Uninformed search methods II.

Homework assignment 1 is out
• Due on Wednesday, September 11, 2019 before the lecture
• Report and programming part:
  – Programming part involves Puzzle 8 problem.
• Assignment (programs and reports) must be done individually not collaboratively!!!

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

Homework submission:
• Electronic via CourseWeb
• Separate submission of the report and programs
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: \( O(b^d) \)
  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

Properties of depth-first search

- Completeness: ?
- Optimality (shortest path): ?
- Time complexity: ?
- Memory (space) complexity: ?
**Properties of depth-first search**

- **Completeness**: No. If infinite loops can occur.
  - **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 \( m \)

**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
Should we eliminate the second A?
Why?

Case A: Corresponds to the path with a cycle

A branch of the tree representing a path with a cycle cannot be the part of the shortest solution and can be safely eliminated.
Elimination of non-cyclic state repeats

Should we eliminate nodeB-2?
Why?

A state B is reached by a longer than optimal path than it cannot be the part of the shortest solution and can be safely eliminated.
How to eliminate suboptimal A, B nodes in BFS?

Simple strategy:
- eliminate the node that is associated with the state that has been expanded before
State repeats and DFS

Root of the search tree

nodeB-1

nodeB-2

nodeB-3

How to avoid exploration of suboptimal nodes in DFS?

Caveat: The order of node expansion does not imply correct elimination strategy

Solution: we need to remember the length of the path in order to safely eliminate any of the nodes
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

Limited-depth depth first search

- Put the limit \( l \) on the depth of the depth-first exploration
- The limit is set externally and may not cover the solution

Limit \( l=2 \)

\[ l \]

Not explored

\[ O(b^l) \]

\[ O(bl) \]

\( l \) is a given limit
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

```
Limit 0

Limit 1

Limit 2

...```

---
Iterative deepening

Cutoff depth = 0
Iterative deepening

Cutoff depth = 1

CS 1571 Intro to AI

M. Hauskrecht
Iterative deepening

Cutoff depth = 1
Iterative deepening

Cutoff depth = 1

Iterative deepening

Cutoff depth = 2
Iterative deepening

Cutoff depth = 2

CS 1571 Intro to AI
M. Hauskrecht
Iterative deepening

Cutoff depth = 2

Iterative deepening

Cutoff depth = 2
Iterative deepening

Cutoff depth = 2

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 when limit is always increased by 1)

• **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**:  
  \[ O(1) + O(b) + O(b^2) + \ldots + O(b^d) = O(b^d) \]  
  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
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.

- Cuts the depth of the search space by half

Initial state

Goal state

\[ d/2 \quad d/2 \]

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

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

**Caveat:** Merge the solutions.

- How?

---

**Bi-directional search**

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

**Caveat:** 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 the other side we have a solution.
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

Minimum cost path search

Traveler example with distances [km]

Optimal path: the shortest distance path between the initial and destination city
Searching for the minimum cost path

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

- **Search strategy:**
  - “Intelligent” expansion of the search tree should be driven by the cost of the current (partially) built path

- **Implementation:**
  - **Path cost function for node** $n$: $g(n)$
    - length of the path represented by the search tree branch starting at the root of the tree (initial state) to $n$
  - **Search strategy**:
    - Expand the leaf node with the minimum $g(n)$ first
    - Can be implemented by a priority queue

The basic algorithm for finding the minimum cost path:
- **Dijkstra’s shortest path**

In AI, the strategy goes under the name
- **Uniform cost search**

- **Note:**
  - When operator costs are all equal to 1 the uniform cost search is equivalent to the breadth first search BFS
Uniform cost search

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

![Diagram of Uniform cost search](image)

- **Arad**
- **Zerind**
- **Sibiu**
- **Timisoara**

- **g(n)** values:
  - Zerind: 75
  - Timisoara: 118
  - Sibiu: 140

![Diagram of Uniform cost search](image)
Uniform cost search

g(n)

queue

Timisoara 118
Sibiu 140
Oradea 146
Arad 150

CS 1571 Intro to AI

M. Hauskrecht

Uniform cost search

queue

Sibiu 140
Oradea 146
Arad 150
Lugoj 129
Arad 236

CS 1571 Intro to AI

M. Hauskrecht
Properties of the uniform cost search

- **Completeness:** assume that operator costs are non-negative

- **Optimality:**

- **Time complexity:**

- **Memory (space) complexity:**

- **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.

\[ g(\text{nodeB-1}) = 120 \]
\[ g(\text{nodeB-2}) = 95 \]
**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.

\[ g(\text{nodeB-1}) = 120 \]
\[ g(\text{nodeB-2}) = 95 \]

**Assuming positive costs:**
- If the state has already been expanded, is there a shorter path to that node? **No!**

**Implementation:** remember expanded nodes in the hash table.