

Blind Search¹

Our first series of lectures is concerned with the following topics in the core area of search:

1. blind, or exhaustive, search: depth-first, breadth-first, iterative deepening, iterative broadening, and bidirectional search
2. heuristic search, in which the search is guided by an heuristic evaluation function: greedy search, beam search, A* and IDA* search
3. adversarial search for game playing: minimax algorithm, $\alpha\beta$ pruning

The following are examples of search problems:

- *theorem proving*: given a set of axioms, some rules of inference, and a select formula, find a proof of the formula, if one exists
- *game playing*: given the rules of game (e.g., chess), find an optimal move
- *class scheduling*: given degree requirements, course offerings and times, and student preferences, find a (n optimal) semester schedule

1 Basic Search Problem

A *basic search problem* is a 4-tuple $\langle X, S, G, \delta \rangle$, where

- X is a set of states
- $S \subseteq X$ is a nonempty set of *start* states
- $G \subseteq X$ is a nonempty set of *goal* states
- $\delta : X \rightarrow 2^X$ is a state transition function
 $\delta(x)$ is the set of successor states of x

Example As an example, consider the sliding tiles puzzle, where the objective is to slide numbered tiles into numerical order, given an $n \times m$ board with $n \times m - 1$ tiles and one blank space. The states X describe the locations of

¹Copyright© Amy Greenwald, 2001–05

the numbers and the blank space. The set of start states $S = X$. The set of goal states $G \subseteq X$ includes only that state in which the tiles are in row-major order: i.e, 1 to n in the first row, $n + 1$ to $2n$ in the second row, \dots , and finally, $n(m - 1) + 1$ to $nm - 1$ in the last row, with the blank square in the bottom-right-most corner. The transition function describes the change in state that results from moving the blank space left, right, up, or down. (See Figure 1).

1	3	5
7	2	4
6	8	

(A) Start State

1	2	3
4	5	6
7	8	

(B) Goal State

X	↓	X
→		←
X	↑	X

→		←
X	↑	X
X	X	X

(C) Examples of Legal Moves

Figure 1: Sliding Tiles Puzzle ($n = m = 3$). (A) Start State. (B) Goal State. (C) Examples of Legal moves: An arrow indicates the direction in which a tile can be moved. An “X” means that a tile cannot be moved.

Exercise Formalize the following version of the *All the King’s Digits* puzzle as a basic search problem: Given the sequence of digits 0123456789, insert the symbols $(,), +, -, \cdot, /$ between the digits such that the resulting expression evaluates to 100.

One possible solution to this puzzle is:

$$0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + (8 \cdot 9) = 100$$

Can you find others?

2 Evaluation Criteria

The following criteria are used to evaluate search algorithms:

- time complexity: how much time is required to find solution?
- space complexity: how much space is required to find solution?
- completeness: is the algorithm guaranteed to find solution, if one exists?
- optimality: does it find an optimal solution, if multiple solutions exists?

The analyses and algorithms presented in this lecture depend on the assumption that the search space is a tree of (possibly infinite) depth d with finite branching factor b . The depth of a tree is defined as the maximum number of hops from the root node to any other node in the tree. The branching factor of a tree is defined as the maximum number of children of any node in the tree.

3 Generic Search Algorithm

Search spaces are formulated as trees or graphs. The *fringe* (or *frontier*) is the collection of nodes waiting to be expanded, usually stored as a stack, a queue, or a priority queue. Nodes on the fringe are called *open*. After nodes are deleted from the fringe, they are labeled *closed*. The following generic search algorithm is designed for trees, not graphs—it does not maintain a list of closed nodes.

SEARCH(X, S, G, δ)	
Inputs	search problem
Output	(path to) goal node
Initialize	$O = S$ is the list of open nodes
while (O is not empty) do	
1. delete some node $n \in O$	
2. if $n \in G$, return (path to) n	
3. append $\delta(n)$ to O	
fail	

Table 1: Generic Search Algorithm.

4 Breadth-First Search

The main idea of breadth-first search (BFS) is to expand all nodes at depth i before expanding those at depth $i + 1$. BFS is implemented by storing the list of open nodes as a *queue*, and accessing its entries in a first-in-first-out fashion.

BFS(X, S, G, δ)	
Inputs	search problem
Output	(path to) goal node
Initialize	$O = S$ is the list of open nodes
while (O is not empty) do	
1. delete <i>first</i> node $n \in O$	
2. if $n \in G$, return (path to) n	
3. append $\delta(n)$ to <i>back</i> of O	
fail	

Table 2: Breadth-First Search.

BFS is complete: it is guaranteed to find a solution, if one exists. Moreover, BFS is optimal: it always finds a goal node of minimal distance from the start.

In the worst case, BFS expands every node, which takes time as follows:

$$1 + b + b^2 + \dots + b^d = \sum_{i=0}^d b^i = \frac{b^{d+1} - 1}{b - 1} = O(b^d)$$

In terms of space, BFS maintains a list of all nodes at all depths: at depth d the length of this list is b^d . The space complexity of BFS is exponential in d .

5 Depth-First Search

The main idea of depth-first search (DFS) is to expand one of the deepest nodes visited. DFS is implemented by storing the list of open nodes as a *stack*, and accessing its entries in a last-in-first-out fashion.

DFS(X, S, G, δ)	
Inputs	search problem
Output	(path to) goal node
Initialize	$O = S$ is the list of open nodes
while (O is not empty) do	
1. delete <i>first</i> node $n \in O$	
2. if $n \in G$, return (path to) n	
3. prepend $\delta(n)$ to <i>front</i> of O	
fail	

Table 3: Depth-First Search.

Like BFS, the time complexity of DFS is $O(b^d)$. It is exponential in d because in the worst-case DFS expands every node. The space complexity of DFS, however, is linear in d , where d is the length of longest path. If b is the width of the widest expansion, at most b nodes are stored at each of the d depths: DFS is $O(bd)$.

DFS is neither complete nor optimal. Given knowledge base $\{\phi \rightarrow \phi, \phi\}$ and formula ϕ , DFS could fail to prove ϕ by forever expanding $\phi \rightarrow \phi$. Using DFS, a robot that intends to visit its neighbor to the west, but starts out searching to the east, travels all the way around the world before finding its neighbor!

6 Iterative Deepening

Iterative deepening (ID) is an optimal search algorithm with the space requirements of DFS—it requires memory linear in d —and the performance properties of BFS—it is complete and optimal. The main idea of iterative deepening is to repeatedly search in depth-first fashion, over subgraphs of depth 0, depth 1, depth 2, and so on, until a goal is found.

ID(X, S, G, δ)	
Inputs	search problem
Output	(path to) goal node
Initialize	$c = 0$ is the cut-off depth $O = S$ is the list of open nodes
<pre> while (1) do 1. while (O is not empty) do (a) delete <i>first</i> node $n \in O$ (b) if $n \in G$, return (path to) goal n (c) if $\text{depth}(n) \leq c$ i. prepend $\delta(n)$ to <i>front</i> of O 2. increment c, $O = S$ </pre>	

Table 4: Iterative Deepening.

ID is optimal and complete: it is guaranteed to find a goal if one exists—specifically, an optimal goal if multiple solutions exist. It iteratively performs depth-first searches; thus, its space complexity is that of DFS, namely $O(bd)$.

Like DFS and BFS, the time complexity of ID is $O(b^d)$: in the worst-case it is exponential in d . ID expands nodes at depth 0 $d + 1$ times, at depth 1 d times, \dots , and at depth d 1 time. Thus, the total time required is given by:

$$1 + \underbrace{1 + b}_{d=1} + \underbrace{1 + b + b^2}_{\text{depth } 2} + \dots + \underbrace{1 + b + b^2 + \dots + b^d}_{\text{depth } d} = \sum_{c=0}^d \sum_{i=0}^c b^i = \sum_{c=0}^d \frac{b^{c+1} - 1}{b - 1} = O(b^d)$$

7 Iterative Broadening

The idea of iterative broadening (IB) is analogous to that of iterative deepening, but IB iteratively broadens its search, whereas ID iteratively deepens it. IB performs depth-first searches on subgraphs of breadth 1, breadth 2, breadth 3, and so on, until a goal node is reached.

IB(X, S, G, δ)	
Inputs	search problem
Output	(path to) goal node
Initialize	$c = 1$ is the cut-off breadth $O = S$ is the list of open nodes
while (1) do	
1. while (O is not empty) do	
(a) delete <i>first</i> node $n \in O$	
(b) if $n \in G$, return (path to) goal n	
(c) if $\text{breadth}(n) \leq c$	
i. prepend c nodes in $\delta(n)$ to <i>front</i> of O	
2. increment c , $O = S$	

Table 5: Iterative Broadening.

IB is not complete: it is susceptible to following infinite paths within its breadth limit as it conducts depth-first search. Neither is IB optimal.

In the worst-case, its space complexity equals that of DFS, and the time it requires is $O(b^d)$. Specifically, the number of nodes IB expands is given by:

$$d + \underbrace{1 + 2 + \dots + 2^d}_{\text{breadth } 2} + \dots + \underbrace{1 + b + \dots + b^d}_{\text{breadth } b} = \sum_{i=0}^d \sum_{c=1}^b c^i = O((b+1)^{d+1})$$

since

$$\frac{b^{i+1}}{i+1} = \int_0^b x^i dx \leq \sum_{c=1}^b c^i \leq \int_1^{b+1} x^i dx = \frac{(b+1)^{i+1} - 1}{i+1}$$

8 Bidirectional Search

A *bidirectional search problem* is a 5-tuple $\langle X, S, G, \delta, \gamma \rangle$, where $\langle X, S, G, \delta \rangle$ is a basic search problem and $\gamma : X \rightarrow 2^X$ is a transition function ($\gamma(x)$ is the set of predecessors of x). Note that not all basic search problems are easily posed as bidirectional search problems: e.g., it is nontrivial to generate the predecessors in the game of chess from the set of goal states (checkmate configurations).

In bidirectional search, as the name suggests, search proceeds simultaneously in two directions: both forwards from an initial state and backwards from a goal state. It terminates where and when the two searches meet.

Bidirectional search is typically implemented as two breadth-first searches. Like BFS, bidirectional BFS is optimal and complete. Bidirectional search improves upon BFS in terms of complexity, however. The time complexity of bidirectional BFS is exponential in $d/2$. In the worst case, each direction of bidirectional BFS expands every node through depth $d/2$ —this expansion requires time equal to $1 + b + b^2 + \dots + b^{d/2} = O(b^{d/2})$ —which for two directions requires $O(2b^{d/2}) = O(b^{d/2})$. Its space complexity is also $O(b^{d/2})$, since it maintains list of all nodes all at depths i , and the number of nodes at depth $d/2 = b^{d/2}$.

9 Summary

Criteria	DFS	BFS	ID	IB	biBFS
Time	$O(b^d)$	$O(b^d)$	$O(b^d)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(bd)$	$O(b^d)$	$O(bd)$	$O(bd)$	$O(b^{d/2})$
Completeness	NO	YES	YES	NO	YES
Optimality	NO	YES	YES	NO	YES

1. BFS is preferred if the branching factor is small
2. IB is preferred if the branching factor is large: the search space is wide (possibly infinite), particularly if goals are known to be deep
3. ID is preferred if the depth is large: the search space is deep (possibly infinite), particularly if goals are known to be shallow
4. DFS is preferred if the maximum depth of the goal nodes is known: if this depth is n , modify DFS to search only to depth n

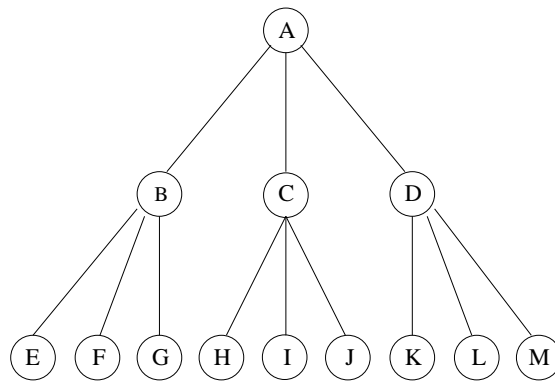


Figure 2: (a) BFS: ABCDEFGHIJKLM. (b) DFS: ABEFGCHIJDKLM. (c) ID: AABCDABEFGCHIJDKLM. (d) IB: ABEABEFCHIABEFGCHIJDKLM.

Problems

#1 Three cats and three mice are on one side of a river, along with a row boat that is capable of transporting either 1 or 2 occupants. We seek a way to move all the animals to the opposite side of the river, without ever allowing the number of cats on a river bank to exceed the number of mice—otherwise the cats would eat the mice. (Imagine that cats and mice can control a row boat.)

(a) State this problem as a formal search problem in terms of $\langle Q, S, F, \delta \rangle$. You need not define the transition function δ in full detail; instead list the possible operators which induce transitions: e.g., 1 cat crosses the river in the boat, 2 cats cross the river in the boat, etc.

(b) Solve the cat-and-mouse problem by drawing the search space, and finding an optimal path to a goal. Limit the size of the search by disallowing moves that revisit past states: e.g., initially, 1 cat can cross the river, but his return is not allowed, since the initial state would be revisited.

#2 A farmer, a fox, a chicken, and a bag of grain stand on one side of a river. A boat can carry the farmer and one additional animal or item across the river. The fox cannot be left unattended with the chicken, however, and the chicken cannot be left unattended with the grain; in either case, the former eats the latter. Find an optimal means of transporting the lot across the river.

#3 Extend the generic search algorithm in Table 1 from trees to graphs: i.e., insert pseudocode that maintains a list of closed nodes.

#4 In the following maze, boldface lines indicate impassable walls. Search for a path from the start state to the goal state visiting neighbors in the following order: EAST, WEST, NORTH, SOUTH.

M	N	O	<i>goal</i> P
I	J	K	L
E	F	G	H
<i>start</i> A	B	C	D

(a) In what order does DFS visit the cells in this grid? Is DFS optimal?

(b) In what order does BFS visit the cells in this grid? Is BFS optimal?