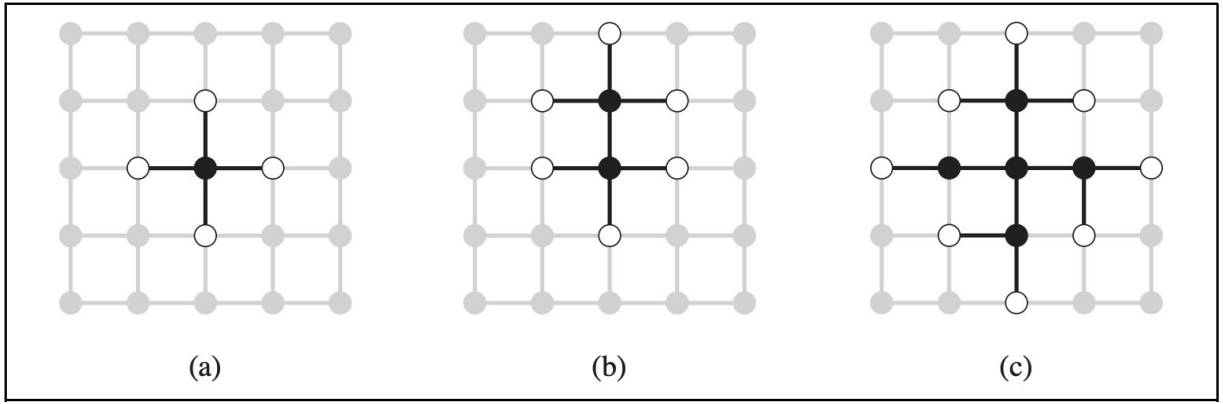

(a) The initial state
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea

Roadtrip - *Tree Search*



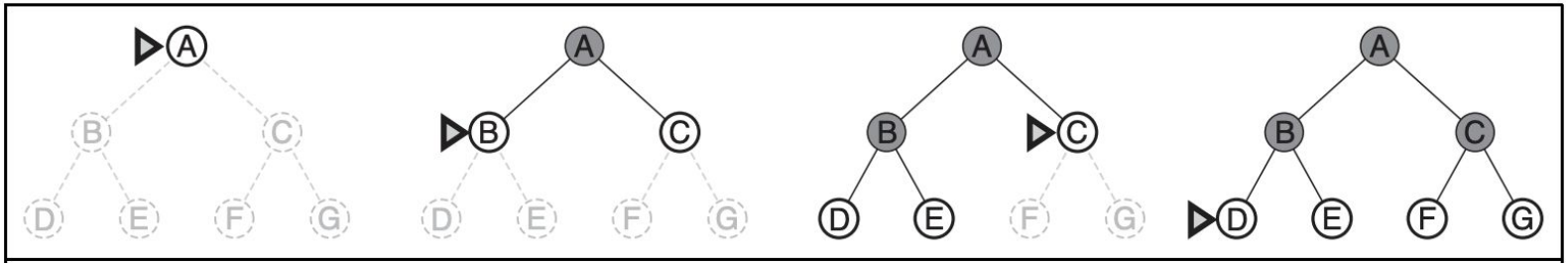
function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

Roadtrip - *Graph Search*



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Uninformed Search - *Breadth First Search*



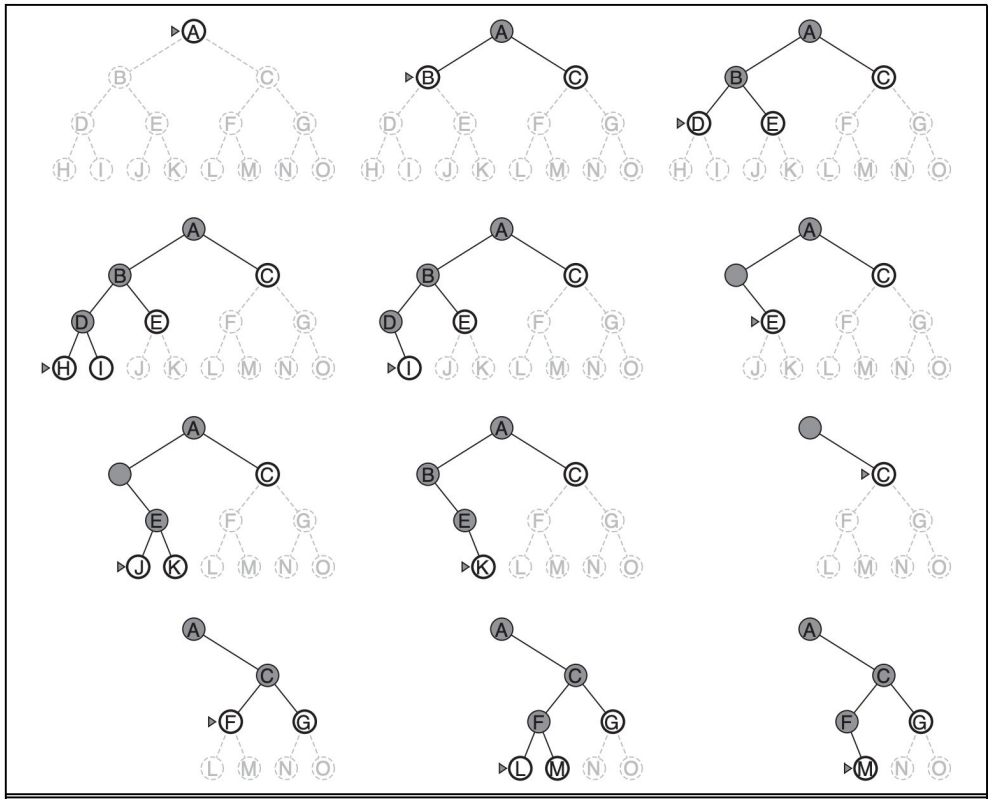
function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure

```
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)
```

Time and space complexity of BFS

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Uninformed Search - Depth first search



Informed Search (Heuristic search)

Heuristic Search

- **Heuristic** or **informed search** exploits additional knowledge about the problem that helps direct search to more promising paths.

- A **heuristic function,** *h(n)*, provides an estimate of the cost of the path from a given node to the closest goal state. Must be zero if node represents a goal state.
  - Example: Straight-line distance from current location to the goal location in a road navigation problem.

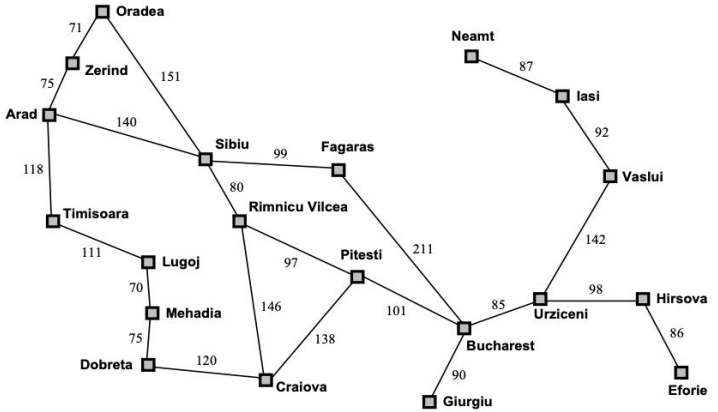
- Many search problems are NP-complete so in the worst case still have exponential time complexity; however a good heuristic can:
  - Find a solution for an average problem efficiently.
  - Find a reasonably good but not optimal solution efficiently.

Best-First Search

- At each step, best-first search sorts the queue according to a heuristic function.

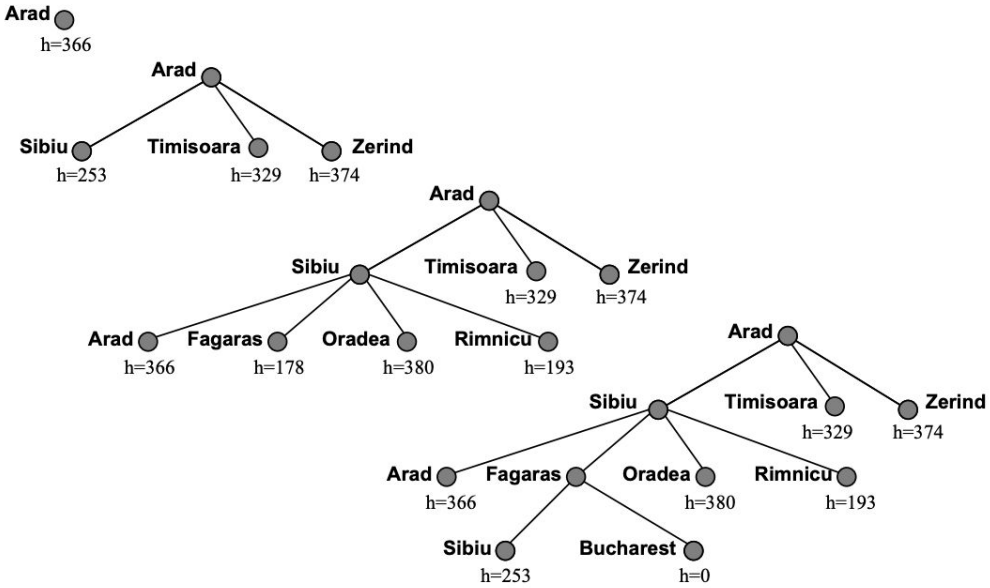
function BEST-FIRST-SEARCH(*problem,* EVAL-FN) **returns** a solution sequence
  inputs: *problem*, a problem
    Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(*problem, Queueing-Fn*)



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Best-First Example



- Does not find shortest path to goal (through Rimnicu) since it is only focused on the cost remaining rather than the total cost.

Best-First Properties

- Not complete, may follow infinite path if heuristic rates each state on such a path as the best option. Most reasonable heuristics will not cause this problem however.

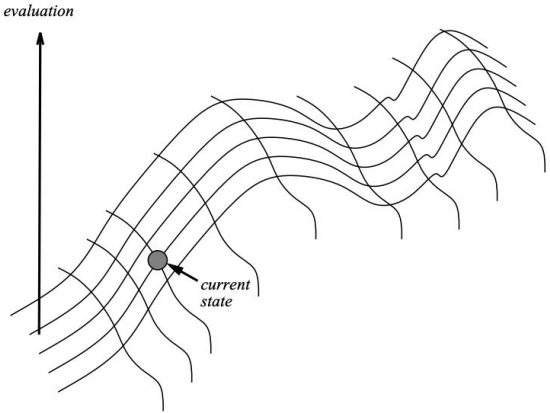
- Worst case time complexity is still $O(b^m)$ where m is the maximum depth.

- Since must maintain a queue of all unexpanded states, space-complexity is also $O(b^m)$.

- However, a good heuristic will avoid this worst-case behavior for most problems.

Hill-Climbing

- Beam search with a beam-width of 1 is called **hill-climbing**.
- Pursues locally best option at each point, i.e. the best successor to the current best node.
- Subject to local maxima, plateaux, and ridges.



Minimizing Total Path Cost: A* Search

- A* combines features of uniform cost search (complete, optimal, inefficient) with best-first (incomplete, non-optimal, efficient).

- Sort queue by estimated total cost of the completion of a path.

  $f(n) = g(n) + h(n)$

- If the heuristic function always underestimates the distance to the goal, it is said to be **admissible.**

- If h is admissible, then $f(n)$ never overestimates the actual cost of the best solution through n.

Other possible search options

- Genetic algorithm
- Djikstra’s algorithm (for route planning)
- Simulated annealing
- Hill Climbing