# COMP 9414 WEEK 2 - Problem Solving and Search

Author: Junji Duan

Last Update: 2025/09/27

# 1 W2A: Problem Solving

## 1.1 Search ——Weak method

Search as a "weak method" of problem solving with wide applicability.

- Uninformed search methods (use no problem-specific information)
  - 无信息搜索（uninformed search）方法不使用额外的"问题特定知识"
  - 它们只知道：
    - 哪些状态是起点、目标；
    - 每个状态可以扩展到哪些后继状态
  - 特点：通用，但可能很慢，因为它们不会"聪明地"选择下一个方向
- Informed search methods (use heuristics to improve efficiency)
  - 有信息搜索（informed search）会用**启发式 (heuristics)** 来指导搜索，提高效率
  - 启发式就是"问题相关的知识"——一个函数/估计值，能帮助判断"哪个方向更接近目标"
  - 特点：更快、更高效，但依赖启发式设计质量

### 1.1.1 State Space Search Problems

- State space - set of all states reachable from initial state(s) by any action sequence
- Initial state(s) - element(s) of the state space
- Transitions
  - Operators - set of possible actions at agent's disposal; describe state reached after performing action in current state, or
  - Successor function $-s(x) =$ set of states reachable from state $x$ by performing a single action
- Goal state(s) - element(s) of the state space
- Path cost - cost of a sequence of transitions used to evaluate solutions (apply to optimization problems)

### 1.1.2 Complications(状态空间复杂性)

Single-state：知道自己确切状态，动作确定。

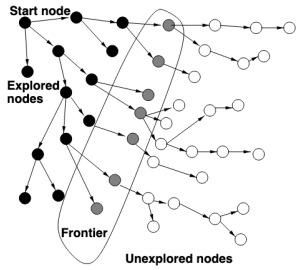
Multiple-state：只能确定可能状态集合。

Contingency：动作结果不确定，需要执行中感知。

Exploration：对环境完全未知，需试探学习。

## 1.2 Basic knowledge on search

### 1.2.1 General Search Space (not State Space)




Search strategy –way in which frontier expands

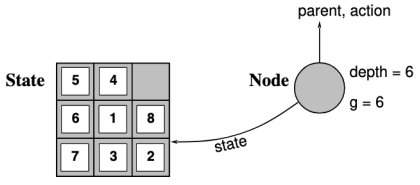
### 1.2.2 State Space vs Search Space

A **state** is part of the formulation of the search problem

A **node** is a data structure used in a search graph/tree, and includes:

- **parent, operator, depth, path cost g(x)**

**States** do not have parents, children, depth, or path cost!



Two different nodes can have the same state

### 1.2.3 Evaluating Search Algorithms

Completeness: strategy guaranteed to find a solution when one exists?

Time complexity: how long to find a solution?

Space complexity: memory required during search?

Optimality: when several solutions exist, does it find the "best" ?

Note: States are constructed during search, not computed in advance, so
efficiently computing successor states is critical!

### 1.2.4 Analysis of Algorithms –Big-O

$T(n)$ is $O(f(n))$ means that there is some $n_0$ and $k$ such that

- $T(n) \leq kf(n)$ for every problem of size $n \geq n_0$

Independent of implementation, compiler, fixed overheads, ...

$O()$ abstracts over constant factors

Examples

- $O(n)$ algorithm is better than an $O\left(n^2\right)$ algorithm (in the long run)
- $100n + 1000$ is better than $n^2 + 1$ for $n > 110$
- Polynomial $O\left(n^k\right)$ much better than exponential $O\left(2^n\right)$

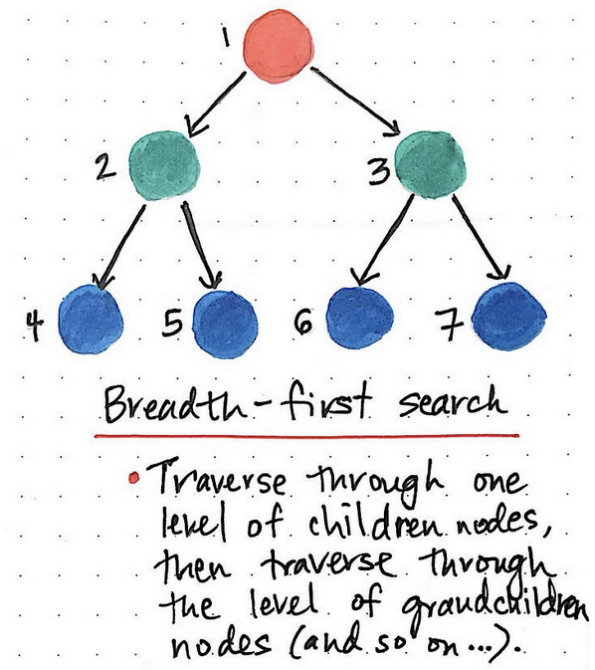
$O()$ notation is a compromise between precision and ease of analysis

## 1.3 Uninformed search methods

### 1.3.1 Breadth-First Search

Expands nodes level by level, exploring all shallow nodes before deeper ones.

- 从根结点开始逐层扩展，先探索所有浅层节点，再进入更深层。




Stop when node with goal state is generated

Include check that generated state not already explored or on frontier

- Needs a new data structure for set of explored states

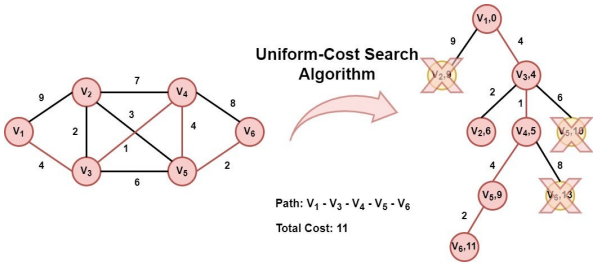
Analysis:

- Complete
- Optimal - provided path cost is nondecreasing function of the depth of the node
- Maximum number of nodes generated: $b + b^2 + b^3 + \ldots + b^d$ (where $b =$ forward branching factor; $d =$ path length to solution)
- Time and space requirements are the same $O\left(b^d\right)$

### 1.3.2 Uniform Cost Search

Always expands the node with the lowest path cost, ensuring the cheapest solution.

- 总是优先扩展当前路径代价最小的节点，保证找到最便宜的解。



Also known as Lowest-Cost-First search

Breadth-first search  uniform cost search where g(n) = depth(n)
(except breadth-first search stops when goal state generated)

Include check that generated state has not already been explored

Include test to ensure frontier contains only one node for any state –
for path with lowest cost

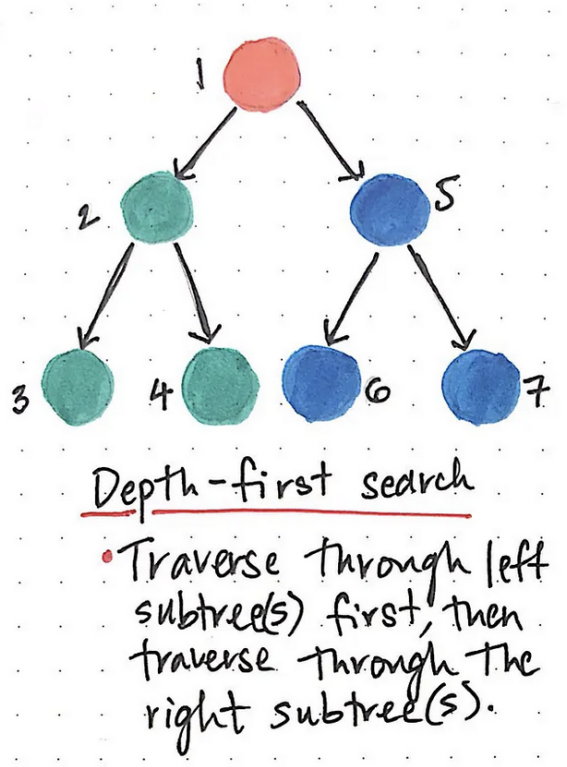
Analysis:

- Complete
- Optimal - provided path cost does not decrease along path (i.e. $g(\text{successor}(n)) \geq g(n)$ for all $n$)
- Reasonable assumption when path cost is cost of applying operators along the path
- Performs like breadth-first search when $g(n) = \text{depth}(n)$
- If there are paths with negative cost, need exhaustive search

### 1.3.3 Depth-First Search

Follows one branch down as far as possible, then backtracks to explore alternatives.

- 沿着一个分支尽可能深入下去，直到终点或死路，再回溯探索其他分支。




Stop when node with goal state is expanded

Include check that generated state has not already been explored
along a path –cycle checking

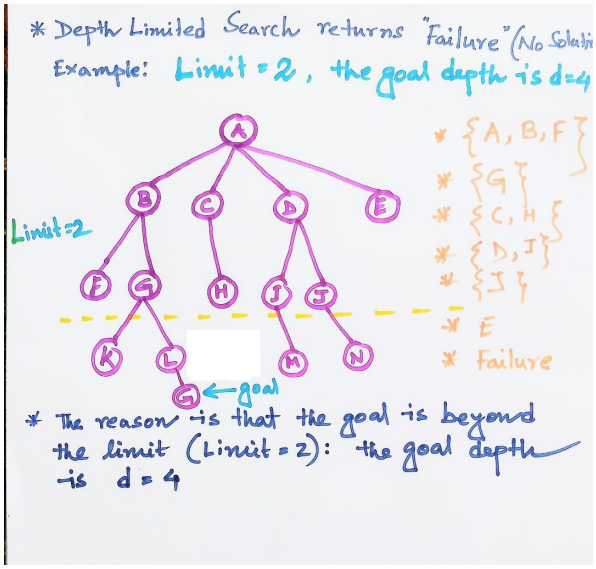
Analysis:

- Storage: $O(bm)$ nodes (where $m$ = maximum depth of search tree)
- Time: $O\left(b^m\right)$
- In cases where problem has many solutions, depth-first search may outperform breadth-first search because there is a good chance it will find a solution after exploring only a small part of the space
- However, depth-first search may get stuck following a deep or infinite path even when a solution exists at a relatively shallow level
- Therefore, depth-first search is not complete and not optimal
- Avoid depth-first search for problems with deep or infinite paths

### 1.3.4 Depth-Limited Search

A depth-first search with a maximum depth limit to prevent infinite descent.

- 在深度优先搜索的基础上加上最大深度限制，避免无限深入。




Analysis:

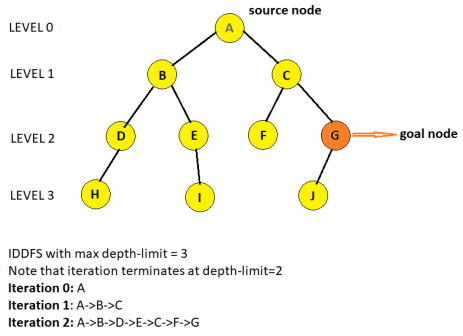
- Complete but not optimal (may not find shortest solution)

- However, if the depth limit chosen is too small a solution may not be found and depth-limited search is incomplete in this case
- Time and space complexity similar to depth-first search (but relative to depth limit rather than maximum depth)

### 1.3.5 Iterative Deepening Search

Repeats depth-limited searches with increasing depth, combining BFS completeness and DFS memory efficiency.

- 反复执行逐渐加深的深度限制搜索，结合 BFS 的完整性与 DFS 的低内存。




IDDFS with max depth-limit = 3
Note that iteration terminates at depth-limit=2
**Iteration 0:** A
**Iteration 1**: A->B->C
**Iteration 2:** A->B->D->E->C->F->G

The maximum path cost between any two nodes is known as the diameter of the state space

This would be a good candidate for a depth limit but it may be difficult to determine in advance (很难提前确定)

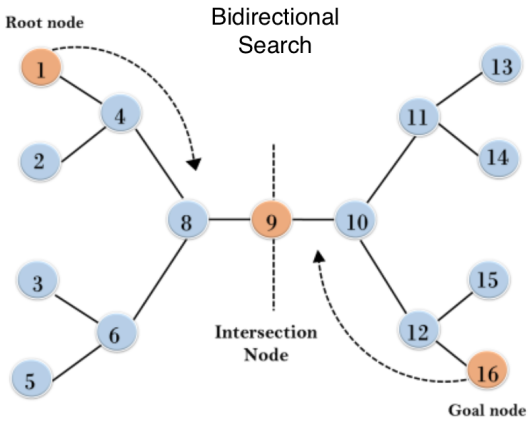
Analysis:

- Optimal; Complete; Space $-O(bd)$
- Some states are expanded multiple times: Isn't this wasteful?
  - Number of expansions to depth $d = 1 + b + b^2 + b^3 + \ldots + b^d$
  - Therefore, for iterative deepening, total expansions to depth $d$ $= (d+1)1 + (d)b + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + b^d$
  - The higher the branching factor, the lower the overhead (even for $b = 2$, search takes about twice as long)
  - Hence time complexity still $O\left(b^d\right)$
- Can double depth limit at each iteration, though no longer optimal
- In general, iterative deepening is the preferred search strategy for a large search space where depth of solution is not known

### 1.3.6 Bidirectional Search

Searches forward from the start and backward from the goal, meeting in the middle to reduce effort.

- 同时从起点和目标两端进行搜索，在中间相遇以减少搜索空间。




Analysis:

- If solution exists at depth $d$ then bidirectional search requires time $O\left(2b^{\frac{d}{2}}\right) = O\left(b^{\frac{d}{2}}\right)$ (assuming constant time checking of intersection)
- To check for intersection must have all states from one of the searches in memory, therefore space complexity is $O\left(b^{\frac{d}{2}}\right)$

### 1.3.7 Summary

| Criterion | Breadth First | Uniform Cost | Depth-First | Depth-Limited | Iterative Deepening | Bidirectional |
|---|---|---|---|---|---|---|
| Time | $b^d$ | $b^d$ | $b^m$ | $b^l$ | $b^d$ | $b^{\frac{d}{2}}$ |
| Space | $b^d$ | $b^d$ | $bm$ | $bl$ | $bd$ | $b^{\frac{d}{2}}$ |
| Optimal | Yes | Yes | No | No | Yes | Yes |
| Complete | Yes | Yes | No | Yes, if $l \geq d$ | Yes | Yes |

$b$ – branching factor
$d$ – depth of shallowest solution
$m$ – maximum depth of tree
$l$ – depth limit

# 2 W2B: Informed Search

Uninformed search methods are inefficient. With the aid of problem-specific knowledge, **informed methods of search are more efficient.**

- Informed search algorithms in AI are search methods that use extra knowledge, called heuristics, to prioritize which paths to explore. By estimating how close each possible step is to the goal, these algorithms can find solutions more quickly and efficiently than uninformed or "blind," search.

## 2.1 Heuristics

**Def: Heuristics are criteria(准则), methods or principles for deciding which among several alternative courses of action(行动方针) promises to be the most effective in order to achieve some goal.**

- In search, heuristic must be an underestimate of actual cost (实际代价的低估) to get from current node to any goal –an admissible heuristic.
- Denoted h(n); h(n) = 0 whenever n is a goal node.

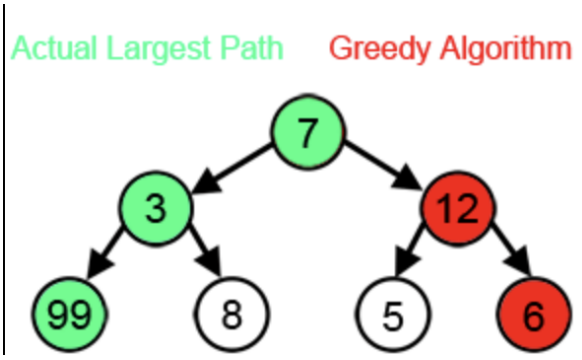
## 2.2 Informed Search Methods

### 2.2.1 Best-First Search and Greedy Search

Best-First Search is a general search strategy that expands the most promising node according to an evaluation function f(n), while Greedy Best-First Search is a special case that uses only the heuristic h(n).

Greedy Best-First Search always picks the next step that looks closest to the goal, using a guess (heuristic) about which choice is best. It tries to reach the goal as quickly as possible but isn't guaranteed to find the shortest path.

- Advantage: Efficient at quickly finding a solution by following the most promising paths.
- Limitation: Can get stuck in local optima; does not guarantee finding the shortest (optimal) path.



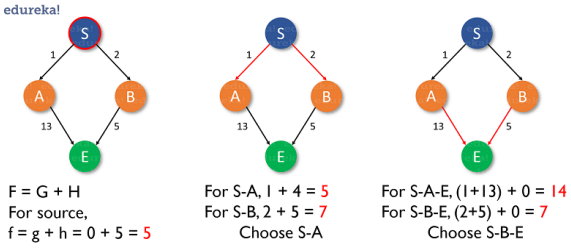

Analysis

- Similar to depth-first search; tends to follow single path to goal
- Not optimal, incomplete
- Time O($b^m$); Space O($b^m$)
- However, good heuristic can reduce time and space complexity significantly

### 2.2.2 $A^*$ Search

A* Search finds the shortest path by combining two things: how far we have already gone and a guess of how far is left to go. It checks both actual progress and potential, so it's very reliable if our guess (heuristic) is good. It uses a combined cost function f(n)=g(n)+h(n) actual path cost so far g(n) plus heuristic estimate h(n) to the goal. Finds both shortest and most cost-effective path with an admissible heuristic.

- Advantage: Complete and optimal (when using an admissible heuristic); widely applicable.
- Limitation: Can consume high memory for large graphs or spaces




Suppose we have a small graph with the vertices: S, A, B, E where S is the source and E is the destination.

Remember that the cost to enter the source and destination is always 0.

The heuristic values are: S -> 5，A -> 4，B -> 5，E -> 0

f = g + h where g is cost to travel and h is the heuristic value.

To reach Source: f(S) = 0 + 5 = 5

The paths from S to other vertices: f(S-A) = 1 + 4 = 5，f(S-B) = 2 + 5 = 7

So, we firstly will choose the path of S -> A as it is the least.

The paths from A and B to the Destination: f(S-A-E) = (1 + 13) + 0 = 14，f(S-B-E) = (2 + 5) + 0 = 7

After calculation, we have now found that B later has given us the least path. So, we change our least path to S-B-E and have reached our destination. That is how we use the formula to find out the most optimal path.

Analysis

- Optimal (and optimally efficient)
- Complete
- Number of nodes searched (and stored) still exponential in worst case（A* 需要维护 Open/Closed 表来记录所有候选节点, 内存占用也可能是指数级的, 最坏情况依然很糟糕）

With an **admissible** heuristic, A* guarantees finding an optimal solution (**Optimality**) and is the most efficient optimal algorithm (**Optimal Efficiency**); if the heuristic also satisfies monotonicity (Consistency), memory can be saved and the implementation is simplified. (A* 在 admissible 启发式下既能保证找到最优解（Optimality），又是扩展节点最少的最优算法（Optimal Efficiency）<— $A^*$Search 的两个重要性质；如果启发式还满足单调性（Monotonicity），就能节省内存并简化实现。)

- $A^*$ is optimally efficient for a given heuristic: there is no other optimal algorithm that expands fewer nodes in finding a solution
- Monotonic heuristic –along any path, the f -cost never decreases

#### 2.2.3 Iterative Deepening $A^*$ Search

IDA* blends A*'s optimality and memory-efficient depth-first search. Performs iterative deepening using increasing cost thresholds computed from $f(n)$.

- 工作方式：
  - 它不像 A* 用一个 OPEN 表，而是做迭代加深。
  - 每次搜索设置一个阈值（threshold），这个阈值来自 f(n) = g(n)+h(n)。
  - 初始阈值 = 起点 f 值。
  - 如果搜索过程中 f(n) 超过阈值，就不继续展开。
  - 如果没找到解，就把阈值增加到 “刚刚超过的最小 f 值”，再重新搜索。
  - 直到找到目标结点。
- Advantage: Uses far less memory than A*, suitable for huge search spaces (like puzzle solving).
- Limitation: Repeats a lot of work; can be slower than $A^*$ due to repeated traversal.

## 参考文献


[1] UNSW COMP9414 Lecture slides –Prof. W.Wobcke

[2] Other online resources (eg.graph)