Breadth first search
Uniform cost search

Robert Platt
Northeastern University

Some images and slides are used from:
1. CS188 UC Berkeley
2. RN, AIMA
What is graph search?

Start state

Goal state
What is a graph?

Graph: \( G = (V, E) \)

Vertices: \( V \)

Edges: \( E \)

Directed graph

\[ V = \{A, B, C\} \]
\[ E = \{(B, A), (A, C), (B, C), (C, B)\} \]
What is a graph?

Graph: \( G = (V, E) \)

Vertices: \( V \)

Edges: \( E \)

Undirected graph

\[
V = \{A, B, C, D\}
\]

\[
E = \{\{A, C\}, \{A, B\}, \{C, D\}, \{B, D\}, \{C, B\}\}
\]
Graph search

**Given**: a graph, G

**Problem**: find a path from A to B

- A: start state
- B: goal state
Graph search

Given: a graph, $G$

Problem: find a path from $A$ to $B$

– $A$: start state
– $B$: goal state

How?
A search tree

Start at A
A search tree

Successors of A
A search tree

![Diagram of a search tree with nodes and edges showing parent-child relationships.]

**Successors of A**

- **Parent of A**
  - **Children of A**
    - Z
    - T
    - S

---

**Notes:**
- The diagram illustrates a tree structure where each node has zero or more children, and every node except the root has exactly one parent.
A search tree

Let's expand S next
A search tree

Successors of $S$
A search tree

A was already visited!
A search tree

So, prune it!
In what order should we expand states?

- here, we expanded S, but we could also have expanded Z or T
- different search algorithms expand in different orders
Breadth first search (BFS)

Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)
Breadth first search (BFS)
Breadth first search (BFS)

\[ A \quad \text{Start node} \]
Breadth first search (BFS)
Breadth-first search (BFS)
Breadth first search (BFS)
Breadth first search (BFS)

**Fringe**

We're going to maintain a queue called the **fringe**

– initialize the fringe as an empty queue
Breadth first search (BFS)

- add A to the fringe
Breadth first search (BFS)

-- remove A from the fringe

-- add successors of A to the fringe
Breadth first search (BFS)

-- remove $B$ from the fringe

-- add successors of $B$ to the fringe
Breadth first search (BFS)

-- remove C from the fringe

-- add successors of C to the fringe
Which state gets removed next from the fringe?
Breadth first search (BFS)

Fringe
D
E
F
G

Which state gets removed next from the fringe?

What kind of a queue is this?
Breadth first search (BFS)

Which state gets removed next from the fringe?

What kind of a queue is this?

FIFO Queue!
(first in first out)
Breadth first search (BFS)

```plaintext
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)

    frontier ← a FIFO queue with node as the only element
    explored ← an empty set

    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier) /* chooses the shallowest node in frontier */
        add node.STATE to explored

        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
        end for
    end loop
```

Figure 3.11  Breadth-first search on a graph.
Breadth first search (BFS)

function BREADTH-FIRST-SEARCH( problem ) returns a solution, or failure

node ← a node with State = problem.INITIAL-STATE, Path-Cost = 0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ← a FIFO queue with node as the only element
explored ← an empty set
loop do
  if EMPTY?(frontier) then return failure
  node ← POP(frontier) /* chooses the shallowest node in frontier */
  add node.STATE to explored
  for each action in problem.ACTIONS(node.STATE) do
    child ← CHILD-NODE( problem, node, action )
    if child.STATE is not in explored or frontier then
      if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
      frontier ← INSERT(child, frontier)
  end loop
end loop

Figure 3.11  Breadth-first search on a graph.

What is the purpose of the explored set?
Is BFS complete?
  – is it guaranteed to find a solution if one exists?
BFS Properties

Is BFS complete?
– is it guaranteed to find a solution if one exists?

What is the time complexity of BFS?
– how many states are expanded before finding a sol'n?
  – b: branching factor
  – d: depth of shallowest solution
  – complexity = ???
BFS Properties

Is BFS complete?
– is it guaranteed to find a solution if one exists?

What is the time complexity of BFS?
– how many states are expanded before finding a sol'n?
  – b: branching factor
  – d: depth of shallowest solution
  – complexity = $O(b^d)$
BFS Properties

Is BFS complete?
– is it guaranteed to find a solution if one exists?

What is the time complexity of BFS?
– how many states are expanded before finding a sol'n?
  – b: branching factor
  – d: depth of shallowest solution
  – complexity = $O(b^d)$

What is the space complexity of BFS?
– how much memory is required?
  – complexity = ???
BFS Properties

Is BFS **complete**?  
– is it guaranteed to find a solution if one exists?

What is the **time complexity** of BFS?  
– how many states are expanded before finding a sol'n?  
  – $b$: branching factor  
  – $d$: depth of shallowest solution  
  – complexity = $O(b^d)$

What is the **space complexity** of BFS?  
– how much memory is required?  
  – complexity = $O(b^d)$
BFS Properties

Is BFS complete?
– is it guaranteed to find a solution if one exists?

What is the time complexity of BFS?
– how many states are expanded before finding a sol'n?
  – b: branching factor
  – d: depth of shallowest solution
  – complexity = $O(b^d)$

What is the space complexity of BFS?
– how much memory is required?
  – complexity = $O(b^d)$

Is BFS optimal?
– is it guaranteed to find the best solution (shortest path)?
Another BFS example...
Uniform Cost Search (UCS)

Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)
Uniform Cost Search (UCS)

Notice the distances between cities
Uniform Cost Search (UCS)

Notice the distances between cities – does BFS take these distances into account?
Uniform Cost Search (UCS)

Notice the distances between cities
– does BFS take these distances into account?
– does BFS find the path w/ shortest milage?
Uniform Cost Search (UCS)

Notice the distances between cities
– does BFS take these distances into account?
– does BFS find the path w/ shortest milage?
– compare S-F-B with S-R-P-B. Which costs less?
Uniform Cost Search (UCS)

Notice the distances between cities:
- does BFS take these distances into account?
- does BFS find the path with the shortest mileage?
- compare S-F-B with S-R-P-B. Which costs less?

How do we fix this?
Uniform Cost Search (UCS)

Notice the distances between cities – does BFS take these distances into account? – does BFS find the path with shortest mileage? – compare S-F-B with S-R-P-B. Which costs less?

How do we fix this? UCS!
Uniform Cost Search (UCS)

Same as BFS except: expand node with smallest path cost

Length of path
Uniform Cost Search (UCS)

Same as BFS except: expand node with smallest path cost

Length of path

Cost of going from state $A$ to $B$: $c(A, B)$

Minimum cost of path going from start state to $B$: $g(B)$
Uniform Cost Search (UCS)

Same as BFS except: expand node w/ smallest path cost

Cost of going from state $A$ to $B$: $c(A, B)$

Minimum cost of path going from start state to $B$: $g(B)$

BFS: expands states in order of hops from start

UCS: expands states in order of $g(s)$
Uniform Cost Search (UCS)

Same as BFS except: expand node with smallest path cost

Length of path

Cost of going from state $A$ to $B$: $c(A, B)$

Minimum cost of path going from start state to $B$: $g(B)$

BFS: expands states in order of hops from start

UCS: expands states in order of path cost

How?
Uniform Cost Search (UCS)

Simple answer: change the FIFO to a priority queue
– the priority of each element in the queue is its path cost.
Uniform Cost Search (UCS)
### UCS

<table>
<thead>
<tr>
<th>Fringe</th>
<th>Path Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
</tbody>
</table>

Explored set:
Explored set: A
Explored set: A, Z

<table>
<thead>
<tr>
<th>Fringe</th>
<th>Path Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>140</td>
</tr>
<tr>
<td>T</td>
<td>118</td>
</tr>
<tr>
<td>Z</td>
<td>75</td>
</tr>
<tr>
<td>T</td>
<td>146</td>
</tr>
</tbody>
</table>
Explored set: A, Z, T
Explored set: A, Z, T, S
Explored set: A, Z, T, S
Explored set: A, Z, T, S, R
When does this end?

UCS


When does this end?
– when the goal state is removed from the queue
When does this end?
- when the goal state is removed from the queue
- NOT when the goal state is expanded

function UNIFORM-COST-SEARCH(problem) returns a solution, or failure

node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
frontier ← a priority queue ordered by PATH-COST, with node as the only element
explored ← an empty set
loop do
  if EMPTY?(frontier) then return failure
  node ← POP(frontier) /* chooses the lowest-cost node in frontier */
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  add node.STATE to explored
  for each action in problem.ACTIONS(node.STATE) do
    child ← CHILD-NODE(problem, node, action)
    if child.STATE is not in explored or frontier then
      frontier ← INSERT(child, frontier)
    else if child.STATE is in frontier with higher PATH-COST then
      replace that frontier node with child

Figure 3.14 Uniform-cost search on a graph. The algorithm is identical to the general graph search algorithm in Figure 3.7, except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered. The data structure for frontier needs to support efficient membership testing, so it should combine the capabilities of a priority queue and a hash table.
Is UCS **complete**?
– is it guaranteed to find a solution if one exists?

What is the **time complexity** of UCS?
– how many states are expanded before finding a sol'n?
  – $b$: branching factor
  – $C^*$: cost of optimal sol'n
  – $e$: min one-step cost
  – complexity = $O\left(b^{C^* / e}\right)$

What is the **space complexity** of BFS?
– how much memory is required?
  – complexity = $O\left(b^{C^* / e}\right)$

Is BFS optimal?
– is it guaranteed to find the best solution (shortest path)?
Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)
UCS vs BFS

**Strategy:** expand a shallowest node first

**Implementation:** Fringe is a FIFO queue

Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)
- Remember: UCS explores increasing cost contours

- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

- We’ll fix that soon!

Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)