CS 1571 Introduction to AI
Lecture 5

Uninformed search methods II.

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

Announcements

Homework assignment 1 is out
• Due on Thursday before the lecture

Course web page:
http://www.cs.pitt.edu/~milos/courses/cs1571/
Uninformed methods

- Uninformed search methods use only information available in the problem definition
  - Breadth-first search (BFS)
  - Depth-first search (DFS)
  - Iterative deepening (IDA)
  - Bi-directional search

- For the minimum cost path problem:
  - Uniform cost search

Breadth first search (BFS)

- The shallowest node is expanded first
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**:
  \[ O(b^d) \]
  same as time - every node is kept in the memory

Depth-first search (DFS)

- The deepest node is expanded first
- Backtrack when the path cannot be further expanded
Properties of depth-first search

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

• **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:**
  \[ O(bl) \]
  linear in the maximum depth of the search tree \( m \)

Limited-depth depth first search

• How to eliminate infinite depth first exploration?
• Put the limit \( l \) on the depth of the depth-first exploration

Limit \( l=2 \)

\[ l \]

\[ l \]

\[ \text{Not explored} \]

• **Time complexity:**
  \[ O(b^l) \]
  \( l \) - is the given limit

• **Memory complexity:**
  \[ O(bl) \]
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

Iterative deepening algorithm (IDA)

- Based on the idea of the limited-depth search, but
- It resolves the difficulty of knowing the depth limit ahead of time.

**Idea:** try all depth limits in an increasing order.

That is, search first with the depth limit \( l=0 \), then \( l=1, l=2 \), and so on until the solution is reached

**Iterative deepening** combines advantages of the depth-first and breadth-first search with only moderate computational overhead
Iterative deepening algorithm (IDA)

- Progressively increases the limit of the limited-depth depth-first search

<table>
<thead>
<tr>
<th>Limit</th>
<th>Depth</th>
<th>Cities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit 0</td>
<td>0</td>
<td>Arad, Zerind, Sibiu, Timisoara</td>
</tr>
<tr>
<td>Limit 1</td>
<td>1</td>
<td>Arad, Sibiu, Timisoara</td>
</tr>
<tr>
<td>Limit 2</td>
<td>2</td>
<td>Arad, Sibiu, Timisoara</td>
</tr>
</tbody>
</table>

Iterative deepening

**Cutoff depth = 0**
Iterative deepening

Cutoff depth = 0

Iterative deepening

Cutoff depth = 1
Iterative deepening

Cutoff depth = 1
Iterative deepening

Cutoff depth = 1

Iterative deepening

Cutoff depth = 1
Iterative deepening

Cutoff depth = 2
Iterative deepening

Cutoff depth = 2

[Diagram showing a tree structure with cities connected through directed edges, starting from Arad and including connections to Zerind, Sibiu, and Timisoara, with further branches to other cities like Oradea, Fagaras, and Lugoj.]
Iterative deepening

Cutoff depth = 2

Iterative deepening

Cutoff depth = 2
Properties of IDA

• Completeness: Yes. The solution is reached if it exists.
  (the same as BFS when limit is always increased by 1)

• Optimality: Yes, for the shortest path.
  (the same as BFS)

• Time complexity:
  ?

• Memory (space) complexity:
  ?
**Properties of IDA**

- **Completeness**: Yes. The solution is reached if it exists.
  (the same as BFS)

- **Optimality**: Yes, for the shortest path.
  (the same as BFS)

- **Time complexity**: 
  \[ O(1) + O(b^1) + O(b^2) + \ldots + O(b^d) = O(b^d) \]
  exponential in the depth of the solution \( d \)
  worse than BFS, but asymptotically the same

- **Memory (space) complexity**: 
  ?
Properties of IDA

- **Completeness:** Yes. The solution is reached if it exists.
  (the same as BFS)
- **Optimality:** Yes, for the shortest path.
  (the same as BFS)
- **Time complexity:**
  \[ O(1) + O(b^1) + O(b^2) + \ldots + O(b^d) = O(b^d) \]
  exponential in the depth of the solution \( d \)
  worse than BFS, but asymptotically the same
- **Memory (space) complexity:**
  \[ O(db) \]
  much better than BFS
**Bi-directional search**

- In some search problems we want to find the path from the initial state to the unique goal state (e.g. traveler problem)
- **Bi-directional search idea:**
  - Search both from the initial state and the goal state;
  - Use inverse operators for the goal-initiated search.

Why bidirectional search? What is the benefit? Assume BFS.
- Cut the depth of the search space by half

\[ O(b^{d/2}) \quad \text{Time and memory complexity} \]
Bi-directional search

Why bidirectional search? Assume BFS.
• It cuts the depth of the search tree by half.

What is necessary?
• Merge the solutions.

How? The hash structure remembers the side of the tree the state was expanded first time. If the same state is reached from other side we have a solution.
Minimum cost path search

Traveler example with distances [km]

Optimal path: the shortest distance path from Arad to Bucharest

Searching for the minimum cost path

- General minimum cost path-search problem:
  - adds weights or costs to operators (links)

  “Intelligent” expansion of the search tree should be driven by the cost of the current (partially) built path

  Path cost function \( g(n) \); path cost from the initial state to \( n \)

Search strategy:

- Expand the leaf node with the minimum \( g(n) \) first.
  - When operator costs are all equal to 1 it is equivalent to BFS
- The basic algorithm for finding the minimum cost path:
  - Dijkstra’s shortest path
- In AI, the strategy goes under the name
  - Uniform cost search
Uniform cost search

- Expand the node with the minimum path cost first
- Implementation: a priority queue
Uniform cost search

queue

\[ g(n) \]

<table>
<thead>
<tr>
<th>Location</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timisoara</td>
<td>118</td>
</tr>
<tr>
<td>Sibiu</td>
<td>140</td>
</tr>
<tr>
<td>Oradea</td>
<td>146</td>
</tr>
<tr>
<td>Arad</td>
<td>150</td>
</tr>
</tbody>
</table>

\[ \text{Arad} \rightarrow \text{Zerind} \rightarrow \text{Sibiu} \rightarrow \text{Timisoara} \]

\[ \text{Arad} \rightarrow \text{Oradea} \]

\[ (150, 146) \]

---

Uniform cost search

queue

\[ g(n) \]

<table>
<thead>
<tr>
<th>Location</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sibiu</td>
<td>140</td>
</tr>
<tr>
<td>Oradea</td>
<td>146</td>
</tr>
<tr>
<td>Arad</td>
<td>150</td>
</tr>
<tr>
<td>Lugoj</td>
<td>129</td>
</tr>
<tr>
<td>Arad</td>
<td>236</td>
</tr>
</tbody>
</table>

\[ \text{Arad} \rightarrow \text{Zerind} \rightarrow \text{Sibiu} \rightarrow \text{Timisoara} \]

\[ \text{Arad} \rightarrow \text{Oradea} \]

\[ (150, 146, 236, 229) \]
Properties of the uniform cost search

• Completeness: Yes, assuming that operator costs are non-negative (the cost of path never decreases)
  \[ g(n) \leq g(\text{successor } n) \]

• Optimality: Yes. Returns the least-cost path.

• Time complexity:
  number of nodes with the cost \( g(n) \) smaller than the optimal cost

• Memory (space) complexity:
  number of nodes with the cost \( g(n) \) smaller than the optimal cost
Elimination of state repeats

Idea:
• A node is redundant and can be eliminated if there is another node with exactly the same state and a shorter path from the initial state

Assuming positive costs:
• If the state has already been expanded, is there a shorter path to that node?

Implementation:
• Marking with the hash table