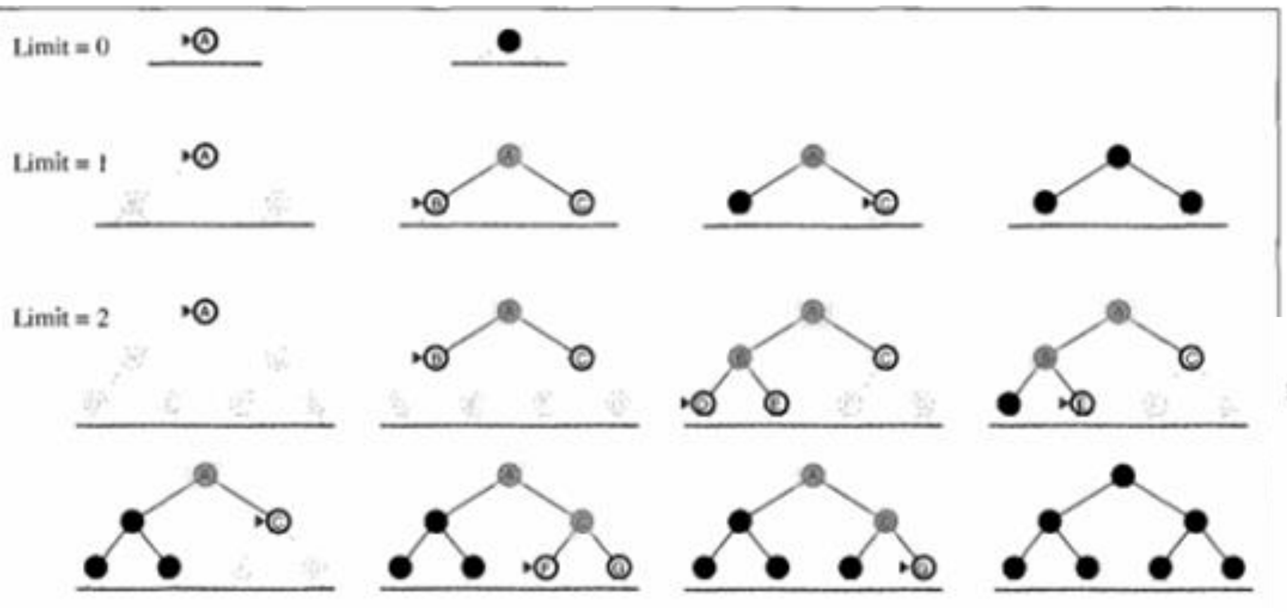
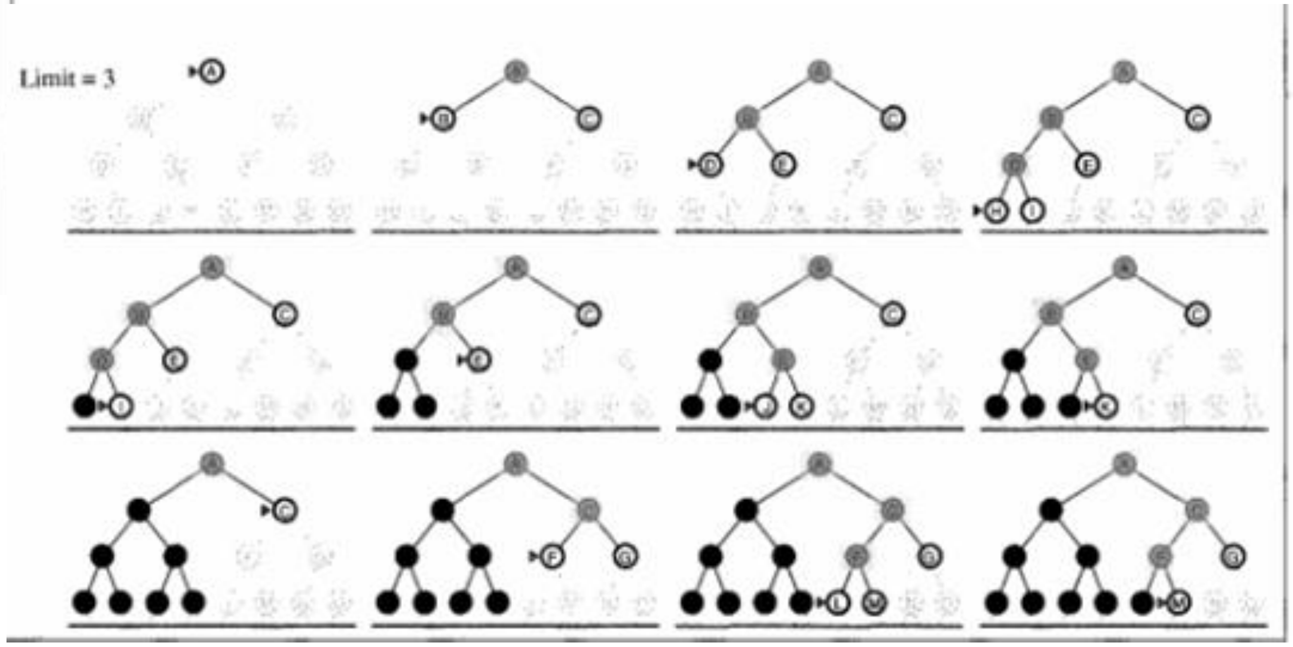
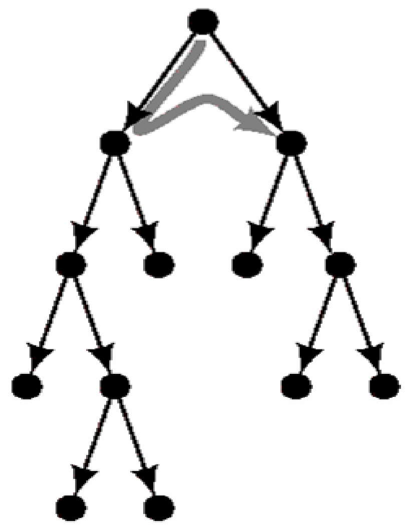
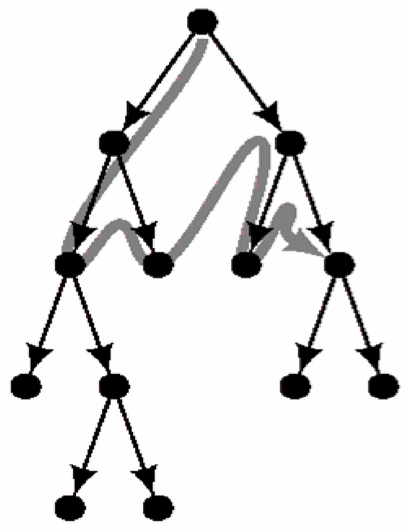
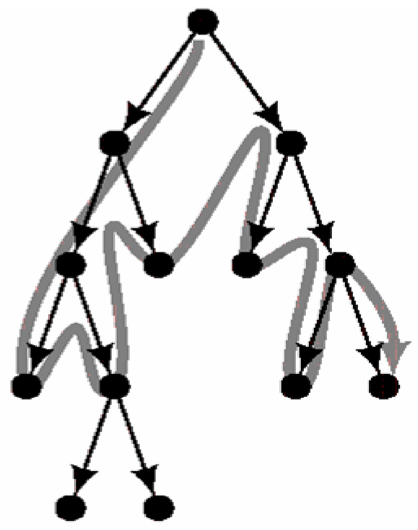
Iterative Deepening Search

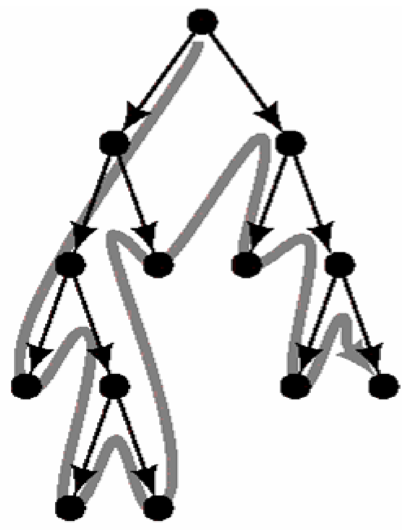

Limit = 0
Limit = 1
Limit = 2


Limit = 3


Depth bound = 1


Depth bound = 2


Depth bound = 3


Depth bound = 4

- combination with depth-first search, that finds the best depth limit.
- -gradually increasing the limit-first 0, then 1, then 2, and so on-until a goal is found (depth limit reaches d, the depth of the shallowest goal node)
- **-**benefits of depth-first + breadth-first search
- Like DFS-memory requirements are very modest:O(bd) to be precise
- Like BFS, it is complete when the branching factor is finite and optimal when the path cost is a nondecreasing function of the depth of the node.

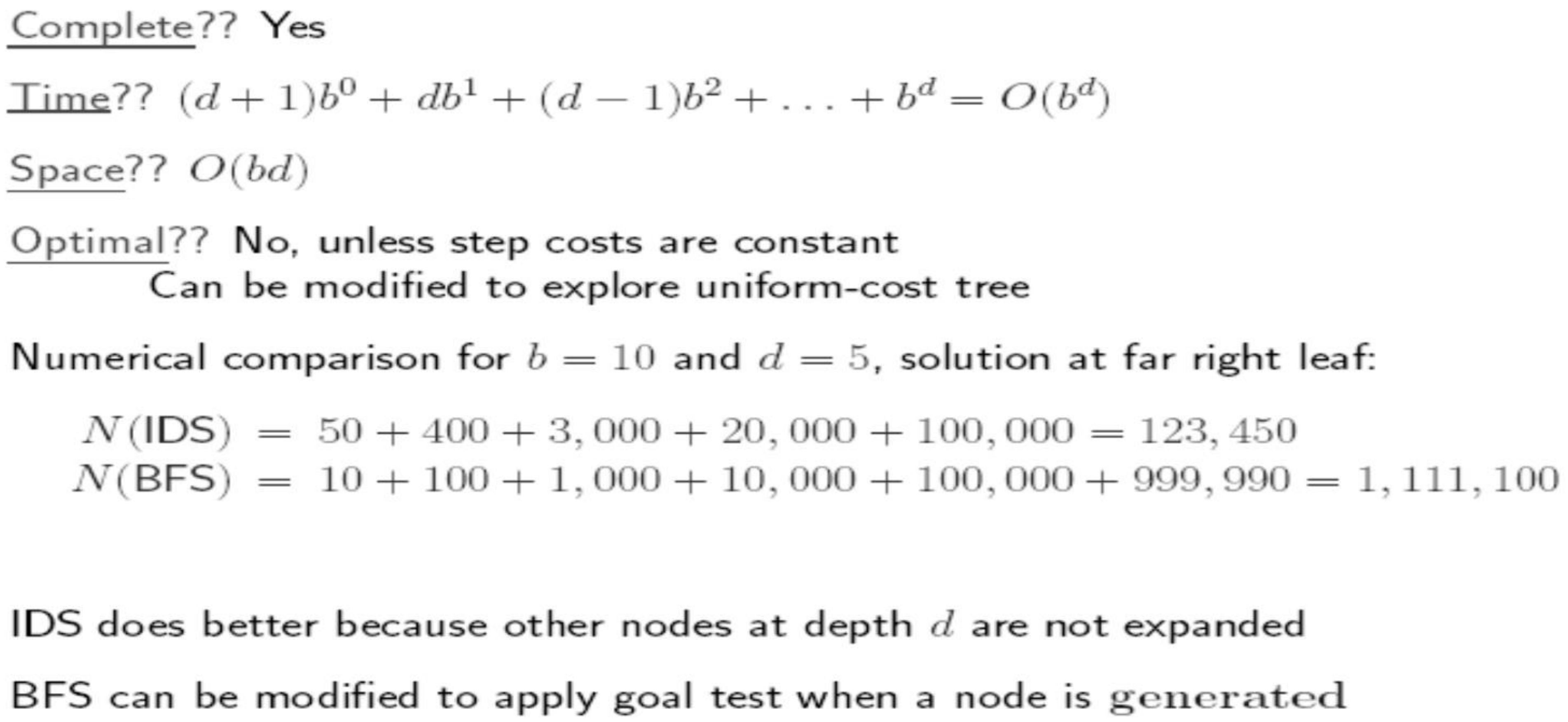
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
   inputs: problem, a problem

   for depth ← 0 to ∞ do
      result ← DEPTH-LIMITED-SEARCH(problem, depth)
      if result ≠ cutoff then return result
```

Figure 3.14 The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns *failure*, meaning that no solution exists.

- Iterative search is not as wasteful as it might seem.
- may seem wasteful, because states are generated multiple times.
- But not without- reason-(with the same (or nearly the same) branching factor at each level, most of the nodes are in the bottom level, so it does not matter much that the upper levels are generated multiple times.)
- (In IDS, the nodes on the bottom level (depth d) are generated once, the next to bottom level are generated twice, and so on, up to the children of the root, which are generated d times.)
- So the total number of nodes generated is

$$N(IDS) = (d)\, b + (d - l)b^2 + . . . + (1)\, b^d$$

Properties of iterative deepening search



Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? No, unless step costs are constant
Can be modified to explore uniform-cost tree

Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$\begin{aligned} N(\text{IDS}) &= 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450 \\ N(\text{BFS}) &= 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100 \end{aligned}$$

IDS does better because other nodes at depth d are not expanded

BFS can be modified to apply goal test when a node is generated

In general,iterative deepening is the prefered uninformed search method when there is a large search space and the depth of solution is not known.

Bidirectional search

- run two simultaneous searches-one forward from the initial state and the other backward from the goal, stopping when the two searches meet in the middle.

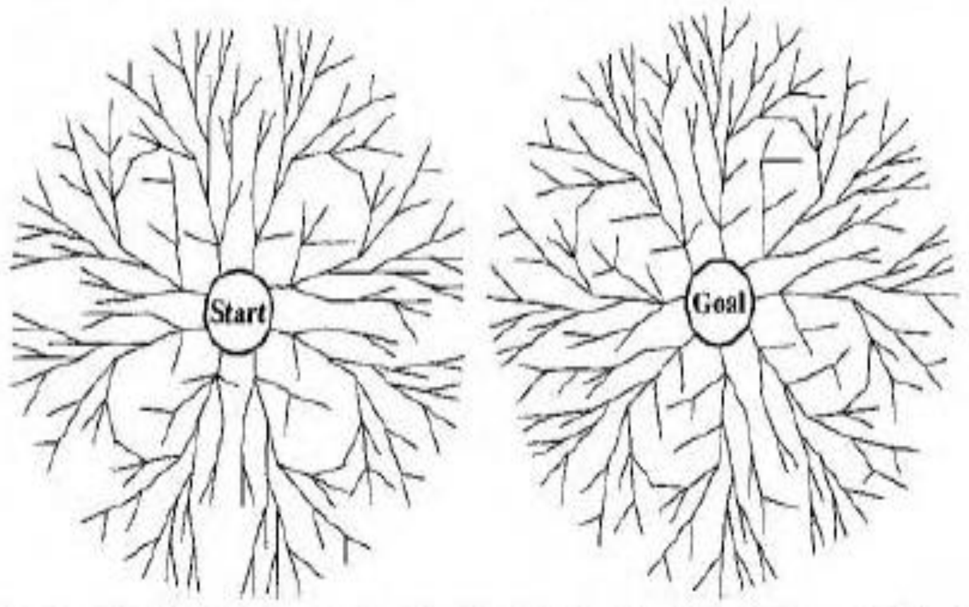
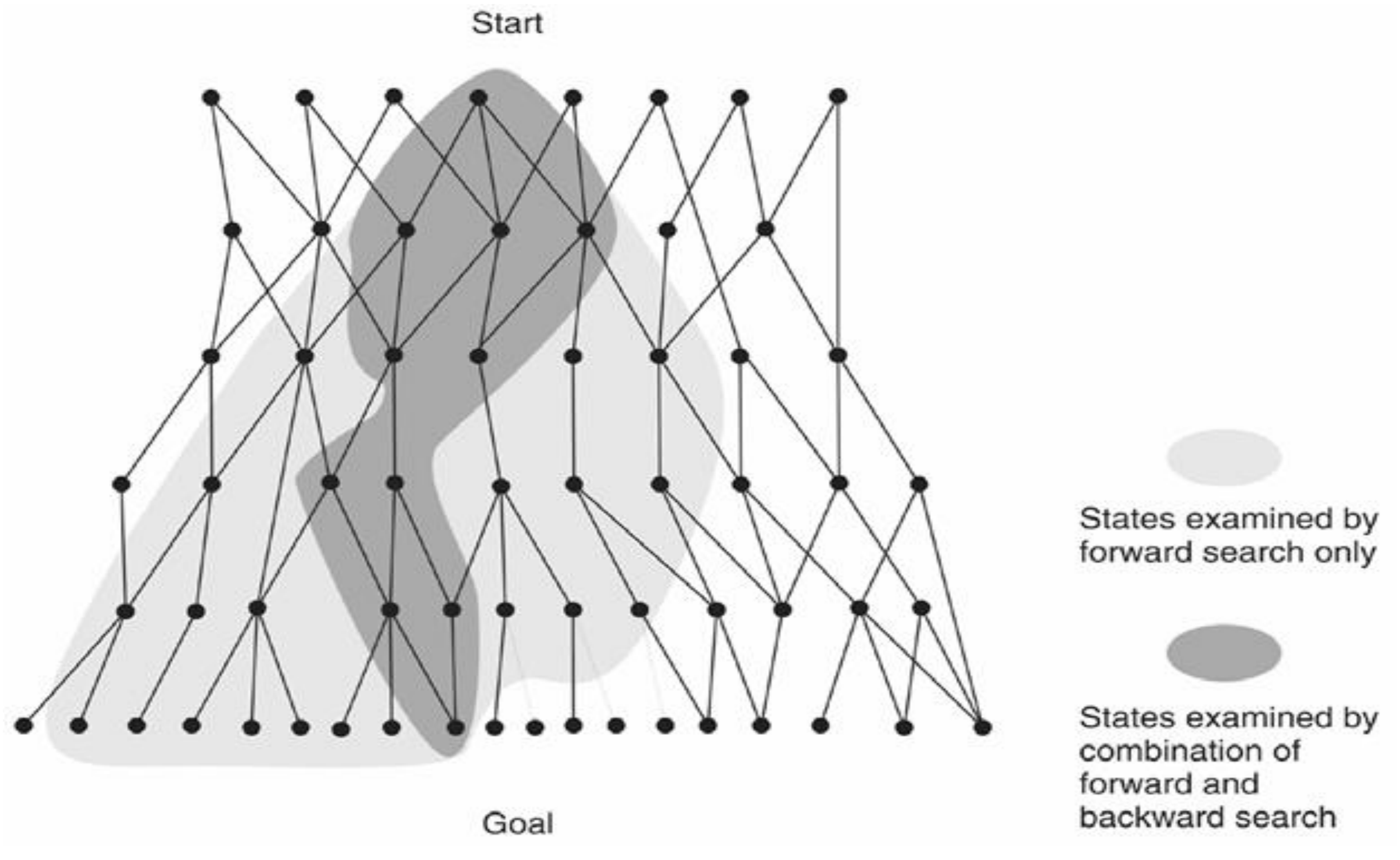


Figure 3.16 A schematic view of a bidirectional search that is about to succeed, when a branch from the start node meets a branch from the goal node.

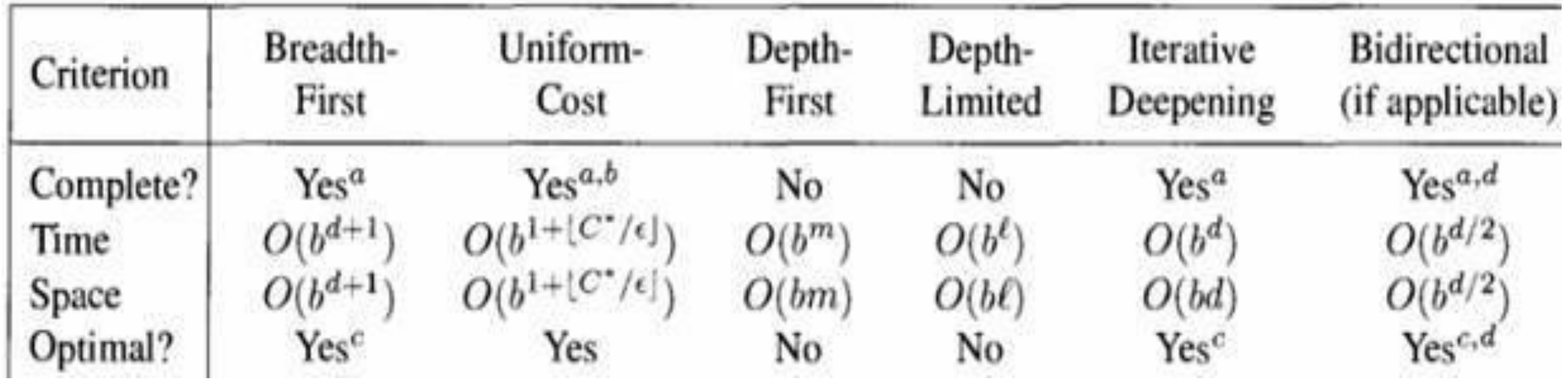
- The motivation is that $b^{d/2} + b^{d/2}$ is much less than b^d, (the area of the two small circles is less than the area of one big circle centered on the start and reaching to the goal.)
- - implemented by having one or both of the searches check each node before it is expanded to see if it is in the fringe of the other search tree; if so, a solution has been found.

- Eg: solution depth d = 6, and each direction runs BFS one node at a time, then in the worst case the two searches meet when each has expanded all but one of the nodes at depth 3.
- For b = 10, this means a total of 22,200 node generations, compared with 1 1,111,100 for a standard breadth-first search.
- Checking a node for membership in the other search tree can be done in constant time with a hash table,
- so the time complexity of bidirectional search is $O(b^{d/2})$.
- At least one of the search trees must be kept in memory so that the membership check can be done, hence the space complexity is also $O(b^{d/2})$. - weakness of bidirectional search.
- The algorithm is complete and optimal (for uniform step costs) if both searches are breadth-first; (other combinations may sacrifice completeness, optimality, or both.)

- Bidirectional search can sometimes lead to finding a solution more quickly. The reason can be seen from inspecting the following figure.


Start
Goal
States examined by forward search only
States examined by combination of forward and backward search

Comparing Uninformed Search Strategies



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	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

- **Figure 1.38** Evaluation of 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 >= E for positive E; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.