Artificial Intelligence **(Harvard CS182, Fall 2015)**

Lectures Notes on Search

Alexander Rush

1 Board 1

Finish up Machine translation
  What are the search properties?
  d, m, C^*, resolving questions.
  Draw on the board the translations
  Reminder

Name	Type	Description
Branching Factor	b	❂
Min Depth	d	❂
Max Depth	m	❂
Optimal Cost	C^*	❂

2 Board 2

Search Model

Name (AIMA)	Type	Description
State space	$\mathcal{S}$	Set of states of world.
Action space	$\mathcal{A}$	Set of actions in world.
Action model	$\textsc{Act} : \mathcal{S} \mapsto 2^{\mathcal{A}}$	Actions applicable at a state
Transition model	$\textsc{Res} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$	Result of taking action at state (stop an explain).
Initial state	$s_0 \in \mathcal{S}$	Starting State of world.
Goal test	$\textsc{Goal} : \mathcal{S} \rightarrow \{0,1\}$	Is state a goal?

(Stop and make clear the function notation)

3 Introduction

The structure of the lectures:
Last time all you ever needed to know about modeling:
This lecture:

- Algorithms!

You will implement everything in this class!

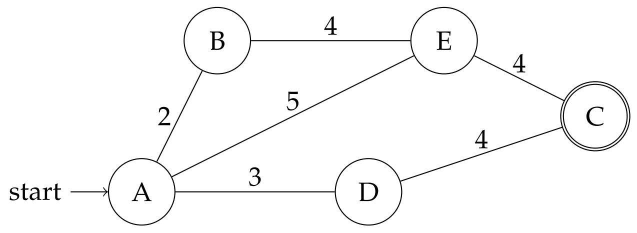
4 Summary Board (2)

Search Algorithms

- Review: Containers
- Search Algorithms
- Properties of Search algorithms
- Uninformed Search

5 Board 3

- MapQuest, google/apple maps
- Commercial routing



6 Computer

Abstract Containers

- `push`,
- `pop`,
- `empty`

Abstract: many possible implementations
(Maybe implement)
* Last-in First-out (LIFO) container. * First-in First-out (LIFO) container.

```python
class Container:
    def __init__(self): self.data = []
    def __str__(self): return str(self.data)
    def empty(self, element): return bool(self.data)

class Stack:
    def push(self, element):
        self.data = self.data + [element]
    def pop(self):
        self.data, element = self.data[:-1], self.data[-1]
        return element

class Queue:
    def push(self, element):
        self.data = [element] + self.data.insert(0, element)
    def pop(self):
        self.data, element = self.data[:-1], self.data[-1]
        return element
```

7 Board 4

All search algorithms utilize the following algorithm:

```
procedure TreeSearch(frontier)
    frontier.push(Path($s_0, \epsilon$))

    while frontier is not empty do
        $p \leftarrow$ frontier.pop()
        $s \leftarrow p_{\text{last}}$
        if Goal($s$) then return $p$

        for $a \in$ Act($s$) do
            $s' \leftarrow$ Res($s, a$)

            frontier.push($p$ extended by $(s, a)$)

```

(Show example of wraparound)
Discuss infinite paths.

8 Board 5

9 Board 8

Algorithmic Properities

- Completeness; Is the algorithm guaranteed to find a solution? (infinite paths)
- Optimality; Is the algorithm guaranteed to find the optimal solution?
- Time Complexity; What is the worst-case running time of the algorithm?

```
procedure GraphSearch(frontier)
    frontier.push(path starting with $s_0$))
    ❂ explored $\leftarrow \{\}$
    while frontier is not empty do
        $p \leftarrow$ frontier.pop()
        $s \leftarrow p_{\text{last}}$
        if Goal($s$) then return $p$
        ❂ explored $\leftarrow \{s\} \cup$ explored
        for $a \in$ Act($s$) do
            $s' \leftarrow$ Res($s, a$)
            if ❂ $s \notin$ explored   then frontier.push($p$ extended by $(s, a)$)
```

- Space Complexity; What is the worst-case memory usage of the algorithm?

Will often be different for tree-search and graph-search.

10 Board 9

Uninformed Search

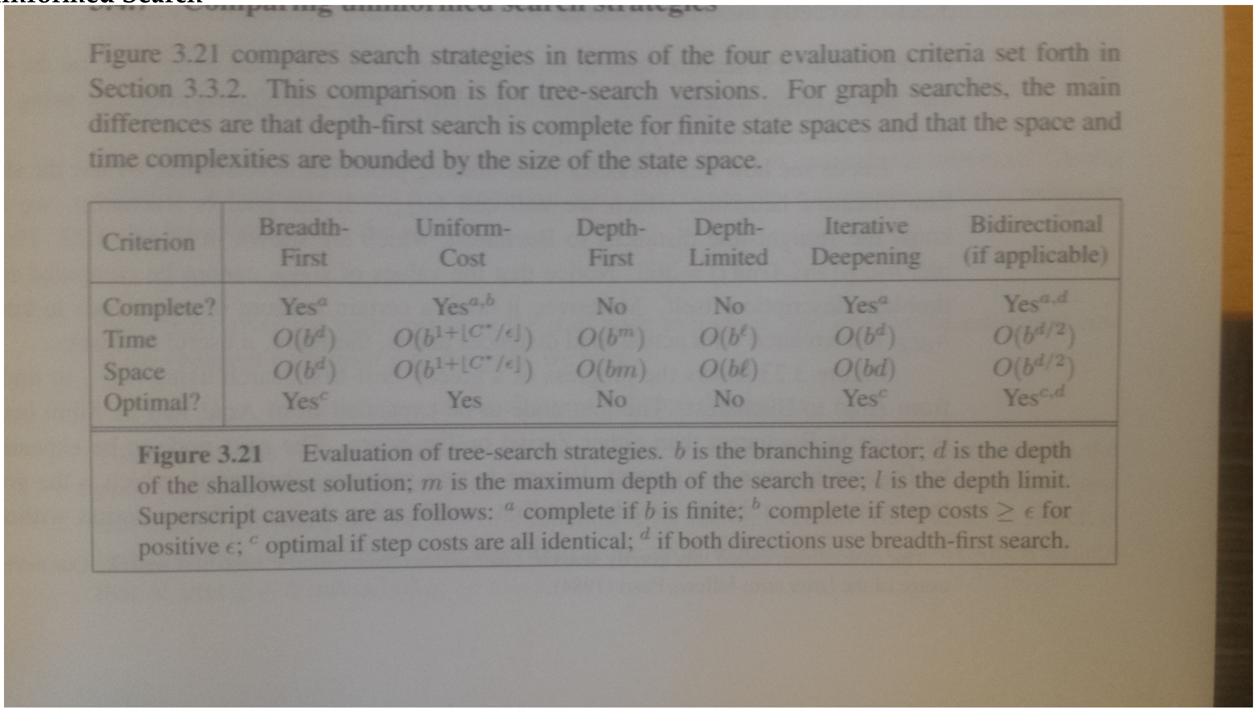


Figure 3.21 compares search strategies in terms of the four evaluation criteria set forth in Section 3.3.2. This comparison is for tree-search versions. For graph searches, the main differences are that depth-first search is complete for finite state spaces and that the space and time complexities are bounded by the size of the state space.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

- Breadth-First Search
- Depth-First Search
- Depth-Limited Search
- Iterative-Deepening Search
- Uniform-Cost Search

11 Board 10

11.1 Breadth-First Search

- Conservative-algorithm
- Explores each depth layer completely.
- FIFO Queue - Push: adds to back - Pop: removes from front

12 Board 11

iter	p	s	frontier by p_{last}	explored
0	-	-	❂ [A]	❂ {}
1	A	A	❂ [B, E, D]	❂ {A}
2	A:B	B	❂ [E, D, E]	❂ {A, B}
3	A:E	E	❂ [D, E, C]	❂ {A, B, E}
4	A:D	D	❂ [E, C, C]	❂ {A, B, E, D}
5	A:B:E	E	❂ [C, C, C]	❂ {A, B, E, D}
6	A:E:C	C	-	-

13 Board 12

Properties of Breadth-First search

(BFS explores every path of each depth in order. How many paths are there up to depth l? Well in the worst-case there are $O(b^l)$. If the shallowest goal node is at depth d, BFS will search $O(b^d)$ states.)

- Completeness: Yes. Finds solution after $O(b^d)$ steps.
- Optimality: No. Optimal solution may not be at depth d. However BFS optimal when costs are all the same.
- Time-Complexity: $O(b^d)$
- Space-Complexity: $O(b^d)$ (size of frontier at depth d).

14 Board 13

14.1 Depth-First Search

- Greedy algorithm
- Explores further depth each time
- LIFO Queue - Push: adds to front
  - Pop: removes from front

15 Board 13

iter	p	s	frontier by p_{last}	explored
0	-	-	❂ [A]	❂ {}
1	A	A	❂ [B, E, D]	❂ {A}
2	A:B	B	❂ [E, E, D]	❂ {A, B}
3	A:B:E	E	❂ [C, E, D]	❂ {A, B, E}
4	A:B:E:C	C	-	-

16 Board 14

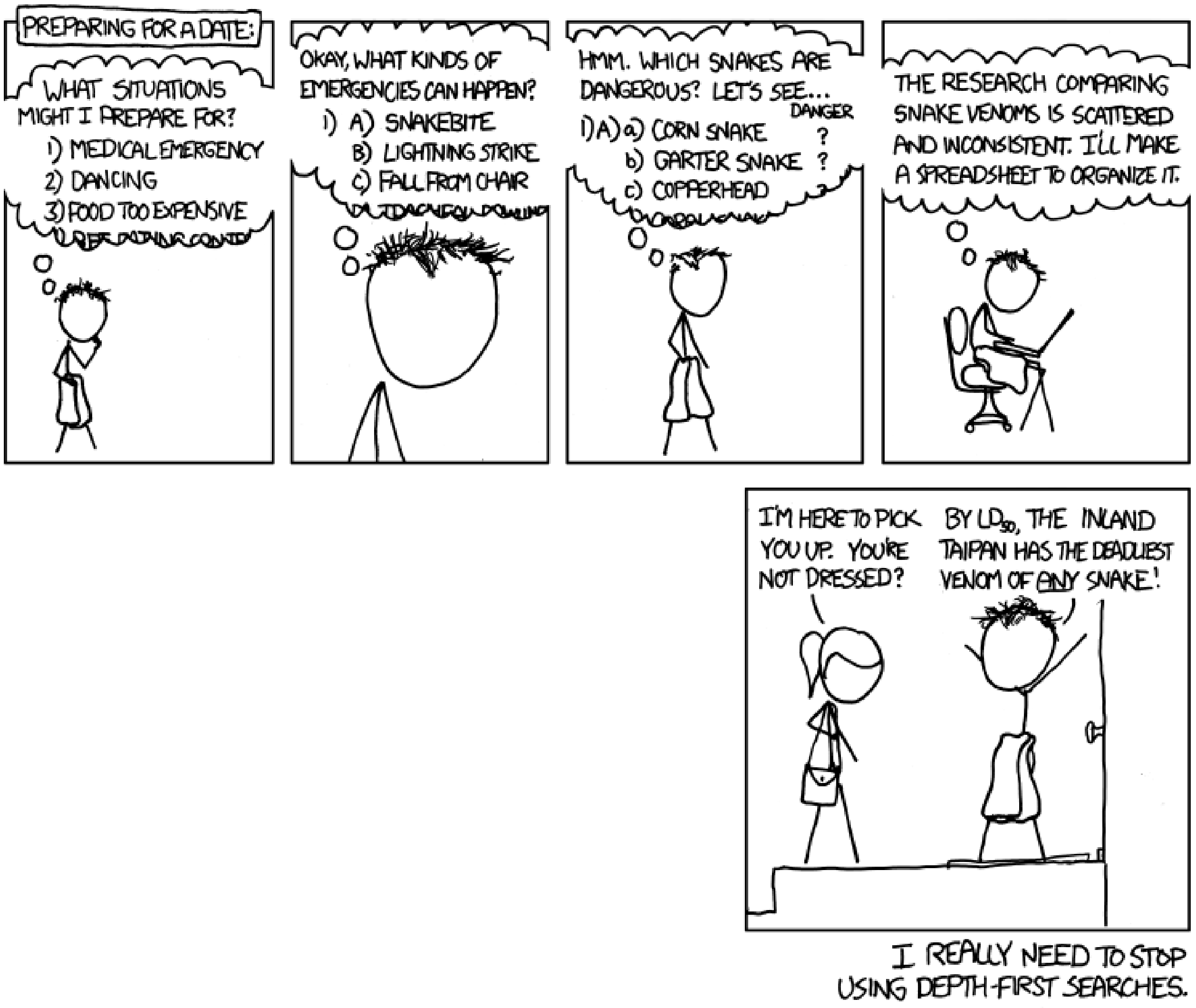
- Completeness: No (graph search). yes (finite, tree-search)
- Optimality: No. (when is it optimal?)
- Time-Complexity: $O(b^m)$
- Space-Complexity: $O(b^m)$ for graph search, but only $O(bm)$ for tree search!

DFS has pretty poor search properties. If there is an infinite path it can go down it forever. Also it can find paths that are much deeper than m since it doesn't explore in depth order. And we have seen in the example that it is non-optimal.

In terms of the properties of the algorithm, this means:

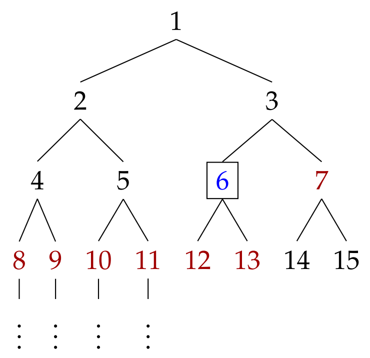
17 Board 15

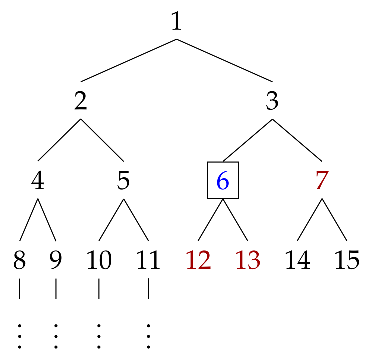
XKCD puts these issues into perspective.


PREPARING FOR A DATE:
WHAT SITUATIONS MIGHT I PREPARE FOR? 1) MEDICAL EMERGENCY 2) DANCING 3) FOOD TOO EXPENSIVE
OKAY, WHAT KINDS OF EMERGENCIES CAN HAPPEN? 1) A) SNAKEBITE B) LIGHTNING STRIKE C) FALL FROM CHAIR
HMM. WHICH SNAKES ARE DANGEROUS? LET'S SEE... 1)A) a) CORN SNAKE b) GARTER SNAKE c) COPPERHEAD DANGER ? ? ?
THE RESEARCH COMPARING SNAKE VENOMS IS SCATTERED AND INCONSISTENT. I'LL MAKE A SPREADSHEET TO ORGANIZE IT.
I'M HERE TO PICK YOU UP. YOU'RE NOT DRESSED?
BY LD50, THE INLAND TAIPAN HAS THE DEADLIEST VENOM OF ANY SNAKE!
I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

18 Board 16

Despite all these issues, we will very often elect to use DFS! The main advantage is its very good space complexity for tree search versus BFS. Consider the following example with $b = 2$. The red nodes indicate the current frontier after expanding the blue node (6). For BFS (left), every node at this depth is being added to the frontier. For DFS (right), we at most have b nodes on the frontier for each depth so far. This means that DFS with Tree-Search requires far less memory than BFS, making it more practical in many settings.





19 Board 17

Variants of depth-first search Fixing DFS completeness

- Idea be greedy
- Maintain worst case complexity.
- drop things from queue

20 Board 18

By doing this we limit the space of paths it can possibly explore to $O(b^l)$ and thus the complexity is improved. However, if $l < d$ the algorithm is still not complete.

21 18

- Completeness: No.
- Optimality: No.
- Time-Complexity: $O(b^l)$
- Space-Complexity: $O(b^l)$ (for graph-search) but only $O(bl)$ for tree-search.

Note that we will implement a variant depth-limited search in the next section on game playing.

22 Board 18

Iterative Deepening Search However, we can take this approach one step further. In iterative deepening search we repeatedly run DLS with $l = 1, \ldots$. By doing so, l will eventually reach d and we will find a solution.

- Completeness: Yes.
- Optimality: No. (Like BFS, optimal when all costs are 1)
- Time-Complexity: $O(b^d)$
- Space-Complexity: $O(b^d)$

23 Board 19

Question 1 *Why is the time-complexity for IDS so low if we have to run it some many times?*

❂ We may have to run the algorithm at most d times to find a solution. In the worst case this gives $O(b^1 + \ldots + b^d)$ time. However, in big O notation that becomes just $O(b^d)$ since the last run dominates.

24 Board 20

Uniform-cost search

Fixing BFS optimality

- Idea: be conservative
- Maintain optimality.
- Priority queue

Priority function:

$$f : \mathcal{P} \mapsto \mathbb{R}^+$$

The failure case for the optimality of BFS is when there is a deep node that has a lower cost than a shallower node. BFS fails to be optimal in this case, because it order paths by depth. Uniform-cost Search (UCS) fixes this issue by ordering paths by cost instead.

To do this we use a **priority queue** where 'pop' gives the next path based on a priority function. Define the priority function as: $f : \mathcal{P} \mapsto \mathbb{R}$ to select the next path. For UCS, we set this function to be $f(p) \triangleq g(p)$, i.e. the cost of the partial path.

25 Board 21

iter	p	s	frontier (p)	explored
0	-	-	❂ [A:0]	❂ {}
1	A	A	❂ [A:B:2, A:D:3, A:E:5]	❂ {A}
2	A:B	B	❂ [A:D:3, A:E:5]	❂ {A, B}
3	A:D	D	❂ [A:E:5, A:D:C:7]	❂ {A, B, D}
4	A:E	E	❂ [A:D:C:8]	❂ {A, B, D, E}
5	A:D:C	C	-	-

Because UCS expands paths in cost order and costs are non-negative, each path it expands must be at least as costly as all previous paths. This means that if UCS finds a goal node it must be optimal. With a few further assumptions (see AIMA) we can also show it is complete. AIMA also includes a description of the time- and space-complexity that uses the optimal score and smallest path cost ϵ.

26 Board 22

- Completeness: Yes.
- Optimality: Yes.
- Time-Complexity: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
- Space-Complexity: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$