Uninformed search methods III.
Informed search methods.

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

Announcements

Homework assignment 2 is out
• in the electronic form on the course web page
• Due on Thursday, September 20, 2007 before the class
• Two parts:
  – Pen and pencil part
  – Programming part (Puzzle 8)

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

Properties of IDA

- **Completeness**: ?
- **Optimality**: ?
- **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:**
  \[ O(db) \]
  much better than BFS

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
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.

---

**Bi-directional search**

Why bidirectional search? What is the benefit? Assume BFS.
- ?
Bi-directional search

Why bidirectional search? What is the benefit? Assume BFS.

- Cut the depth of the search space by half

Initial state  
Goal state

\[ \frac{d}{2} \quad \frac{d}{2} \]

\[ O\left( b^{d/2} \right) \]  
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.

Initial state Goal state

Equal ?

• How?

Bi-directional search

Why bidirectional search? Assume BFS.
• It cuts the depth of the search tree by half.
What is necessary?
• Merge the solutions.

Initial state Goal state

Equal ?

• 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**: priority queue

![Uniform cost search diagram](image)
Uniform cost search

queue

$g(n)$

<table>
<thead>
<tr>
<th>City</th>
<th>$g(n)$</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>

Arad

Zerind 75

Sibiu 140

Timisoara 118

queue

$g(n)$

<table>
<thead>
<tr>
<th>City</th>
<th>$g(n)$</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>

Arad

Zerind 75

Sibiu 140

Timisoara 118

queue

$g(n)$

<table>
<thead>
<tr>
<th>City</th>
<th>$g(n)$</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>

Arad

Zerind 75

Sibiu 140

Timisoara 118
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
Informed (heuristic) search methods

Additional information to guide the search

- **Uninformed search methods**
  - use only the information from the problem definition; and
  - past explorations, e.g. cost of the path generated so far.

- **Informed search methods**
  - incorporate additional measure of a potential of a specific state to reach the goal
  - a potential of a state (node) to reach a goal is measured through a **heuristic function**

- Heuristic function is denoted $h(n)$
Evaluation-function driven search

- A search strategy can be defined in terms of a node evaluation function
- **Evaluation function**
  - Denoted $f(n)$
  - Defines the desirability of a node to be expanded next

- **Evaluation-function driven search:** expand the node (state) with the best evaluation-function value
- **Implementation:** priority queue with nodes in the decreasing order of their evaluation function value

---

Uniform cost search

- **Uniform cost search** (*Dijkstra’s shortest path)*:
  - A special case of the evaluation-function driven search
  
  \[ f(n) = g(n) \]

- **Path cost function** $g(n)$;
  - path cost from the initial state to $n$

- **Uniform-cost search**:
  - Can handle general minimum cost path-search problem:
  - weights or costs associated with operators (links).

- **Note:** Uniform cost search relies on the problem definition only
  - It is an uninformed search method
Best-first search

Best-first search
• incorporates a **heuristic function**, \( h(n) \), into the evaluation function \( f(n) \) to guide the search.

**Heuristic function:**
• Measures a potential of a state (node) to reach a goal
• Typically in terms of some distance to a goal estimate

**Example of a heuristic function:**
• Assume a shortest path problem with city distances on connections
• Straight-line distances between cities give additional information we can use to guide the search

---

**Example: traveler problem with straight-line distance information**

- **Straight-line distances** give an estimate of the cost of the path between the two cities
Best-first search

Best-first search
• incorporates a **heuristic function**, \( h(n) \), into the evaluation function \( f(n) \) to guide the search.
• **heuristic function**: measures a potential of a state (node) to reach a goal

Special cases (differ in the design of evaluation function):
– **Greedy search**
  \[ f(n) = h(n) \]
– **A* algorithm**
  \[ f(n) = g(n) + h(n) \]
  + **iterative deepening** version of A*: IDA*