Uninformed Search

Alice Gao
Lecture 3

Based on work by K. Leyton-Brown, K. Larson, and P. van Beek

Outline

Learning goals

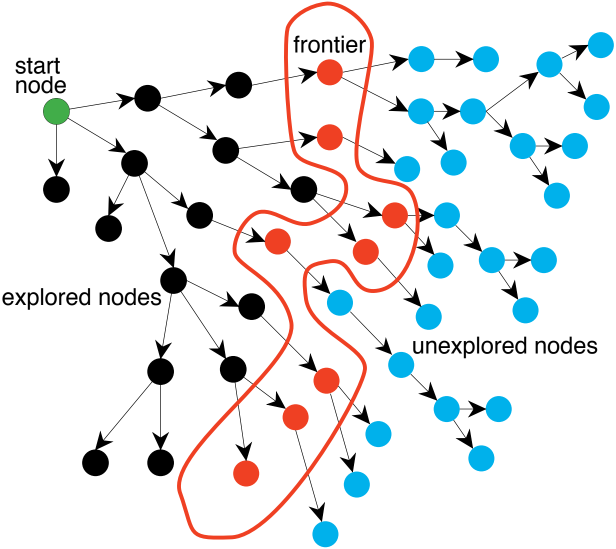
By the end of the lecture, you should be able to

- Describe what it means to expand a node and what is a frontier in a search tree.
- Define/trace/implement search algorithms (with/without cost) (dealing with repeated states)
- Define completeness, optimality, time complexity and space complexity.
- Determine properties of search algorithms: completeness, optimality, time and space complexity.
- Given a scenario, explain why it is appropriate or not appropriate to use a particular search algorithm.

Terminologies

- A search tree: the nodes are the states, the arcs are actions, and the root is the start state.
- Expanding a node means applying every legal action to the current state to generate a set of new states.
- The frontier contains the set of all leaf nodes available for expansion.

Problem Solving by Graph Searching



Graph Search Algorithm

Algorithm 1 Generic graph search algorithm

```
put the start state in the frontier
while frontier is not empty do
    remove a node from the frontier
    if the node contains a goal state then
        return the solution
    end if
    generate all the successors of the node
    add every successor of the node to the frontier
end while
return failure
```

The Search Strategy

Search strategy: which node do we remove from the frontier?

- Breadth-first search treats the frontier as a queue (FIFO).
- Depth-first search treats the frontier as a stack (LIFO).
- Informed search treats the frontier as a priority queue.

Evaluating an Algorithm's Performance

Definition (complete)

If a solution exists, a complete algorithm is
guaranteed to find a solution within a finite amount of time.

Definition (optimal)

If a solution exists and an algorithm finds a solution,
then the first solution found by an optimal algorithm is
the solution with the lowest cost.

Evaluating an Algorithm's Performance

Definition (time complexity)

The time complexity of a search algorithm is an expression for the worst-case amount of time it will take to run, expressed in terms of b, d, and m.

Definition (space complexity)

The space complexity of a search algorithm is an expression for the worst-case amount of memory that the algorithm will use, expressed in terms of b, d, and m.

Useful definitions:

- b: the maximum branching factor (may be infinite).
- d: the depth of the shallowest goal node (finite).
- m: the maximum path length (may be infinite).

Breadth-First Search

Algorithm 2 Breadth-First Search

```
put the start state in the frontier
while frontier is not empty do
    remove the oldest node added to the frontier
    if the node contains a goal state then
        return the solution
    end if
    generate all the successors of the node
    add every successor of the node to the frontier
end while
return failure
```

Breadth-First Search

Treats the frontier as a queue (FIFO).
Expands the shallowest node in the frontier.

- Complete?
- Optimal? Yes if all arc costs are the same.
- Time complexity: $O(b^d)$
- Space complexity:

CQ: Is BFS complete?

CQ: Is BFS complete?

(A) Yes.
(B) No.
(C) The answer depends on the branching factor.
(D) The answer depends on whether there are infinite paths in the search tree.

CQ: Space complexity of BFS

CQ: What is the space complexity of BFS?

(A) $O(bd)$

(B) $O(b^d)$

(C) $O(bm)$

(D) $O(b^m)$

Using Breadth-First Search

Would you use Breadth-First Search in any scenario below?

1. Memory is limited.
2. All solutions are deep in the tree.
3. There are infinite paths in the tree.
4. The branching factor is large.
5. We must find the shallowest goal node.

Depth-First Search

Algorithm 3 Depth-First Search

```
put the start state in the frontier
while frontier is not empty do
    remove the newest node added to the frontier
    if the node contains a goal state then
        return the solution
    end if
    generate all the successors of the node
    add every successor of the node to the frontier
end while
return failure
```

Depth-First Search

Treats the frontier as a stack (LIFO).
Expands the deepest node in the frontier.

- Complete?
- Optimal? No.
- Time complexity: $O(b^m)$
- Space complexity:

CQ: Is DFS complete?

CQ: Is DFS complete?

(A) Yes.

(B) No.

(C) The answer depends on the branching factor.

(D) The answer depends on whether there are infinite paths in the search tree.

CQ: Space complexity of DFS

CQ: What is the space complexity of DFS?

(A) $O(bd)$

(B) $O(b^d)$

(C) $O(bm)$

(D) $O(b^m)$

Using Depth-First Search

Would you use Depth-First Search in any scenario below? Why?

1. Memory is limited.
2. Some paths have infinite lengths.
3. The graph contains cycles.
4. Some solutions are very shallow.

Handling Repeated States

- Store visited states in a hash table.
- How would the properties of DFS change if we prune visited states?

Comparing BFS and DFS

- What are the advantages of BFS over DFS?
- What are the advantages of DFS over BFS?

CQ: BFS or DFS?

CQ: Suppose that some solutions are very deep and some solutions are very shallow in the search tree of a problem. Which of BFS and DFS should we use to solve this problem?

(A) BFS

(B) DFS

(C) Both of BFS and DFS

(D) Neither of BFS and DFS

CQ: BFS or DFS?

CQ: If we have very limited memory and the search graph of a problem contains cycles, which of BFS and DFS should we use to solve this problem?

(A) BFS

(B) DFS

(C) Both of BFS and DFS

(D) Neither of BFS and DFS

The best of BFS and DFS

BFS	DFS
requires lots of space	requires little space
complete	not complete

The best of both worlds:

- Run DFS until level l.
- If no solution found, try level $l + 1$, $l + 2$, etc.

Iterative Deepening Search

Algorithm 4 Iterative Deepening Search

```
for l = 0 to ∞ do
    Perform depth-first search up to maximum depth limit l
end for
```

Iterative-Deepening Search

For depth limit $l = 0, 1, \ldots$
Perform a depth-first search with maximum depth l.

- Complete? Yes if the branching factor is finite.
- Optimal? Yes if all arc costs are the same.
- Time complexity: $O(b^d)$
- Space complexity: $O(bd)$

Using Iterative-Deepening Search

- How is IDS similar to/different from DFS?
- How is IDS similar to/different from BFS?
- Is it too costly for IDS to generate states in the upper levels multiple times?

Comparing BFS, DFS, and IDS

Algorithm	Complete?	Optimal?	Time	Space
IDS				
DFS				
BFS				

* if the branching factor is finite.

** if the search tree does not have infinite paths.

CQ: Comparing IDS and DFS

CQ: Suppose that the search tree of a problem has no infinite paths. Which one of DFS and IDS should we use to solve this problem?

(A) DFS is better than IDS.

(B) IDS is better than DFS.

(C) DFS and IDS are the same.

Revisiting the learning goals

By the end of the lecture, you should be able to

- Describe what it means to expand a node and what is a frontier in a search tree.
- Define/trace/implement search algorithms (with/without cost) (dealing with repeated states)
- Define completeness, optimality, time complexity and space complexity.
- Determine properties of search algorithms: completeness, optimality, time and space complexity.
- Given a scenario, explain why it is appropriate or not appropriate to use a particular search algorithm.