Chapter 4
Search Methodologies
Problem Solving

- Missionaries and Cannibals
- Hanoi Towers
- Maze
- Puzzles (8-Puzzle, 15-Puzzle)
- Brix (消除方塊)
- Freecell (新接龍)
- Magic Cube (Rubik’s Cube)
- Web Spidering
- Xmas Gift, etc.

Initial State (Start)
Actions
Goal(s) & Dead States?

最古老談論 State Search
是春秋時代的（____）
Example (Missionaries and Cannibals)

- **Start**
  
  \[(M, C, Canoe) = (0,0,0)\]

- **operators**
  
  1. Move one cannibal across the river.
  2. Move two cannibals across the river.
  3. Move one missionary across the river.
  4. Move two missionaries across the river.
  5. Move one missionary and one cannibal.
Example (Hanoi Tower)
Example (8-Puzzle)
Example (Maze)
Example (Maze, dead states)

Infinite Cycle

Pitfall
Example (Brix)

- ???
- Well
- Well
- Dead
Example (Freecell)
Chapter 4 Contents

- Brute force search
- Depth-first search
- Breadth-first search
- Properties of search methods
- Implementations
- Depth first iterative deepening
- Heuristics
- Hill climbing
- Best first search
- Beam search
- Optimal paths
- A* algorithms
- Uniform cost search
- Greedy search
Problem Solving

Data-Driven (forward-chaining)

Brute-Force Search (exhaustive Search)

Blind Search (Generate & Test)

Goal-Driven (back-chaining)

Beam Search

Heuristic Search (Informed Method)

Hill Climbing

Best-first S.

A* family

Greedy Search

A* Alg.

Uniform Cost Search (Branch and Bound, Dijkstra)

Not always complete

Evaluation function

British Museum Procedure (identifying the optimal path)
• break
Brute Force Search

- Search methods that examine every node in the search tree – also called *exhaustive*.
- Generate and test is the simplest brute force search method:
  - Generate possible solutions to the problem.
  - Test each one in turn to see if it is a valid solution.
  - Stop when a valid solution is found.
- The method used to generate possible solutions must be carefully chosen.
Depth-First Search

- An exhaustive search method.
- Follows each path to its deepest node, before backtracking to try the next path.
Breadth-First Search

- An exhaustive search method.
- Follows each path to a given depth before moving on to the next depth.
Comparison of Depth-First and Breadth-First Search

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Depth first</th>
<th>Breadth first</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some paths are extremely long, or even infinite</td>
<td>Performs badly</td>
<td>Performs well</td>
</tr>
<tr>
<td>All paths are of similar length</td>
<td>Performs well</td>
<td>Performs well</td>
</tr>
<tr>
<td>All paths are of similar length, and all paths lead to a goal state</td>
<td>Performs well</td>
<td>Wasteful of time and memory</td>
</tr>
<tr>
<td>High branching factor</td>
<td>Performance depends on other factors</td>
<td>Performs poorly</td>
</tr>
</tbody>
</table>

- It is important to choose the correct search method for a given problem.
- In some situations, for example, depth first search will never find a solution, even though one exists.
Properties of Search Methods

- **Complexity**
  - How much time and memory the method uses.

- **Completeness**
  - A complete method is one that is guaranteed to find a goal if one exists.

- **Optimality & Admissibility**
  - A method is optimal (or admissible) if it is guaranteed to find the best path to the goal.

- **Irrevocability**
  - A method is irrevocable if, like hill climbing, it does not ever back-track.

---

**Optimal:**
1. Alg./Sol.
2. Steps/ Costs

討論最佳解而非最佳演算法(AI可視為合理的次佳演算法)

**Recoverable:**
不必每個state都如實記錄可以逆推還原，例如 puzzle而反例是 brix 必須如實記錄
Implementations

- Depth-first search can be implemented using a queue to store states
  - But a stack makes more sense
  - And enables us to create a recursive depth-first search function
- Breadth-first search implementations are almost identical to depth-first
  - Except that they place states on the back of the queue instead of on the front.
Depth-First Iterative Deepening

- An exhaustive search method based on both depth-first and breadth-first search.
- Carries out depth-first search to depth of 1, then to depth of 2, 3, and so on until a goal node is found.
- Efficient in memory use, and can cope with infinitely long branches.
- Not as inefficient in time as it might appear, particularly for very large trees, in which it only needs to examine the largest row (the last one) once.

DFID, IDS (Iterative Deepening Search):
- Case 1. Resource有限 (memory/旅遊時間)
- Case 2. Game Tree, Min-Max變化大，限時
- break
Heuristics

- Heuristic: a rule or other piece of information that is used to make methods such as search more efficient or effective.
- In search, often use a heuristic evaluation function, f(n):
  - f(n) tells you the approximate distance of a node, n, from a goal node.
- f(n) may not be 100% accurate, but it should give better results than pure guesswork.
Heuristics – how informed?

- The more informed a heuristic is, the better it will perform.
- Heuristic \( h \) is more informed than \( j \), if:
  - \( j(n) \leq h(n) \leq h^*(n) \) for all nodes \( n \).
- A search method using \( h \) will search more efficiently than one using \( j \).
- A heuristic should reduce the number of nodes that need to be examined.

\[
\text{8-puzzles: } h1(), h2(), h3();
\text{Maze: straight-line distance}
\]

- 課本嚴重錯誤!!
Hill Climbing

- An informed, irrevocable search method.
- Easiest to understand when considered as a method for finding the highest point in a three dimensional search space:
- Check the height one foot away from your current location in each direction; North, South, East and West.
- As soon as you find a position whose height is higher than your current position, move to that location, and restart the algorithm.

DFS $\rightarrow$ Hill Climbing $\rightarrow$ Best First $\rightarrow$ A*

展開過程中遇到更好的點，就直接走過去，走一步算一步；樹形沒記錄，向前無法回頭，不是實用的解題法。另有一說，Hill Climbing是展開的點先排好，再Stack到OPEN list去，便可回溯。
Foothills

Cause difficulties for hill-climbing methods.

A foothill is a local maximum.
Plateaus

- Cause difficulties for hill-climbing methods.
- Flat areas that make it hard to find where to go next.
Ridges

- Cause difficulties for hill-climbing methods
- B is higher than A.

- At C, the hill-climber can’t find a higher point North, South, East or West, so it stops.
Best-First Search

- Works rather like hill-climbing:
- Picks the most likely path (based on heuristic value) from the partially expanded tree at each stage.
- Tends to find a shorter path than depth-first or breadth-first search, but does not guarantee to find the best path.
Beam Search

- Breadth-first method.
- Only expands the best few paths at each level.
- Thus has the memory advantages of depth-first search.
- **Not exhaustive**, and so may not find the best solution.
- May not find a solution at all.
• break
Outline

Problem Solving

Data-Driven (forward-chaining)

Brute-Force Search (exhaustive Search)

Blind Search (Generate & Test)

Goal-Driven (back-chaining)

Beam Search

Heuristic Search (Informed Method)

Hill Climbing

Best-first S.

A* family

A* Alg.

Uniform Cost Search (Branch and Bound, Dijkstra)

Greedy Search

British Museum Procedure (identifying the optimal path)

Evaluation function

Not always complete

e.g.

Maze, Proof

DFS

BFS

Iterative Deepening Search

A* Alg.
Optimal Paths

- An optimal path through a tree is one that is the shortest possible path from root to goal.
- In other words, an optimal path has the lowest cost (not necessarily at the shallowest depth, if edges have costs associated with them).
- The British Museum procedure finds an optimal path by examining every possible path, and selecting the one with the least cost.
- There are more efficient ways.

**BM**: 所有State都要能回溯其path/actions, 找出所有解再選最佳解
A* Algorithms

- Uses the cost function:
  \[ f(node) = g(node) + h(node). \]
- \( g(node) \) is the cost of the path so far leading to the node.
- \( h(node) \) is an *underestimate* of how far node is from a goal state.
- \( f \) is a path-based evaluation function.
- An A* algorithm expands paths from nodes that have the lowest \( f \) value.
Add START to OPEN list

while OPEN not empty

get node n from OPEN that has the lowest f(n)
if n is GOAL then return path
move n to CLOSED
for each n' = CanMove(n, direction)
    g(n') = g(n) + cost(n,n')
    calculate f(n')=g(n')+h(n')
    if n' in OPEN list and new n' is not better, continue
    if n' in CLOSED list and new n' is not better, continue
    replacement or insertion
        remove any n' from OPEN and CLOSED
        add n as n’s parent
        add n’ to OPEN ...從OPEN開始
end for
end while
if we get here, then there is No Solution

其實goal, i.e., Z,已放在OPEN而且經評估h=0, 唯尚未確定其為lowest之前都不算找到最佳解

For any X in the opt. path, if A…X…Z is Opt. Solution 則 ZY 將被永久放在 OPEN List
f(X)<f*(X)<f*(ZY)=f(ZY)
A* Alg (admissible heuristic)

- A* algorithms are optimal:
  - They are guaranteed to find the shortest path to a goal node.
- Provided $h$(node) is always an underestimate. ($h$ is an admissible heuristic).
- A* methods are also optimally efficient – they expand the fewest possible paths to find the right one.
- If $h$ is not admissible, the method is called A, rather than A*. 
A* Alg (example)

Note: 不必用 DFS 或 BFS 把每條路逕都找出來比較

還有「取代既有 CLOSE」未舉例

<table>
<thead>
<tr>
<th>Step</th>
<th>OPEN</th>
<th>CLOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A/0+8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>B/4+5, E/4+6, C/2+9</td>
<td>A/8</td>
</tr>
<tr>
<td>2</td>
<td>D/5+3, G/7+2, E/4+6, C/2+9, (E/9+6)</td>
<td>A/8, B/9 (E 自消不進OPEN)</td>
</tr>
<tr>
<td>3</td>
<td>G/6+2, (G/7+2), E/4+6, C/2+9</td>
<td>A/8, B/9, D/8 (G 取代既有OPEN)</td>
</tr>
<tr>
<td>4</td>
<td>H/7+3, E/4+6, C/9+2, M/13+0</td>
<td>A/8, B/9, D/8, G/8 (M 非最佳解)</td>
</tr>
<tr>
<td>5</td>
<td>E/4+6, C/9+2, M/12+0, L/11+1</td>
<td>A/8, B/9, D/8, G/8, H/10</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Uniform Cost Search

- Also known as branch and bound. or Dijkstra
- Like A*, but uses:
  - $f(\text{node}) = g(\text{node}); \ h(\text{node})=0$
  - $g(\text{node})$ is the cost of the path leading up to node.
- Once a goal node is found, the method needs to continue to run in case a preferable solution is found.

Recall: 此即以前看過的 Dijkstra Algorithm shortest path searching \(\rightarrow\) 從未要求所有連結自始展開
Greedy Search

- Like A*, but uses:
  - \( f(\text{node}) = h(\text{node}); \ g(\ ) = 0 \)
- Hence, always expands the node that appears to be closest to a goal, regardless of what has gone before.
- Not optimal, and not guaranteed to find a solution at all.
- Can easily be fooled into taking poor paths.
Outline

Problem Solving

Data-Driven (forward-chaining)

- Brute-Force Search (exhaustive Search)
  - DFS
  - BFS
  - Iterative Deepening Search

Goal-Driven (back-chaining)

- Beam Search

Blind Search (Generate & Test)

- Heuristic Search (Informed Method)
  - Hill Climbing
  - Best-first S.
  - A* family

A* Alg.

Uniform Cost Search
(Branch and Bound, Dijkstra)

Greedy Search

British Museum Procedure
(identification of the optimal path)

Not always complete

Evaluation function
TSP using A*

h(B1) = min(B-CDE) + min(CDE-A) + min(C-D,E-D,E-C) = 1+5+(2+6)=14

h(C2) = min(C-DE) + min(DE-A) + min(D-E) = 6+5+(2)=13