Problem solving by searching

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

A search problem

Many interesting problems in science and engineering are solved using search

A search problem is defined by:

• **A search space:**
  – The set of objects among which we search for the solution
    **Examples:** routes between cities, or n-queens configuration

• **A goal condition**
  – Characteristics of the object we want to find in the search space?
    – **Examples:**
      • Path between cities A and B
      • Non-attacking n-queen configuration
Graph Search Problems

Search problems can be often represented as graph search problems:

- **Initial state**
  - State (configuration) we start to search from (e.g. start city, initial game position)

- **Operators**:  
  - Transform one state to another (e.g. valid connections between cities, valid moves in Puzzle 8)

- **Goal condition**:  
  - Defines the target state (destination, winning position)

*Search space* is now defined indirectly through:

The initial state + Operators

---

Search

- **Search**: The process of exploration of the search space
- **Design goal**: We want the search to be as efficient as possible
- **The efficiency of the search depends on**:  
  - The search space and its size  
  - Method used to explore (traverse) the search space  
  - Condition to test the satisfaction of the search objective
Search

- **Search**: The process of exploration of the search space
- **Design goal**: We want the search to be as efficient as possible
- **The efficiency of the search depends on**:
  - The search space and its size
  - **Method used to explore (traverse) the search space**
  - Condition to test the satisfaction of the search objective
This lecture

- **Focus on:**
  - Methods used to explore (traverse) the search space

Search process

**Exploration of the state space** through successive application of operators from the initial state

- **Search tree = structure representing the exploration trace**
- Built on-line during the search process
- Branches correspond to explored paths, and leaf nodes to the exploration fringe
A search tree = (search) exploration trace
- different from the graph representation of the problem
- states can repeat in the search tree

A branch in the search tree = path in the graph
A branch in the search tree = path in the graph

General search algorithm

```
General-search (problem, strategy)
initialize the search tree with the initial state of problem
loop
  if there are no candidate states to explore return failure
  choose a leaf node of the tree to expand next according to strategy
  if the node satisfies the goal condition return the solution
  expand the node and add all of its successors to the tree
end loop
```
General search algorithm

General-search \((proble, strategy)\)
initialize the search tree with the initial state of \(problem\)
loop
  if there are no candidate states to explore next return failure
  choose a leaf node of the tree to expand next according to \(strategy\)
  if the node satisfies the goal condition return the solution
  expand the node and add all of its successors to the tree
end loop
General search algorithm

**General-search** *(problem, strategy)*
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore next return failure
  choose a leaf node of the tree to expand next according to *strategy*
    if the node satisfies the goal condition return the solution
  expand the node and add all of its successors to the tree
end loop

---

**Expanded nodes**
- Arad
- Zerind
- Sibiu
- Timisoara
- Oradea

**Generated (active, open) nodes**
- Arad
- Oradea

---

General search algorithm

**General-search** *(problem, strategy)*
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore next return failure
  choose a leaf node of the tree to expand next according to *strategy*
    if the node satisfies the goal condition return the solution
  expand the node and add all of its successors to the tree
end loop

---

**Expanded nodes**
- Arad
- Zerind
- Sibiu
- Timisoara
- Oradea

**Generated (active, open) nodes**
- Arad
- Oradea
- Arad
- Oradea
- Fagaras
- Rimnicu Vilcea
General search algorithm

General-search \((\text{problem, strategy})\)
initialize the search tree with the initial state of \text{problem}
loop
  if there are no candidate states to explore next return failure
  choose a leaf node of the tree to expand next according to \text{strategy}
  if the node satisfies the goal condition return the solution
  expand the node and add all of its successors to the tree
end loop

• Search methods differ in how they explore the space, that is how they choose the node to expand next !!!!
Implementation of search

• Search methods can be implemented using the queue structure

**General search** \((\text{problem, Queuing-fn})\)

\[
\begin{align*}
\text{nodes} & \leftarrow \text{Make-queue}(\text{Make-node}(\text{Initial-state}(\text{problem}))) \\
\text{loop} & \\
\quad & \text{if nodes is empty then return failure} \\
\quad & \quad \text{node} \leftarrow \text{Remove-node}(\text{nodes}) \\
\quad & \quad \text{if Goal-test}(\text{problem}) \text{ applied to State}(\text{node}) \text{ is satisfied then return node} \\
\quad & \quad \text{nodes} \leftarrow \text{Queuing-fn}(\text{nodes, Expand}(\text{node, Operators}(\text{node}))) \\
\end{align*}
\]

• Candidates are added to \text{nodes} representing the queue structure

---

Implementation of search

• A search tree node is a data-structure that is a part of the search tree

- **State**

- **ST Node**

- **Other attributes:**
  - state value (cost)
  - depth
  - path cost

• Expand function – applies Operators to the state represented by the search tree node. Together with Queuing-fn it fills the attributes.
Uninformed search methods

• Search techniques that rely only on the information available in the problem definition
  – Breadth first search
  – Depth first search
  – Iterative deepening
  – Bi-directional search

For the minimum cost path problem:
  – Uniform cost search

Search methods

Properties of search methods:

• Completeness.
  – Does the method find the solution if it exists?

• Optimality.
  – Is the solution returned by the algorithm optimal? Does it give a minimum length path?

• Space and time complexity.
  – How much time it takes to find the solution?
  – How much memory is needed to do this?
Parameters to measure complexities.

- **Space and time complexity.**
  - *Complexity* is measured in terms of the following tree parameters:
    - $b$ – maximum branching factor
    - $d$ – depth of the optimal solution
    - $m$ – maximum depth of the state space

**Branching factor**

Breadth first search (BFS)

- **The shallowest node is expanded first**
Breadth-first search

- Expand the shallowest node first
- Implementation: put successors to the end of the queue (FIFO)
Breadth-first search
Breadth-first search

Properties of breadth-first search

• Completeness: ?

• Optimality: ?

• Time complexity: ?
• Memory (space) complexity: ?

– For complexity use:
  • $b$ – maximum branching factor
  • $d$ – depth of the optimal solution
  • $m$ – maximum depth of the search tree
Properties of breadth-first search

- **Completeness**: Yes. The solution is reached if it exists.

- **Optimality**: Yes, for the shortest path.

- **Time complexity**: ?

- **Memory (space) complexity**: ?

BFS – time complexity

<table>
<thead>
<tr>
<th>depth</th>
<th>number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$2^1 = 2$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2 = 4$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3 = 8$</td>
</tr>
</tbody>
</table>

Expanded nodes: $O(b^d)$

Total nodes: $O(b^{d+1})$
Properties of breadth-first search

- **Completeness:** Yes. The solution is reached if it exists.

- **Optimality:** Yes, for the shortest path.

- **Time complexity:**
  \[ 1 + b + b^2 + \ldots + b^d = O(b^d) \]
  exponential in the depth of the solution \( d \)

- **Memory (space) complexity:** ?

**BFS – memory complexity**

- Count nodes kept in the tree structure or in the queue

<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>(2)</td>
</tr>
<tr>
<td>2</td>
<td>(2^2 = 4)</td>
</tr>
<tr>
<td>3</td>
<td>(2^3 = 8)</td>
</tr>
<tr>
<td>(d)</td>
<td>(2^d )</td>
</tr>
<tr>
<td>(d+1)</td>
<td>(2^{d+1} )</td>
</tr>
</tbody>
</table>

Expanded nodes: \(O(b^d)\)

Total nodes: \(O(b^{d+1})\)
Properties of breadth-first search

- Completeness: Yes. The solution is reached if it exists.
- Optimality: Yes, for the shortest path.
- Time complexity:

\[ 1 + b + b^2 + \ldots + b^d = O(b^d) \]

exponential in the depth of the solution \( d \)
- Memory (space) complexity:

\[ O(b^d) \]

nodes are kept in the memory

Depth-first search (DFS)

- The deepest node is expanded first
- Backtrack when the path cannot be further expanded
Depth-first search

- The deepest node is expanded first
- Implementation: put successors to the beginning of the queue
Depth-first search

Note: Arad – Zerind – Arad cycle
Properties of depth-first search

- **Completeness**: No. when no limit Infinite loops can occur. **May be** when the max depth limit is set - depends on how it is set
- **Optimality**: No. Solution found first may not be the shortest possible.
- **Time complexity**: ?
- **Memory (space) complexity**: ?

### DFS – time complexity

<table>
<thead>
<tr>
<th>depth</th>
<th>number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$2^1=2$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2=4$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3=8$</td>
</tr>
</tbody>
</table>

**Complexity**: $O(b^m)$
Properties of depth-first search

- **Completeness:** No. when no limit Infinite loops can occur. **May be** when the max depth limit is set - depends on how it is set
- **Optimality:** No. Solution found first may not be the shortest possible.
- **Time complexity:** \( O(b^m) \)
  
  exponential in the maximum depth of the search tree \( m \)
- **Memory (space) complexity:** ?

---

**DFS – memory complexity**

<table>
<thead>
<tr>
<th>depth</th>
<th>number of nodes kept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### DFS – memory complexity

<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of Nodes Kept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$2 = b$</td>
</tr>
</tbody>
</table>

**Diagram:**

- **DFS:** Depth-first search algorithm for artificial intelligence.
- **Memory Complexity:** The number of nodes kept in memory as a function of depth.

- **Diagram Details:**
  - Node b is highlighted to indicate the focus on memory usage at different depths.
  - Depth 0 has no nodes kept.
  - Depth 1 keeps 2 nodes, which is $b - 1$.
  - Depth 2 keeps 2 nodes, indicating the memory complexity.

---

### DFS – memory complexity

<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of Nodes Kept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$1 = (b-1)$</td>
</tr>
<tr>
<td>2</td>
<td>$2 = b$</td>
</tr>
</tbody>
</table>

**Diagram:**

- **DFS:** Depth-first search algorithm for artificial intelligence.
- **Memory Complexity:** The number of nodes kept in memory as a function of depth.

- **Diagram Details:**
  - Node b is highlighted to indicate the focus on memory usage at different depths.
  - Depth 0 has no nodes kept.
  - Depth 1 keeps 1 node, which is $b - 1$.
  - Depth 2 keeps 2 nodes, indicating the memory complexity.
DFS – memory complexity

<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of Nodes Kept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>m</td>
<td>2=b</td>
</tr>
</tbody>
</table>

Complexity: $O(bm)$
Properties of depth-first search

- **Completeness**: No. when no limit Infinite loops can occur.
  May be when the max depth limit is set
  - depends on how it is set
- **Optimality**: No. Solution found first may not be the shortest possible.

- **Time complexity**: 
  \[ O(b^m) \]
  exponential in the maximum depth of the search tree \( m \)

- **Memory (space) complexity**: 
  \[ O(bm) \]
  linear in the maximum depth of the search tree \( m \)