Uninformed search methods

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

Announcements

Homework assignment 1 is out
• Due on Tuesday, September 12, 2017 before the lecture
• Report and programming part:
  – Programming part involves Puzzle 8 problem.

Course web page:
http://www.cs.pitt.edu/~milos/courses/cs2710/

Homework submission:
• Electronic via CourseWeb
• Separate report and programming part
Search

- **Search (process)**
  - The process of exploration of the search space

- **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
    (what it takes to determine I found the desired goal object)

---

Search

- **Search (process)**
  - The process of exploration of the search space

- **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
    (what it takes to determine I found the desired goal object)
Search process

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

- **Search tree** = structure representing the exploration trace
  - Is built on-line during the search process
  - Branches correspond to explored paths, and leaf nodes to the exploration fringe

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

The number of applicable operators

---

**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

queue

Sibiu
Timisoara
Arad
Oradea

Breadth-first search

queue

Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea

Breadth-first search

queue

Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
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>
<tr>
<td>$d$</td>
<td>$2^d$ ($b^d$)</td>
</tr>
<tr>
<td>$d+1$</td>
<td>$2^{d+1}$ ($b^{d+1}$)</td>
</tr>
</tbody>
</table>

Total nodes: $O(b^d)$

Expanded nodes: $O(b^d)$
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(^1)=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 ) (( b^d ))</td>
</tr>
<tr>
<td>( d+1 )</td>
<td>( 2^{d+1} ) (( b^{d+1} ))</td>
</tr>
</tbody>
</table>

Total nodes: ?
**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

1. Start at Arad
2. Visit Zerind, Sibiu, Timisoara in that order
3. Queue: Zerind, Sibiu, Timisoara
4. Repeat until the queue is empty
Properties of depth-first search

- Completeness: Does it always find the solution if it exists?

- Optimality: ?

- Time complexity: ?

- Memory (space) complexity: ?
Properties of depth-first search

- **Completeness**: No. Infinite loops can occur.

- **Optimality**: does it find the minimum length path?

- **Time complexity**: ?

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

---

Properties of depth-first search

- **Completeness**: No. Infinite loops can occur.
  - **Solution 1**: set the maximum depth limit $m$
  - **Solution 2**: prevent occurrence of cycles

- **Optimality**: does it find the minimum length path?

- **Time complexity**: ?

- **Memory (space) complexity**: ?
Properties of depth-first search

- **Completeness:** No. If we permit infinite loops.
  - Solution 1: set the maximum depth limit $m$
  - Solution 2: prevent occurrence of cycles

- **Optimality:** does it find the minimum length path?

- **Time complexity:** ?

- **Memory (space) complexity:** ?
Properties of depth-first search

- **Completeness**: No. If we permit infinite loops.
  - **Solution 1**: set the maximum depth limit $m$
  - **Solution 2**: prevent occurrence of cycles

- **Optimality**: No. Solution found first may not be the shortest possible.

- **Time complexity**: assume a finite maximum tree depth $m$

- **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>
<tr>
<td>$d$</td>
<td>$2^d$</td>
</tr>
<tr>
<td>$m$</td>
<td>$2^m - 2^{m-d}$</td>
</tr>
</tbody>
</table>

Complexity:
Properties of depth-first search

- **Completeness**: No. If we permit infinite loops.
  - **Solution 1**: set the maximum depth limit \( m \)
  - **Solution 2**: prevent occurrence of cycles
- **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>
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>

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:
DFS – memory complexity

Count nodes kept in the tree structure or 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 = b</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>2</td>
</tr>
</tbody>
</table>

Total nodes: $O(bm)$

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>1 = (b-1)</td>
</tr>
<tr>
<td>3</td>
<td>1 = (b-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. If we permit infinite loops.
  - Solution 1: set the maximum depth limit $m$
  - Solution 2: prevent occurrence of cycles
- **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 $m$

Setting the maximum depth of the depth-first search

- Setting the maximum depth of the search tree avoids pitfalls of depth first search
- Use cutoff on the maximum depth of the tree
- **Problem:** How to pick the maximum depth?

- **Assume:** we have a traveler problem with 20 cities
- How to pick the maximum tree depth?
Setting the maximum depth of the depth-first search

- Setting the maximum depth of the search tree avoids pitfalls of depth first search
- Problem: How to pick the maximum depth?

- Assume: we have a traveler problem with 20 cities
  - How to pick the maximum tree depth?
  - We need to consider only paths of length < 20

- Limited depth DFS
- Time complexity: \(O(b^m)\)
- Memory complexity: \(O(bm)\)

Elimination of state repeats

While searching the state space for the solution we can encounter the same state many times.

**Question:** Is it necessary to keep and expand all copies of states in the search tree?

**Two possible cases:**

(A) Cyclic state repeats
(B) Non-cyclic state repeats
Elimination of cycles

Case A: Corresponds to the path with a cycle

Question: Can the branch (path) in which the same state is visited twice ever be a part of the optimal (shortest) path between the initial state and the goal?

No!!

Branches representing cycles cannot be the part of the shortest solution and can be eliminated.
Elimination of cycles

How to check for cyclic state repeats:
Do not expand the node with the state that is the same as the state in one of its ancestors.
• Check ancestors in the tree structure
• Caveat: we need to keep the tree.

Elimination of non-cyclic state repeats

Case B: nodes with the same state are not on the same path from the initial state

Question: Is one of the nodes nodeB-1, nodeB-2 better or preferable?
Elimination of non-cyclic state repeats

**Case B**: nodes with the same state are not on the same path from the initial state

**Question**: Is one of the nodes nodeB-1, nodeB-2 better or preferable?

**Yes**, nodeB-1 represents a shorter path from the initial state to B

---

**Conclusion**: Since we are happy with the optimal solution, nodeB-2 can be eliminated. It does not affect the optimality of the solution.

**Problem**: Nodes can be encountered in different order during different search strategies.
Elimination of non-cyclic state repeats with BFS

Breadth FS is well behaved with regard to non-cyclic state repeats: nodeB-1 is always expanded before nodeB-2

• Order of expansion determines the correct elimination strategy
• we can safely eliminate the node that is associated with the state that has been expanded before

Elimination of state repeats for the BFS

For the breadth-first search (BFS)

• we can safely eliminate all second, third, fourth, etc. occurrences of the same state
• this rule covers both cyclic and non-cyclic repeats !!!

Implementation of all state repeat elimination through marking:

• All expanded states are marked
• All marked states are stored in a hash table
• Checking if the node has ever been expanded corresponds to the mark structure lookup

Use hash table to implement marking
Elimination of non-cyclic state repeats with DFS

**Depth FS:** nodeB-2 can be expanded before nodeB-1
- The order of node expansion does not imply correct elimination strategy
- we need to remember the length of the path between nodes to safely eliminate them

Elimination of all state redundancies

**General strategy:** A node is redundant if there is another node with exactly the same state and a shorter path from the initial state
- Works for any search method
- Uses additional path length information

**Implementation: hash table with the minimum path value:**
- The new node is redundant and can be eliminated if
  - it is in the hash table (it is marked), and
  - its path is longer or equal to the value stored.
- Otherwise the new node cannot be eliminated and it is entered together with its value into the hash table. (if the state was in the hash table the new path value is better and needs to be overwritten.)