Al-Mustaqbal University
College of Science

جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلـوم
قســـم علوم الذكاء الاصطناعي

المحاضرة الثانية

المادة: Heuristic Search
المرحلة: الثانية
اسم الاستاذ: م.م هادي صلاح

Study Year: 2025-2026

General Problem-Solving Approaches

There exist quite a large number of problem-solving techniques in AI that rely on search. The simplest among them is the generate-and-test method. The algorithm for the generate-and-test method can be formally stated in figure (1). It is clear from the algorithm that the algorithm continues the possibility of exploring a new state in each iteration of the repeat-until loop and exits only when the current state is equal to the goal. Most important part in the algorithm is to generate a new state. This is not an easy task.

```
Procedure Generate & Test
Begin
    Repeat
      Generate a new state and call it current-state;
    Until current-state = Goal;
End.
```

Figure (1). Generate and Test Algorithm

Note (State Generation Failure)

This note explains that if no new states can be generated, the search algorithm must terminate. This situation often occurs when no applicable operators remain.

Example: In a puzzle-solving problem, if all possible moves have already been explored and no new configuration can be produced, the algorithm stops.

If generation of new states is not feasible, the algorithm should be terminated. In simple algorithm, we, however, did not include this intentionally to keep it simplified. But how does one generate the states of a problem? To formalize this, we define a four tuple, called state space, denoted by

{nodes, arc, goal, current}

Clarification (State Space Components)

This clarification explains the four components of the state space in a formal way.

Example:

In a route-finding problem:

Nodes = cities

Arc = roads

Goal = destination city

Current = present city

Nodes represent the set of existing states in the search space;

an arc denotes an operator applied to an existing state to cause transition to another state;

Goal denotes the desired state to be identified in the nodes;

and current represents the state, now generated for matching with the goal.

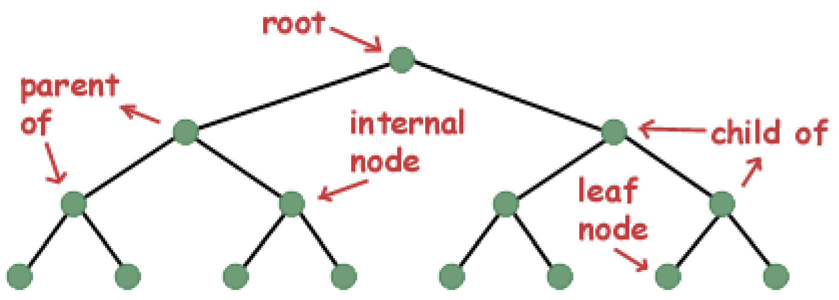
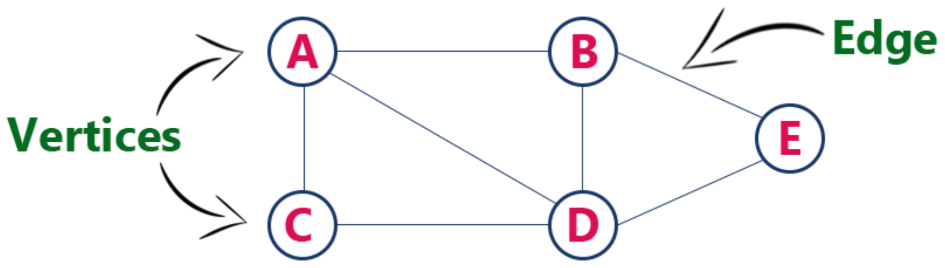
Note (Tree vs Graph Representation)

This note highlights the structural difference between tree-based and graph-based state spaces.
Example: A family tree is a tree structure, while a city map with multiple routes is a graph.
The state space for most of the search problems takes the form of a tree or a graph. Graph may contain more than one path between two distinct nodes, while for a tree it has maximum value of one.

The state space for most search problems takes the form of either a **tree** or a **graph**. A graph may contain more than one path between two distinct nodes, whereas in a tree there is at most one path between any two nodes.

Difference between Tree and Graph

Trees	Graphs
Only one path between any two nodes	May have multiple paths between nodes
Always connected	May or may not be connected
No loops or cycles (acyclic)	May contain cycles and self-loops
Has a root node	No root node
Traversal (pre-order, in-order, post-order)	Searching (BFS, DFS)
n vertices have n−1 edges	Edges independent of number of vertices
Hierarchical structure	Network structure
All trees are graphs	Not all graphs are trees
Lower complexity	Higher complexity

Tree:

Graph:


To build a system capable of solving a particular problem, four essential steps must be followed:

1. **Define the problem precisely.** This definition must include clear specifications of the initial situation(s), as well as a description of what final situations constitute acceptable solutions to the problem.
2. **Analyze the problem.** Certain important features of the problem can have a significant impact on the suitability of various techniques that may be used to solve it.
3. **Isolate and represent the task knowledge** that is necessary to solve the problem.
4. **Choose the best problem-solving technique(s)** and apply it (them) to the particular problem.

Measuring problem-solving performance is an essential aspect of any problem-solving approach. The output of a problem-solving algorithm is either a solution or a failure. (In some cases, an algorithm may become stuck in an infinite loop and never produce an output.) The performance of an algorithm is evaluated using four main criteria:

- **Completeness**: Is the algorithm guaranteed to find a solution when one exists?
- **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 required to perform the search?

Search Technique Process

Having formulated some problems, we now need to solve them. This is achieved by performing a search through the state space. The root of the search tree is a search node corresponding to the initial state. The first step in the process is to test whether this state is a goal state. Since it is not a goal state, we must consider other possible states.

This is accomplished by expanding the current state; that is, by applying the successor function to the current state, which generates a new set of states. At this point, we must choose which of these possibilities should be considered further. This process of choosing, testing, and expanding continues until either a solution is found or there are no more states left to be expanded. The decision regarding which state to expand is determined by the chosen search strategy.

It is important to distinguish between the **state space** and the **search tree**. For example, in the route-finding problem, there are only *N* states in the state space, one corresponding to each city. However, there may be an infinite number of nodes in the search tree.

There are many ways to represent nodes. In this discussion, we assume that a node is a data structure consisting of five components:

- **STATE**: the state in the state space to which the node corresponds؛
- **PARENT-NODE**: the node in the search tree that generated this node؛
- **ACTION**: the action that was applied to the parent node to generate this node؛
- **PATH-COST**: the cost, traditionally denoted by *g(n)*, of the path from the initial state to the node, as indicated by the parent pointers؛ and
- **DEPTH**: the number of steps along the path from the initial state.

As usual, we differentiate between two main families of search strategies: systematic search and local search. Systematic search visits each state that could be a solution, or skips only states that are shown to be dominated by others, so it is always able to find an optimal solution.

(Heuristic Search Algorithms)

In this section, we observe that many problems falling within the scope of artificial intelligence are too complex to be solved using direct techniques alone. Instead, such problems must be addressed through appropriate search methods that are supported by whatever direct techniques are available to guide the search process. These methods are collectively known as heuristic search algorithms.

Heuristic search algorithms can be described independently of any specific task or problem domain. However, when applied to particular problems, their effectiveness depends heavily on how well they exploit domain-specific knowledge. On their own, these methods are generally unable to overcome the combinatorial explosion to which search processes are highly vulnerable. For this reason, they are often referred to as weak methods.

Despite the recognition emerging from decades of artificial intelligence research that these weak methods have limited effectiveness in solving complex problems independently, they remain highly valuable. They continue to provide a general framework into which domain-specific knowledge can be incorporated, either manually or through automatic learning techniques.