UIT2504 Artificial Intelligence

Basic Search Strategies

C. Aravindan
<AravindanC@ssn.edu.in>

Professor of Computer Science
SSN College of Engineering

August 08, 2024

Searching for a solution

- Start with initial state in the working set

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty
  - Choose and remove a state x from the working set

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty
  - Choose and remove a state x from the working set
  - If it is a goal state, return solution

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty
  - Choose and remove a state x from the working set
  - If it is a goal state, return solution
  - Else, expand x and add the successor states $S(x)$ to the working set

Search Strategies

- Uninformed:
  - Breadth-First
  - Depth-First
  - Iterative Deepening
  - Bi-Directional Search
- Informed (Heuristics):
  - Best-first Greedy
  - A^*
  - Local Search Strategies
- Constraint Satisfaction

Performance Measures

- Completeness
- Time Complexity
- Space Complexity
- Optimality

BFS: Complexities

- Is BFS complete? — that is, will it find a solution if one exists? — Yes! — but the branching factor b should be finite!
- Is it optimal? — No, unless optimality is defined by the length of the path
- Time complexity? — $1 + b + b^2 + \cdots + b^d + (b^{d+1} - b)$ — $O(b^{d+1})$
- Space complexity? — same as that of time complexity — $O(b^{d+1})$

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal? — Yes, if the cost function is monotonic
- Time complexity? — similar to that of breadth-first — $O(b^{\lceil C^*/e \rceil})$
- Space complexity? — same as that of time complexity — $O(b^{\lceil C^*/e \rceil})$

UCS: Complexities

- Is UCS complete? — that is, will it find a solution if one exists? — Yes! — but there should not be any negative cost!
- Is it optimal? — Yes, if the cost function is monotonic
- Time complexity? — similar to that of breadth-first — $O(b^{\lceil C^*/e \rceil})$
- Space complexity? — same as that of time complexity — $O(b^{\lceil C^*/e \rceil})$

Revised in third edition

NOTE: In the third edition of the book, the complexity is revised as $O(b^{1+\lfloor C^*/e \rfloor})$

DFS: Complexities

- Is DFS complete? — No! — not in general
- Is it optimal? — No!
- Time complexity? — $O(b^m)$
- Space complexity? — $O(bm)$

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time
- When a "dead-leaf" is encountered, or when search fails for all the successors, the algorithm backtracks to the parent node to generate the next successor

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time
- When a "dead-leaf" is encountered, or when search fails for all the successors, the algorithm backtracks to the parent node to generate the next successor
- It is easy to implement using recursion

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time
- When a "dead-leaf" is encountered, or when search fails for all the successors, the algorithm backtracks to the parent node to generate the next successor
- It is easy to implement using recursion — usage of stack is implicit!

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time
- When a "dead-leaf" is encountered, or when search fails for all the successors, the algorithm backtracks to the parent node to generate the next successor
- It is easy to implement using recursion — usage of stack is implicit!
- Space complexity?

Backtracking

- Backtracking is a variation of depth-first search where only one successor is generated at a time
- When a "dead-leaf" is encountered, or when search fails for all the successors, the algorithm backtracks to the parent node to generate the next successor
- It is easy to implement using recursion — usage of stack is implicit!
- Space complexity? — $O(m)$

Questions?

Can we do better?

- Is it possible to do better?

Can we do better?

- Is it possible to do better?
- May be not wrt time complexity

Can we do better?

- Is it possible to do better?
- May be not wrt time complexity
- Can we design an algorithm that is complete (like BFS) and with linear space complexity (like DFS)?

Can we do better?

- Is it possible to do better?
- May be not wrt time complexity
- Can we design an algorithm that is complete (like BFS) and with linear space complexity (like DFS)?
- Can you suggest some modification to DFS to make it complete?

Depth-Limited Search

- We may impose a depth-limit L

Depth-Limited Search

- We may impose a depth-limit L
- That is, do not expand beyond the depth of L — consider all the nodes at depth L as "dead-leafs"

Depth-Limited Search

- We may impose a depth-limit L
- That is, do not expand beyond the depth of L — consider all the nodes at depth L as "dead-leafs"
- But how do we choose appropriate L for a given problem?

Depth-Limited Search

- We may impose a depth-limit L
- That is, do not expand beyond the depth of L — consider all the nodes at depth L as "dead-leafs"
- But how do we choose appropriate L for a given problem?
- Obviously, we want L to be larger than d (but not too large!) — but d is unknown

Depth-Limited Search

- We may impose a depth-limit L
- That is, do not expand beyond the depth of L — consider all the nodes at depth L as "dead-leafs"
- But how do we choose appropriate L for a given problem?
- Obviously, we want L to be larger than d (but not too large!) — but d is unknown
- For some problems, we may be able to reason out an appropriate value for such as limit

Depth-Limited Search: Complexities

- Is DLS complete?

Depth-Limited Search: Complexities

- Is DLS complete? — No! — not in general — finds a solution only when the limit chosen is sufficiently large

Depth-Limited Search: Complexities

- Is DLS complete? — No! — not in general — finds a solution only when the limit chosen is sufficiently large
- Is it optimal?

Depth-Limited Search: Complexities

- Is DLS complete? — No! — not in general — finds a solution only when the limit chosen is sufficiently large
- Is it optimal? — No! — not in general

Depth-Limited Search: Complexities

- Is DLS complete? — No! — not in general — finds a solution only when the limit chosen is sufficiently large
- Is it optimal? — No! — not in general
- Time complexity?

Depth-Limited Search: Complexities

- Is DLS complete? — No! — not in general — finds a solution only when the limit chosen is sufficiently large
- Is it optimal? — No! — not in general
- Time complexity? — $O(b^L)$

Depth-Limited Search: Complexities

- Is DLS complete? — No! — not in general — finds a solution only when the limit chosen is sufficiently large
- Is it optimal? — No! — not in general
- Time complexity? — $O(b^L)$
- Space complexity?

Depth-Limited Search: Complexities

- Is DLS complete? — No! — not in general — finds a solution only when the limit chosen is sufficiently large
- Is it optimal? — No! — not in general
- Time complexity? — $O(b^L)$
- Space complexity? — $O(bL)$

Iterative Deepening

- Can we dynamically find the suitable depth-limit!?

Iterative Deepening

- Can we dynamically find the suitable depth-limit!?
- Perform several depth-limited searches with different limit each time

Iterative Deepening

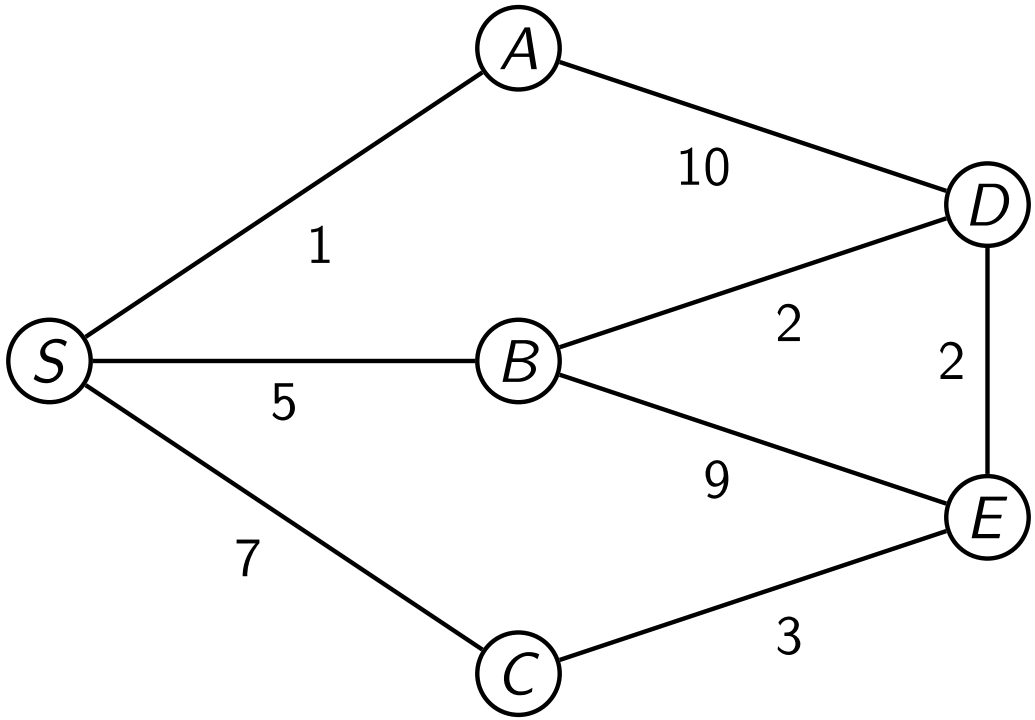
- Can we dynamically find the suitable depth-limit!?
- Perform several depth-limited searches with different limit each time
- For example, start with depth-limit of 0, and increment the same in each iteration

Iterative Deepening

- Can we dynamically find the suitable depth-limit!?
- Perform several depth-limited searches with different limit each time
- For example, start with depth-limit of 0, and increment the same in each iteration
- That means, in the first iteration, perform depth-limit search with a limit of 0, in the next iteration limit is increased to 1, and so on, until a solution is found

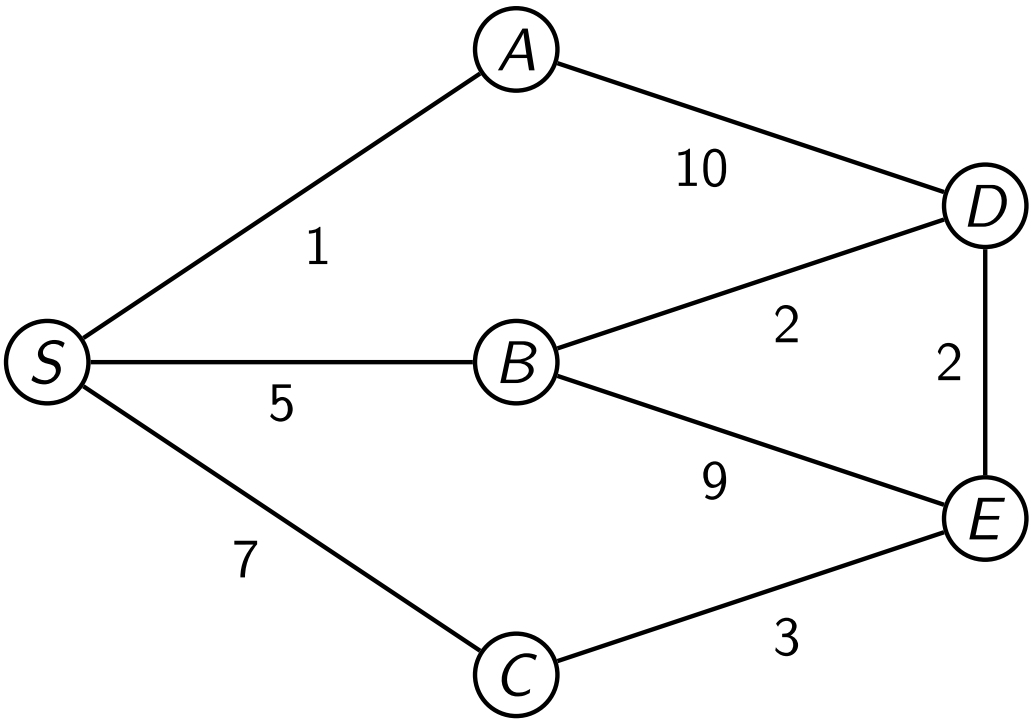
Iterative Deepening: Illustration

- How does it work on our toy example?



Find a route from S to E

Iterative Deepening: Illustration



Iterative Deepening: Complexities

- Is it complete?

Iterative Deepening: Complexities

- Is it complete? — Yes — similar to BFS

Iterative Deepening: Complexities

- Is it complete? — Yes — similar to BFS
- Is it optimal?

Iterative Deepening: Complexities

- Is it complete? — Yes — similar to BFS
- Is it optimal? — No, in general — but similar to BFS

Iterative Deepening: Complexities

- Is it complete? — Yes — similar to BFS
- Is it optimal? — No, in general — but similar to BFS
- Time complexity? $(d)b + (d-1)b^2 + (d-2)b^3 + \cdots + (1)b^d$

Iterative Deepening: Complexities

- Is it complete? — Yes — similar to BFS
- Is it optimal? — No, in general — but similar to BFS
- Time complexity? $(d)b + (d-1)b^2 + (d-2)b^3 + \cdots + (1)b^d$ — $O(b^d)$ — similar to BFS and DFS

Iterative Deepening: Complexities

- Is it complete? — Yes — similar to BFS
- Is it optimal? — No, in general — but similar to BFS
- Time complexity? $(d)b + (d-1)b^2 + (d-2)b^3 + \cdots + (1)b^d$ — $O(b^d)$ — similar to BFS and DFS
- Space complexity?

Iterative Deepening: Complexities

- Is it complete? — Yes — similar to BFS
- Is it optimal? — No, in general — but similar to BFS
- Time complexity? $(d)b + (d-1)b^2 + (d-2)b^3 + \cdots + (1)b^d$ — $O(b^d)$ — similar to BFS and DFS
- Space complexity? — $O(bd)$ — similar to DFS

Iterative Deepening: Complexities

- Is it complete? — Yes — similar to BFS
- Is it optimal? — No, in general — but similar to BFS
- Time complexity? $(d)b + (d-1)b^2 + (d-2)b^3 + \cdots + (1)b^d$ — $O(b^d)$ — similar to BFS and DFS
- Space complexity? — $O(bd)$ — similar to DFS

Iterative Lengthening Search

A similar idea can be used with Uniform Cost Search as well — we may iteratively increase the path cost limits instead of depth limits. The resulting algorithm is referred to as Iterative Lengthening Search

Questions?

Bidirectional Search

- Is there any way to reduce the time complexity?

Bidirectional Search

- Is there any way to reduce the time complexity?
- Try to run two simultaneous searches — one forward from the initial state and the other backward from the goal — check if the two searches "meet" in the middle

Bidirectional Search

- Is there any way to reduce the time complexity?
- Try to run two simultaneous searches — one forward from the initial state and the other backward from the goal — check if the two searches "meet" in the middle
- If done carefully, each search may run only to a depth of $d/2$ — reducing the time complexity to $O(b^{d/2})$

Bidirectional Search

- Goal test is replaced by an "intersection test" — a node being examined in one search is compared with all the nodes on the frontier of the other test

Bidirectional Search

- Goal test is replaced by an "intersection test" — a node being examined in one search is compared with all the nodes on the frontier of the other test
- Effective hashing techniques may be used for this check

Bidirectional Search

- Goal test is replaced by an "intersection test" — a node being examined in one search is compared with all the nodes on the frontier of the other test
- Effective hashing techniques may be used for this check
- But this requires the entire frontiers to be kept in memory

Bidirectional Search

- Goal test is replaced by an "intersection test" — a node being examined in one search is compared with all the nodes on the frontier of the other test
- Effective hashing techniques may be used for this check
- But this requires the entire frontiers to be kept in memory — space complexity $O(b^{d/2})$

Bidirectional Search

- How do we search “backwards”?

Bidirectional Search

- How do we search “backwards”? — need to define predecessors of a state

Bidirectional Search

- How do we search "backwards"? — need to define predecessors of a state — may be easy if the actions are reversible

Bidirectional Search

- How do we search "backwards"? — need to define predecessors of a state — may be easy if the actions are reversible
- What if there are many goal states?

Bidirectional Search

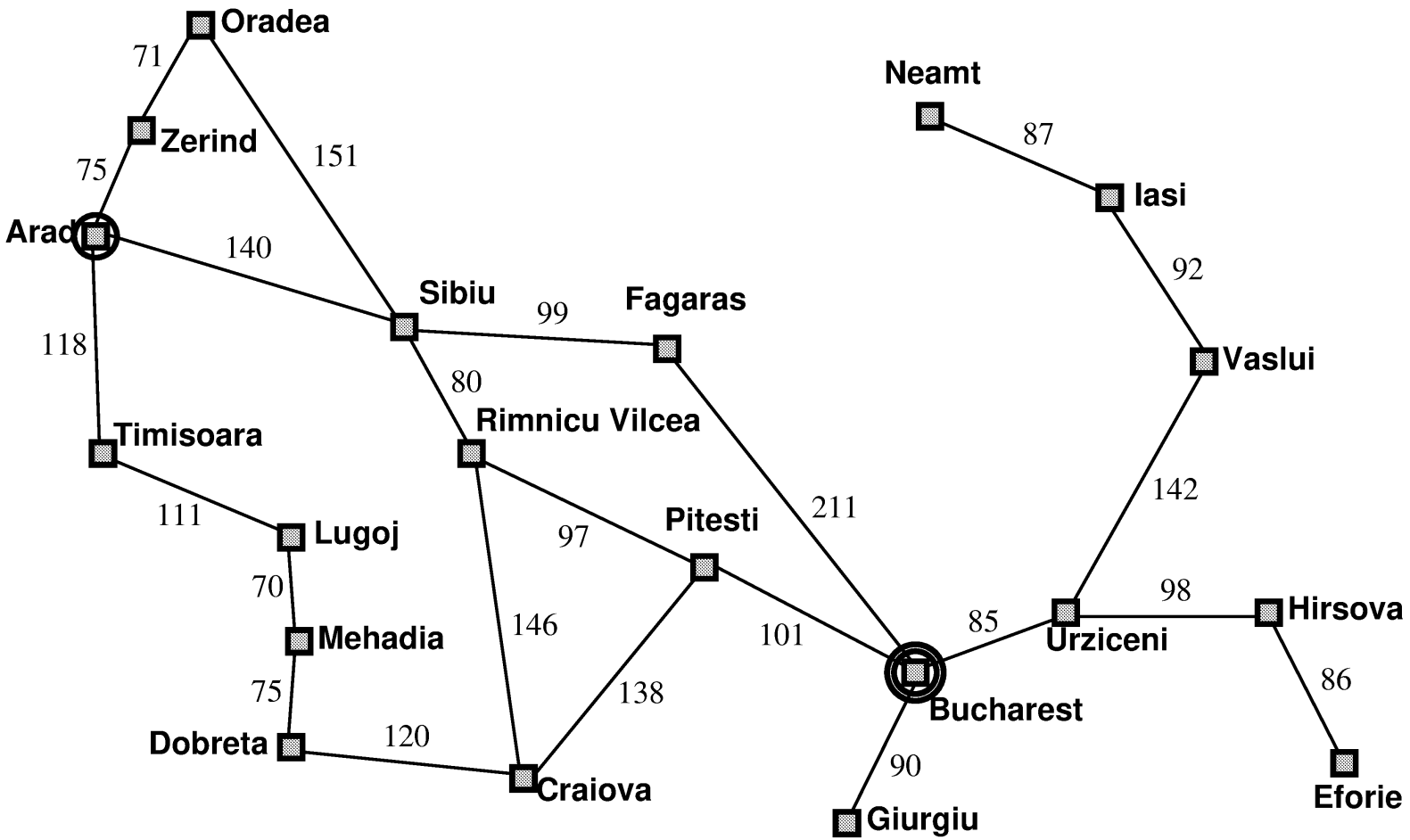
- How do we search "backwards"? — need to define predecessors of a state — may be easy if the actions are reversible
- What if there are many goal states? — we may define a new goal state and make all the original goal states as predecessors of the new goal state

Bidirectional Search

- How do we search "backwards"? — need to define predecessors of a state — may be easy if the actions are reversible
- What if there are many goal states? — we may define a new goal state and make all the original goal states as predecessors of the new goal state
- What if the definition of goal state is abstract? — for example, as in the case of the n-queens problem?

Exercise

- Practice all the basic search strategies to find a route from Arad to Bucharest in the following state graph



Questions?

- Read Chapter 3 of the text book!