CS540
Uninformed Search

Xiaojin Zhu
jerryzhu@cs.wisc.edu

Computer Sciences Department
University of Wisconsin, Madison
Main messages

• Many AI problems can be formulated as search.

• Iterative deepening is good when you don’t know much.
The search problem

• State space $S$ : all valid configurations
• Initial states (nodes) $I = \{(\text{CSDF,})\} \subseteq S$
  ▪ Where’s the boat?
• Goal states $G = \{,(\text{CSD})\} \subseteq S$
• Successor function $\text{succs}(s) \subseteq S$ : states reachable in one step (one arc) from $s$
  ▪ $\text{succs}((\text{CSDF,})) = \{ (\text{CD, SF}) \}$
  ▪ $\text{succs}((\text{CDF,S})) = \{ (\text{CD,FS}), (\text{D,CFS}), (\text{C, DFS}) \}$
• Cost$(s,s') = 1$ for all arcs. (weighted later)
• The search problem: find a solution path from a state in $I$ to a state in $G$
  ▪ Optionally minimize the cost of the solution.
Search examples

• 8-puzzle

- States = configurations
- Successor function = up to 4 kinds of movement
- Cost = 1 for each move
Search examples

• Water jugs: how to get 1?

• Goal? (How many goal states?)

• Successor function: fill up (from tap or other jug), empty (to ground or other jug)
Search examples

- Route finding (state? Successors? Cost weighted)
8-queens

- State: complete configuration vs. column-by-column

- Tree instead of graph
A directed graph in state space

In general there will be many generated, but un-expanded states at any given time.

One has to choose which one to expand next.
Different search strategies

- The generated, but not yet expanded states form the fringe (OPEN).
- The essential difference is which one to expand first.
- Deep or shallow?

![Diagram showing different search strategies]
Uninformed search on trees

- **Uninformed** means we only know:
  - The goal test
  - The `succs()` function
- But **not** which non-goal states are better: that would be informed search (next lecture).
- For now, we also assume `succs()` graph is a tree.
  - Won’t encounter repeated states.
  - We will relax it later.
- Search strategies: BFS, UCS, DFS, IDS, BIBFS
- Differ by what un-expanded nodes to expand
Breadth-first search (BFS)

Expand the shallowest node first

- Examine states **one** step away from the initial states
- Examine states **two** steps away from the initial states
- and so on…

ripple
Breadth-first search (BFS)

Use a queue (First-in First-out)
1. en_queue(Initial states)
2. While (queue not empty)
3.   s = de_queue()
4.   if (s==goal) success!
5.   T = succs(s)
6.   en_queue(T)
7. endWhile
Breadth-first search (BFS)

Use a **queue** (First-in First-out)

1. `en_queue(Initial states)`
2. While (queue not empty)
3. `s = de_queue()`
4. if (s==goal) success!
5. `T = succs(s)`
6. `en_queue(T)`
7. `endWhile`

![Diagram showing the breadth-first search process with a queue (fringe, OPEN) leading to the goal state A.](image)
Breadth-first search (BFS)

Use a queue (First-in First-out)

1. `en_queue(Initial states)`
2. While (queue not empty)
3. `s = de_queue()`
4. if (s==goal) success!
5. `T = succs(s)`
6. `en_queue(T)`
7. endwhile

queue (fringe, OPEN) → [CB] → A
Breadth-first search (BFS)

Use a queue (First-in First-out)
1. en_queue(Initial states)
2. While (queue not empty)
3. s = de_queue()
4. if (s==goal) success!
5. T = succs(s)
6. en_queue(T)
7. endwhile

queue (fringe, OPEN) → [EDC] → B
Breadth-first search (BFS)

Use a queue (First-in First-out)

1. en_queue(Initial states)
2. While (queue not empty)
3. s = de_queue()
4. if (s==goal) success!
5. T = succs(s)
6. en_queue(T)
7. endWhile

Queue (fringe , OPEN)
→[GFED] → C

If G is a goal, we've seen it, but we don't stop!
Breadth-first search (BFS)

Use a queue (First-in First-out)
• en_queue(Initial states)
• While (queue not empty)
  • s = de_queue()
  • if (s==goal) success!
  • T = succs(s)
  • for t in T: t.prev=s
• en_queue(T)
• endWhile

Looking stupid? Indeed. But let’s be consistent...

... until much later we pop G.

We need back pointers to recover the solution path.
Performance of BFS

- Assume:
  - the graph may be infinite.
  - Goal(s) exists and is only finite steps away.
- Will BFS find at least one goal?
- Will BFS find the least cost goal?
- Time complexity?
  - # states generated
  - Goal \( d \) edges away
  - Branching factor \( b \)
- Space complexity?
  - # states stored
Performance of BFS

Four measures of search algorithms:

- **Completeness** *(not finding all goals)*: yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing in depth), no otherwise.
- **Time** complexity (worst case): goal is the last node at radius \(d\).
  - Have to generate all nodes at radius \(d\).
  - \(b + b^2 + \ldots + b^d \sim O(b^d)\)
- **Space** complexity *(bad)*
  - Back pointers for all generated nodes \(O(b^d)\)
  - The queue / fringe (smaller, but still \(O(b^d)\))
What’s in the fringe (queue) for BFS?

- Convince yourself this is $O(b^d)$
Performance of search algorithms on trees

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>optimal</th>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-first search</td>
<td>Y</td>
<td>Y, if $^1$</td>
<td>$O(b^d)$</td>
<td>$O(b^d)$</td>
</tr>
</tbody>
</table>

1. Edge cost constant, or positive non-decreasing in depth
Performance of BFS

Four measures of search algorithms:

• **Completeness** (not finding all goals): yes, BFS will find a goal.

• **Optimality**: yes if edges cost 1 (more generally positive non-decreasing with depth), no otherwise.

• **Time complexity** (worst case): goal is the last node at radius $d$.
  - Have to generate all nodes at radius $d$.
  - $b + b^2 + \ldots + b^d \sim O(b^d)$

• **Space complexity** (bad, Figure 3.11)
  - Back points for all generated nodes $O(b^d)$
  - The queue (smaller, but still $O(b^d)$)

Solution: Uniform-cost search
Uniform-cost search

• Find the least-cost goal
• Each node has a path cost from start (= sum of edge costs along the path). Expand the least cost node first.
• Use a priority queue instead of a normal queue
  ▪ Always take out the least cost item
  ▪ Remember heap? time \( O(\log(\#\text{items in heap})) \)

That’s it*

* Complications on graphs (instead of trees). Later.
Uniform-cost search (UCS)

- Complete and optimal (if edge costs $\geq \varepsilon > 0$)
- Time and space: can be much worse than BFS
  - Let $C^*$ be the cost of the least-cost goal
  - $O(b^{C^*/\varepsilon})$, possibly $C^*/\varepsilon >> d$
### Performance of search algorithms on trees

**b**: branching factor (assume finite)  
**d**: goal depth

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-first search</td>
<td>Y</td>
<td>Y, if $^1$</td>
<td>$O(b^d)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Uniform-cost search$^2$</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^{C^*/\varepsilon})$</td>
<td>$O(b^{C^*/\varepsilon})$</td>
</tr>
</tbody>
</table>

1. edge cost constant, or positive non-decreasing in depth

- edge costs $\geq \varepsilon > 0$. $C^*$ is the best goal path cost.
General State-Space Search Algorithm

function general-search(problem, QUEUEING-FUNCTION)
    ;; problem describes the start state, operators, goal test, and
    ;; operator costs
    ;; queueing-function is a comparator function that ranks two states
    ;; general-search returns either a goal node or "failure"

    nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
    loop
        if EMPTY(nodes) then return "failure"
        node = REMOVE-FRONT(nodes)
        if problem.GOAL-TEST(node.STATE) succeeds
            then return node
        nodes = QUEUEING-FUNCTION(nodes, EXPAND(node, problem.OPERATORS))
    ;; succ(s)=EXPAND(s, OPERATORS)
    ;; Note: The goal test is NOT done when nodes are generated
    ;; Note: This algorithm does not detect loops
end
Recall the bad space complexity of BFS

Four measures of search algorithms:

- **Completeness** (not finding all goals): yes, BFS will find a goal.
- **Optimality**: yes if edges cost 1 (more generally positive non-decreasing with depth), no otherwise.
- **Time complexity** (worst case): goal is the last node at radius $d$.
  - Have to generate all nodes at radius $d$.
  - $b + b^2 + \ldots + b^d \sim O(b^d)$
- **Space complexity** (bad, Figure 3.11)
  - Back points for all generated nodes $O(b^d)$
  - The queue (smaller, but still $O(b^d)$)
Depth-first search

Expand the deepest node first

1. Select a direction, go deep to the end
2. Slightly change the end
3. Slightly change the end some more…

fan
Depth-first search (DFS)

Use a stack (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endWhile

stack (fringe) [] ⇔
What's in the fringe for DFS?

- $m = \text{maximum depth of graph from start}$
- $m(b-1) \sim O(mb)$
  (Space complexity)

- "backtracking search" even less space
  - generate siblings (if applicable)
What’s wrong with DFS?

- Infinite tree: may not find goal (incomplete)
- May not be optimal
- Finite tree: may visit almost all nodes, time complexity $O(b^m)$

c.f. BFS $O(b^d)$
## Performance of search algorithms on trees

*b*: branching factor (assume finite)  
*d*: goal depth  
*m*: graph depth

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>optimal</th>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-first search</td>
<td><strong>Y</strong></td>
<td><strong>Y, if</strong></td>
<td><strong>O(b^d)</strong></td>
<td><strong>O(b^d)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uniform-cost search²</td>
<td><strong>Y</strong></td>
<td><strong>Y</strong></td>
<td><em><em>O(b^{C</em>/\varepsilon})</em>*</td>
<td><em><em>O(b^{C</em>/\varepsilon})</em>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Depth-first search</td>
<td><strong>N</strong></td>
<td><strong>N</strong></td>
<td><strong>O(b^m)</strong></td>
<td><strong>O(bm)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. edge cost constant, or positive non-decreasing in depth  
   - edge costs $\geq \varepsilon > 0$. $C^*$ is the best goal path cost.
How about this?

1. DFS, but stop if path length > 1.
2. If goal not found, repeat DFS, stop if path length > 2.
3. And so on...

fan within ripple
Iterative deepening

- Search proceeds like BFS, but fringe is like DFS
  - Complete, optimal like BFS
  - Small space complexity like DFS
- A huge waste?
  - Each deepening repeats DFS from the beginning
  - No! $db + (d-1)b^2 + (d-2)b^3 + \ldots + b^d \sim O(b^d)$
  - Time complexity like BFS
- Preferred uninformed search method
## Performance of search algorithms on trees

**b**: branching factor (assume finite)  \hspace{1em} **d**: goal depth  \hspace{1em} **m**: graph depth

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-first search</td>
<td>Y</td>
<td>Y, if (^1)</td>
<td>(O(b^d))</td>
<td>(O(b^d))</td>
</tr>
<tr>
<td>Uniform-cost search(^2)</td>
<td>Y</td>
<td>Y</td>
<td>(O(b^{C*/\varepsilon}))</td>
<td>(O(b^{C*/\varepsilon}))</td>
</tr>
<tr>
<td>Depth-first search</td>
<td>N</td>
<td>N</td>
<td>(O(b^m))</td>
<td>(O(bm))</td>
</tr>
<tr>
<td>Iterative deepening</td>
<td>Y</td>
<td>Y, if (^1)</td>
<td>(O(b^d))</td>
<td>(O(bd))</td>
</tr>
</tbody>
</table>

1. edge cost constant, or positive non-decreasing in depth  
   - edge costs \(\geq \varepsilon > 0\). \(C^*\) is the best goal path cost.
### Performance of search algorithms on trees

**b**: branching factor (assume finite)  
**d**: goal depth  
**m**: graph depth

<table>
<thead>
<tr>
<th>Search Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-first search</td>
<td>Y</td>
<td>Y, if $^1$</td>
<td>$O(b^d)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Uniform-cost search</td>
<td></td>
<td></td>
<td>$O(b^{C*/\epsilon})$</td>
<td>$O(b^{C*/\epsilon})$</td>
</tr>
<tr>
<td>Depth-first search</td>
<td></td>
<td></td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>Iterative deepening</td>
<td>Y</td>
<td>Y, if $^1$</td>
<td>$O(b^d)$</td>
<td>$O(bd)$</td>
</tr>
</tbody>
</table>

1. edge cost constant, or positive non-decreasing in depth  
   - edge costs $\geq \epsilon > 0$. $C^*$ is the best goal path cost.

---

*How to reduce the number of states we have to generate?*
Bidirectional search

- Breadth-first search from both start and goal
- Fringes meet
- Generates $O(b^{d/2})$ instead of $O(b^d)$ nodes
Bidirectional search

- But
  - The fringes are $O(b^{d/2})$
  - How do you start from the 8-queens goals?
Performance of search algorithms on trees

<table>
<thead>
<tr>
<th>Search Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-first search</td>
<td>Y</td>
<td>Y, if $^1$</td>
<td>$O(b^d)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Uniform-cost search$^2$</td>
<td>Y</td>
<td>Y</td>
<td>$O(b^{C*}/\varepsilon)$</td>
<td>$O(b^{C*}/\varepsilon)$</td>
</tr>
<tr>
<td>Depth-first search</td>
<td>N</td>
<td>N</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>Iterative deepening</td>
<td>Y</td>
<td>Y, if $^1$</td>
<td>$O(b^d)$</td>
<td>$O(bd)$</td>
</tr>
<tr>
<td>Bidirectional search$^3$</td>
<td>Y</td>
<td>Y, if $^1$</td>
<td>$O(b^{d/2})$</td>
<td>$O(b^{d/2})$</td>
</tr>
</tbody>
</table>

1. edge cost constant, or positive non-decreasing in depth
   - edge costs $\geq \varepsilon > 0$. $C^*$ is the best goal path cost.
   - both directions BFS; not always feasible.

---

$b$: branching factor (assume finite)  
$d$: goal depth  
$m$: graph depth
If state space graph is not a tree

- The problem: repeated states

- Ignore the danger of repeated states: wasteful (BFS) or impossible (DFS). Can you see why?

- How to prevent it?
If state space graph is not a tree

• We have to remember already-expanded states (CLOSED).

• When we take out a state from the fringe (OPEN), check whether it is in CLOSED (already expanded).
  ▪ If yes, throw it away.
  ▪ If no, expand it (add successors to OPEN), and move it to CLOSED.
If state space graph is not a tree

• BFS:
  ▪ Still $O(b^d)$ space complexity, not worse
• DFS:
  ▪ Known as Memorizing DFS (MEMDFS)
    • Space and time now $O(min(N, b^M))$ – much worse!
    • N: number of states in problem
    • M: length of longest cycle-free path from start to anywhere
  ▪ Alternative: Path Check DFS (PCDFS): remember only expanded states on current path (from start to the current node)
    • Space $O(M)$
    • Time $O(b^M)$
Example

(All edges are directed, pointing downwards)
Nodes expanded by:

- Depth-First Search: S A D E G
  Solution found: S A G

- Breadth-First Search: S A B C D E G
  Solution found: S A G

- Uniform-Cost Search: S A D B C E G
  Solution found: S B G (This is the only uninformed search that worries about costs.)

- Iterative-Deepening Search: S A B C S A D E G
  Solution found: S A G
### Depth-First Search

<table>
<thead>
<tr>
<th>expanded node</th>
<th>nodes list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>{ A B C }</td>
</tr>
<tr>
<td>A</td>
<td>{ D E G B C}</td>
</tr>
<tr>
<td>D</td>
<td>{ E G B C }</td>
</tr>
<tr>
<td>E</td>
<td>{ G B C }</td>
</tr>
<tr>
<td>G</td>
<td>{ B C }</td>
</tr>
</tbody>
</table>

Solution path found is S A G  <-- this G has cost 10
Number of nodes expanded (including goal node) = 5
## Breadth-First Search

<table>
<thead>
<tr>
<th>expanded node</th>
<th>nodes list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>{ S }</td>
</tr>
<tr>
<td>A</td>
<td>{ A, B, C }</td>
</tr>
<tr>
<td>B</td>
<td>{ C, D, E, G }</td>
</tr>
<tr>
<td>C</td>
<td>{ D, E, G, G' }</td>
</tr>
<tr>
<td>D</td>
<td>{ E, G, G' }</td>
</tr>
<tr>
<td>E</td>
<td>{ G, G' }</td>
</tr>
<tr>
<td>G</td>
<td>{ G' }</td>
</tr>
</tbody>
</table>

Solution path found is S A G  <-- this G also has cost 10  
Number of nodes expanded (including goal node) = 7
### Uniform-Cost Search

<table>
<thead>
<tr>
<th>expanded node</th>
<th>nodes list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{ S }</td>
</tr>
<tr>
<td>S</td>
<td>{ A(1) B(5) C(8) }</td>
</tr>
<tr>
<td>A</td>
<td>{ D(4) B(5) C(8) E(8) G(10) }  (note, we don’t return G)</td>
</tr>
<tr>
<td>D</td>
<td>{ B(5) C(8) E(8) G(10) }</td>
</tr>
<tr>
<td>B</td>
<td>{ C(8) E(8) G(9) G(10) }</td>
</tr>
<tr>
<td>C</td>
<td>{ E(8) G(9) G(10) G(13) }</td>
</tr>
<tr>
<td>E</td>
<td>{ G(9) G(10) G(13) }</td>
</tr>
<tr>
<td>G</td>
<td>{}</td>
</tr>
</tbody>
</table>

Solution path found is $S \rightarrow B \rightarrow G$  
$<--$ this $G$ has cost 9, not 10

Number of nodes expanded (including goal node) = 7
What you should know

• Problem solving as search: state, successors, goal test
• Uninformed search
  ▭ Breadth-first search
    • Uniform-cost search
  ▭ Depth-first search
  ▭ Iterative deepening
  ▭ Bidirectional search
• Can you unify them (except bidirectional) using the same algorithm, with different priority functions?
• Performance measures
  ▭ Completeness, optimality, time complexity, space complexity