AI Revise lesson

UNIT I: Artificial Intelligence

AI (Artificial intelligence) is the ability of a computer that is able to perform tasks like humans such as reasoning, learning, perception, problem-solving, and linguistic intelligence (ability to speak, write in any languages).

An agent is anything that can be viewed as perceiving its environment through the sensor and acting upon that environment through actuators.
An agent is something that perceives and acts in an environment.

Applications of AI

1. Problem Solving, Game Playing, Theorem Proving
2. Natural Language Processing, Speech recognition, Pattern recognition
3. Image Processing & Machine Translation
4. Finance industry (international stock trade)
5. Space exploration (NASA vehicle on Mars)
6. Data mining
7. Computer vision,
8. pattern recognition (optical character recognition), handwriting recognition
9. intelligent agents (smart home security systems),
10. Robotics Vehicles

Percept: it refers to the agent's perceptual inputs at any given instant.
Percept Sequence: is the complete history of everything the agent has ever perceived.

Agent Function: The agent function is an abstract mathematical description that maps any given percept sequence to an action.
The agent function maps from percept histories to actions:
f : P* → A
Agent Program: is a concrete implementation, running within some physical system.
The agent program runs on the physical architecture to produce f.

Agent Example: **Vacuum Cleaner(move left, move right, do nothing, suck up)** squares A and B, If the square is clean, The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck

up the dirt, or do nothing. If the current square is dirty, then suck; otherwise, move to the other square.

A rational agent is one that does the right thing.

A rational agent acts to maximize the expected value of the performance measure.

Rationality maximizes expected performance, while **perfection** maximizes actual performance.

Task environments are the **problems** to which rational agents are the **solution**.

Task Environment

To do a **performance measure** of an agent, we use the PEAS table, which stands for Performance Measure, Environment, Actuators, Sensors.

Performance Measure is the criteria that are utilized to measure the performance of an agent.

The performance measure evaluates the behavior of the agent in an environment.

Environment: is the environment surrounding the agent.

Sensor: is the device through which the percept input into the system.

Actuator: is a component of a machine that is responsible for moving and controlling a mechanism or system.

Example of Task environment

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

Exam of Cooking Agent:

Performance Measure: cooking like using the stove or other utensils properly, balance the heat, choose and mix the ingredients, recognize the place where we put the food, remember the order list and the receipt to create the food.

Environment: In the kitchen that sometimes might tough the hot thing or cold thing or wet thing…, may utensil

Actuators: Hand to make a response, speaker to talk back, machine to understand.

Sensor: microphone, a sensor to measure the heart or cool, camera to see the thing, taste and smell sensor.

Properties of Task Environment

Fully observable vs. partially observable:

- **Fully observable:** if an agent's sensors give all access to the complete state of the environment at each point in time.
- **Partially observable:** An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.

Unobservable: If the agent has no sensors.

Single-agent vs. Multi-agent:

- **Single-agent:** If only one agent is involved in an environment, and operating by itself then such an environment is called a single-agent environment.
- **Multi-agent:** If multiple agents are operating in an environment, then such an environment is called a multi-agent environment.

Multi-agent is divided into two: **Competitive multi-agent environment: Ex:** Playing Chess and **Cooperative multi-agent environment**: **Ex:** Taxi Driving.

Deterministic vs. Stochastic:

- **Deterministic**: If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic.
- **Stochastic**: is random in nature and cannot be determined completely by an agent, it is based on some probability measure.

Uncertain: if it is not fully observable or not deterministic.

Episodic vs. Sequential

- **Episodic:** It doesn't affect the future decision.
- **Sequential:** It affects the future decision**.**

Static vs. Dynamic

If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static.

semi-dynamic: If the environment doesn't change with time but the agent's performance, the score does.

Discrete vs. Continuous

- **Discrete:** is the duration amount of time.
- **Continuous:** is the duration of time continuously.

Known vs. Unknown

- **Known:** is known partially and known agent know which cause of action to take in this environment
- **Unknown:** This is completely known but the agent doesn't know which cause of action to take in the current environment.

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

Structure of Agent

agent = architecture + program

The job of AI is to design an agent program that implements the agent function of mapping from precepts to actions.
The difference between the agent program, which takes the current percept as input, and the agent function, which takes the entire percept history.

Agent Types

Simple reflex agents respond directly to percepts, whereas model-based reflex agents maintain an internal state to track aspects of the world that are not evident in the current percept. Goal-based agents act to achieve their goals, and utility-based agents try to maximize their own expected "happiness."

Simple reflex agents(fully observable): simplest agents and take decisions on the basis of the current percepts and ignore the rest of the percept history

Model-Based reflex Agent: agent should maintain some sort of internal state(knowledge of knowing the world) that depend on the percept history and reflects at least some of the unobserved aspect of the current state

Goal-based Agents: the knowledge of the current state environment is not always sufficient to decide for an agent what to do and Expand the capabilities of model-based agents by having the "goal" information. The agent will achieve the goal so that can achieve the goal.

Utility-based agent: similar to the goal-based agent but provides an extra component of utility measurement which makes them different by providing a measure of success at a given state. Utility-based agents act based not only on goals but also on the best way to achieve the goal.

Learning Agent

It is a different component of AI agents.

It is impossible to program a perfect intelligent machine and deploy it in one go.

Alan Turing proposed a method to build up a learning machine and then to teach them perfectly in an environment.

- **Components of Learning Agent:**

Learning agent can be divided into 4 conceptual components:

Learning Element: 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. The design of the learning element depends very much on the design of the performance element.

Performance Element: is responsible for selecting external actions. The The performance element is the entire agent: it takes in percepts and decides on actions.

Critic: decides how the agent is doing and determines how the performance element should be modified to do better in the future. The critic tells the learning element how well the agent is doing with respect to a fixed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent's success.

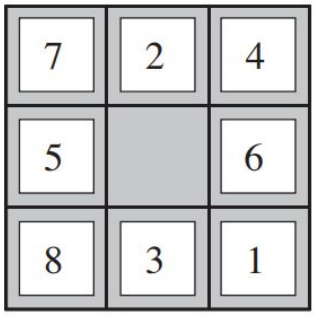
Problem Generator: It is responsible for suggesting actions that will lead to new and informative experiences.

All agents can improve their performance through learning.

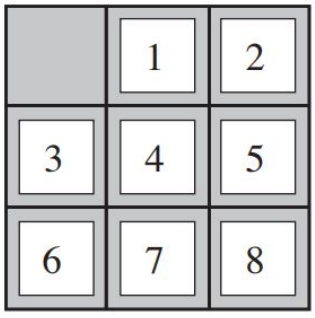
UNIT II: Solving Problem by Searching

Problem statement
Problem solving Agent
Goals help organize behavior by limiting the objectives that the agent is trying to achieve.
Goal formulation : It is the first step in problem solving and based on the current situation and the agent's performance measure.
A goal to be a set of world states—exactly those states in which the goal is satisfied.
The agent's task is to find out how to act, now and in the future, so that it reaches a goal state. it needs to decide what sorts of actions and states it should consider.
Problem formulation is the process of deciding what actions and states to consider, given a goal.
Search Algorithm Terminologies:
Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
Search Space: Search space represents a set of possible solutions, which a system may have.
Start State: It is a state from where an agent begins the search.
Goal test: It is a function which observes the current state and returns whether the goal state is achieved or not.
Search tree: A tree representation of search problems is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
Actions: It gives the description of all the available actions to the agent.
Transition model: A description of what each action does, can be represented as a transition model.
Path Cost: It is a function which assigns a numeric cost to each path.
Solution: It is an action sequence which leads from the start node to the goal node.
Optimal Solution: If a solution has the lowest cost among all solutions.

Example: Toy Problem



Start State

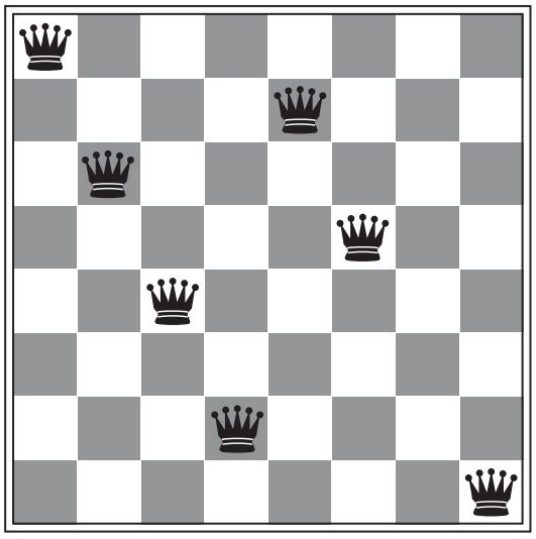


Goal State

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.
Actions: The simplest formulation defines the actions as movements of the blank space Left, Right, Up, or Down.
Transition model: Given a state and action, this returns the resulting state.
Goal test: This checks whether the state matches the goal configuration.
Path cost: Each step costs 1, so the path cost is the number of steps in the path.

Example: 8 queens



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.

Real World Problem : Airline Travel Problems

States: Each state obviously includes a location (e.g., an airport) and the current time.
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.

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
```

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
```

Properties of Search Algorithms:

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution is said to be an optimal solution.

Time Complexity: Time complexity is a measure of time for an algorithm to complete its task.

Space Complexity: It is the maximum storage space required at any point during the search, as the complexity of the problem.

ANOTHER MEANING
an algorithm's performance can be evaluated 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?

❖ Search Algorithms in Artificial Intelligence

Uninformed Search Algorithms: does not contain any domain knowledge such as closeness, the location of the goal.
It can be divided into five main types:

1. Breadth-first search
2. Uniform cost search
3. Depth-first search
4. Iterative deepening depth-first search
5. Bidirectional Search
6. Informed Search

Informed Search Algorithms: use domain knowledge.

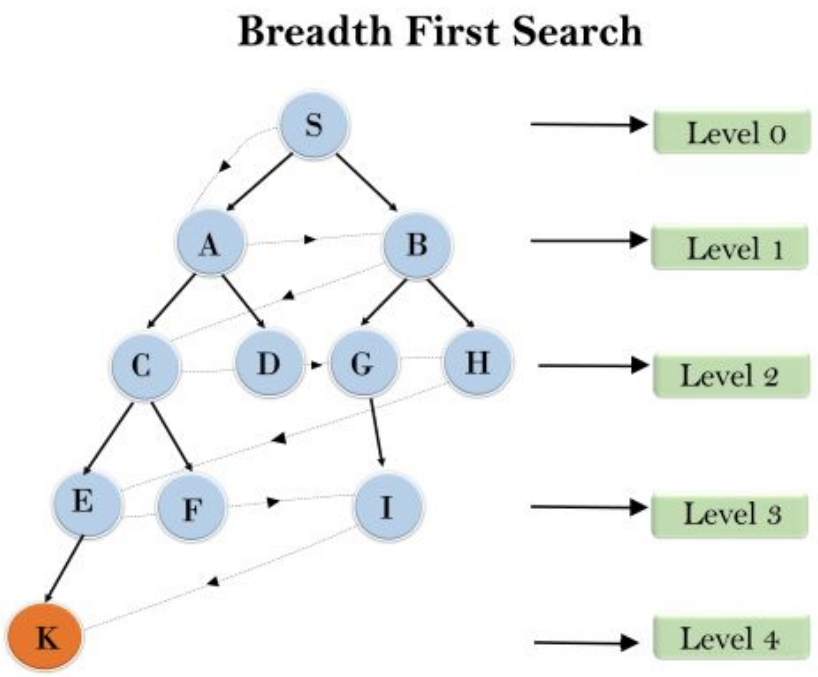
1. Best first search
2. A* search

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.
Link Uninformed Search: https://www.javatpoint.com/ai-uninformed-search-algorithms
Uninformed Search Algorithms:

Breadth-First Search

Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. DFS uses a **queue** data structure

- **it is complete**—if the shallowest goal node is at some finite depth d, breadth-first search will eventually find it after generating all shallower nodes (provided the branching factor b is finite).
- The shallowest goal node need not compulsorily be the optimal goal node. **BFS is optimal if the path cost is a non-decreasing function of d(depth) of the node.** The most common use of BFS when all actions have the same cost.



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.....+ b^d= O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

Depth-First Search

- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.

Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

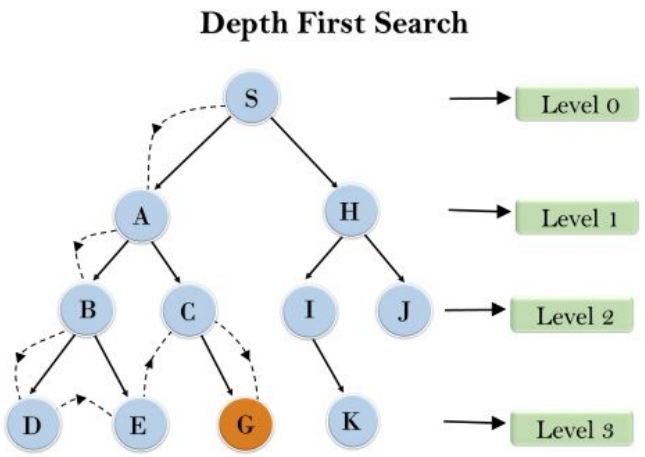
Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n)= 1+ n^2+ n^3 +.....+ n^m=O(n^m)$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

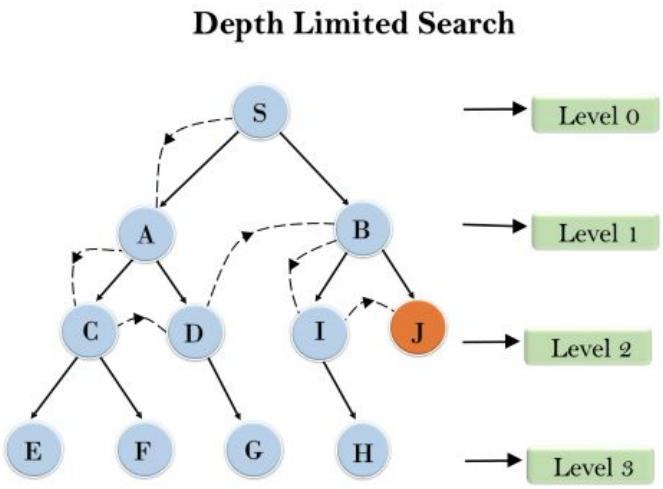


Depth-limit First Search

- A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that the problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

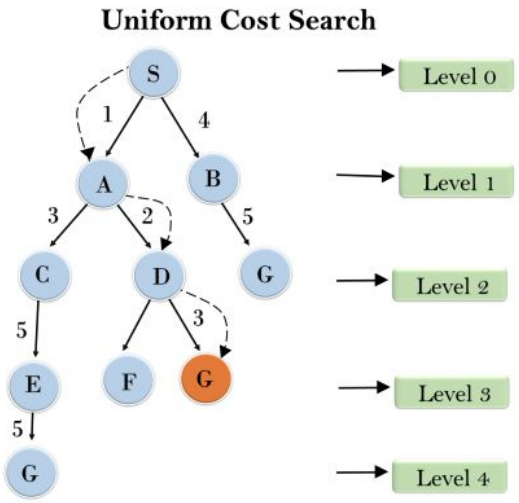
Time Complexity: Time complexity of DLS algorithm is $O(b^{\ell})$.

Space Complexity: Space complexity of DLS algorithm is $O(b \times \ell)$.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell > d$.

Uniform-cost Search Algorithm

Uniform-cost Search(UCS) Algorithm is optimal with any step-cost function. Instead of expanding the shallowest node, uniform-cost search expands the node n with the lowest path cost g(n). This is done by storing the frontier as a priority queue ordered by g.



Completeness:

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity:

Let C* **is Cost of the optimal solution**, and ε is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.

Hence, the worst-case time complexity of Uniform-cost search is$O(b^{1 + [C^*/\varepsilon]})/$.

Space Complexity:

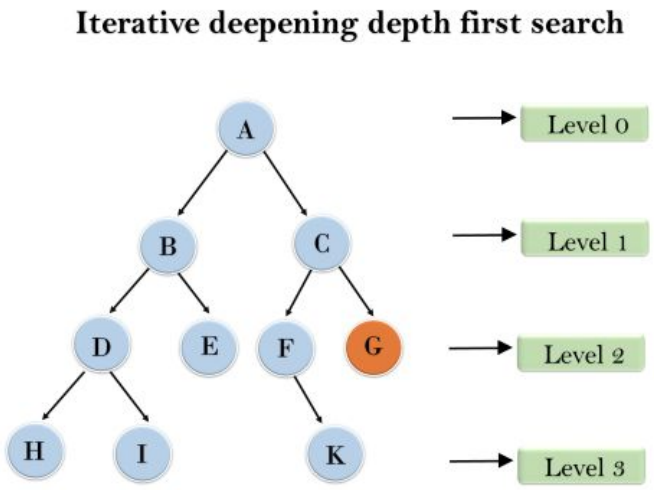
The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $O(b^{1 + [C^*/\varepsilon]})$.

Optimal:

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

Iterative deepening depth-first Search

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.



1'st Iteration-----> A

2'nd Iteration----> A, B, C

3'rd Iteration------>A, B, D, E, C, F, G

4'th Iteration------>A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

Completeness:

This algorithm is complete if the branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is O(bd).

Space Complexity:

The space complexity of IDDFS will be O(bd).

Optimal:

The IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

Bidirectional Search Algorithm

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from the goal vertex. The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Comparing uninformed search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Informed Search Strategies

Greed base-first Search

Greedy best-first search expands the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, f(n) = h(n).

we use the straight-line distance heuristic, which we will call .

A* Search

It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal: f(n) = g(n) + h(n) .

Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal,

we have f(n) = estimated cost of the cheapest solution through n .

It turns out that this strategy is more than just reasonable: provided that the heuristic function h(n) satisfies certain conditions, A∗ search is both complete and optimal. The algorithm is identical to UNIFORM-COST-SEARCH except that A∗ uses g + h instead of g.