UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Artificial Intelligence

Uniformed Search

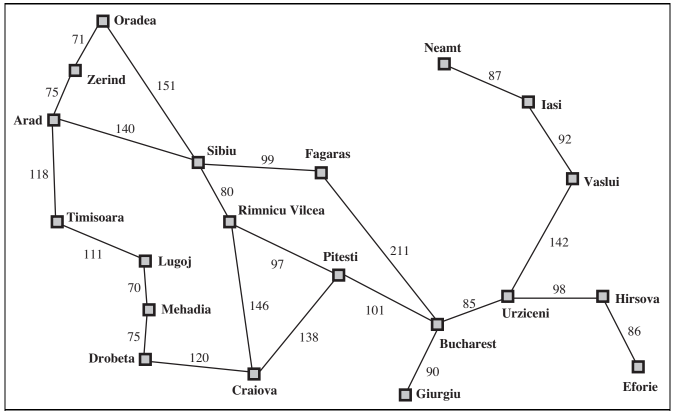
LESSON 4

prof. Antonino Staiano

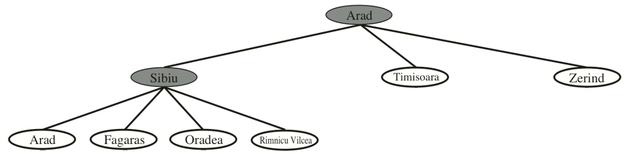
M.Sc. In ''Machine Learning e Big Data'' - University Parthenope of Naples

Search Strategies

- Search algorithms differ only in the criterion to choose one of the partial solutions to follow up at each step



current search tree:



which of the six partial solutions should one choose?

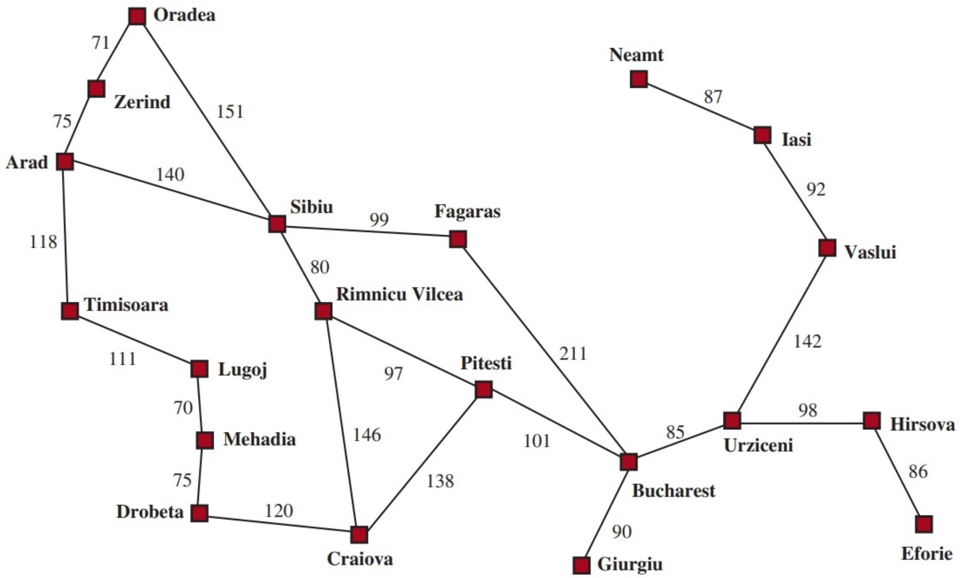
- Two kinds of strategies exist, depending on the available information about which choice is better than another
  - No information: Uninformed search strategies must be used
  - Some information: Informed search strategies can be used

Uninformed Search Strategies

- Rationale
  - In absence of any information about the best partial solution, systematically explore the state space
- Main strategies
  - Breadth-first
  - Depth-first
  - Uniform-cost
  - Depth-limited
  - Iterative-deepening depth first

Avoiding repeated states

- Search algorithms may waste time by expanding different nodes associated with the same state
  - Action are reversible, allowing loops
    - Arad -> Zerind -> Arad-> Zerind …
  - Different paths can lead to the same state, e.g.,:
    - Arad -> Sibiu, and Arad -> Zerind -> Oradea -> Sibiu
  - Cyclical paths exist
    - Arad -> Zerind -> Oradea -> Sibiu -> Arad

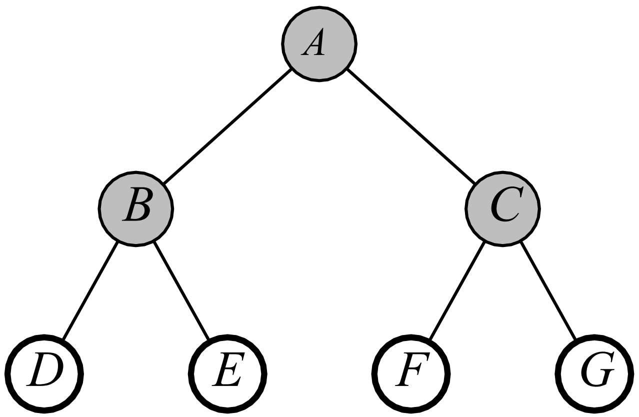


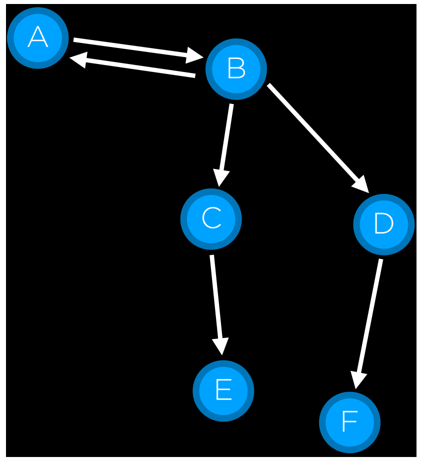
Avoiding repeated states

- Three approaches possible
  - Remember all previously reached states
    - Allows us to detect redundant paths
    - Appropriate for state spaces with many redundant paths
    - It is the choice when the list of the reached states fits in memory
  - Don't worry about the past
    - For some problems, where it is impossible for two paths to reach the same state
  - Compromise ancd check for cycles but not for redundant path in general
- Observation
  - A cycle generates a repeated state (loopy path) and is a special case of a redundant path

Tree-like and Graph Search

- When looking for a path toward a goal state the search is
  - A tree-like search, if we don't worry about possibly repeated states
    - This could lead to a cycle or repeated paths toward a solution
  - A graph search if we try to avoid repeated states





Measuring Search Strategies Performance

- A strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions
  - Effectiveness: how good is the solution found?
    - Completeness: is the algorithm guaranteed to find a solution, when there is one?
    - Optimality: when a solution is found, is its path cost minimal?
  - Efficiency: what is the processing cost of finding a solution (computational complexity)?
    - Time complexity: how long does it take to find a solution?
    - Space complexity: how much memory is needed?
    - Time and space complexity are measured in terms of
      - b – maximum branching factor of a node that needs to be considered
      - d – depth of the least-cost solution
      - m – maximum depth of the state space
- Often a trade-off between effectiveness and efficiency is required

Computational complexity of search algorithms

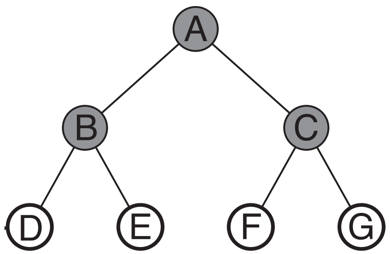
- Worst-case time complexity
  - The highest number of nodes that are generated before a solution is found (if any)
- Worst-case space complexity:
  - The highest number of nodes that must be simultaneously stored in memory

Evaluating BFS

- In terms of effectiveness, it can be easily shown that BFS is
  - Complete: a solution is always found if one exists
  - Non-optimal: it is not guaranteed that the solution with minimum path cost is found (if any) unless the path cost is a non-decreasing function of depth

BFS: computational complexity

- In the specific case of BSF, it is not difficult to see that computational complexity depends on two main factors
  - The number of successors of each node of the search tree
  - The depth d of the shallowest solution, which is the one found by BFS
- Since different nodes can have a different number of successors (see e.g., 8-puzzle and route finding on maps), to simplify computations a constant number of successors b, named branching factor, is considered
- For instance, for b=2 we have a binary tree



BFS: computational complexity

- Fixed b, the computational complexity can be evaluated as a function of d only
- Time complexity
  - In the worst case, the goal state is in the last node to be expanded among all the ones at depth d
    - This means that all the other nodes at depth d are expanded before
  - The number of generated nodes can be computed by evaluating the number of nodes that are generated at each depth

Depth	**Number of generated nodes**
0	1 (root node)
1	b
2	b^2
3	b^3
$\dots$	$\dots$
d	b^d
$d+1$	$b^{d+1} - b$
Total:	$1 + b + b^2 + b^3 + \dots + b^d + (b^{d+1} - b)$

BFS: computational complexity

- Space complexity
  - All generated nodes in memory until a solution is found
  - It follows that the space complexity equals the time complexity
- The worst-case time and space complexity of BFS, given b and the shallowest solution at depth d, are

$$1 + b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - b) = O(b^{d+1})$$

BFS: computational complexity

- As an example of what exponential complexity means, consider a search problem with the following settings
  - Branching factor b = 10
  - Time for generating one node: 10^{-4} s
  - Storage required for a single node: 1.000 bytes
- Worst-case time and space complexity of BFS, as a function of the depth d of the shallowest solution

Depth	Nodes	Time	Memory
2	1,100	0.11 sec.	1 megabyte
4	111,100	11 sec.	106 megabytes
6	10^7	0.19 minutes	10 gigabytes
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Summarizing properties of BFS

- Complete
  - A solution is always found if any
- Non-optimal
  - It is not guaranteed that the solution with minimum path cost is found (if any) unless the path cost is a non-decreasing function of depth
- Exponential time and space complexity w.r.t. the depth of the shallowest solution

What about DFS?

- DFS effectiveness
  - DFS can get stuck carrying on with very long paths
    - Infinite paths possible
  - DFS has limited memory requirements
    - If all paths from a given node are all explored with no solutions found, the sub-tree rooted in that node is removed from memory
    - Only a single path from the root to a leaf node needs to be stored in memory during the search
      - The unexpanded sibling nodes for each node on the path also

DFS: Computational complexity

- DFS complexity is evaluated by assuming
  - All nodes have the same number of successors b (branching factor)
  - All solutions have the same depth m
  - m is also the maximum depth of the search tree, when loops are avoided (worst case)
- In the worst case, the goal state is in the last path explored
- Time complexity
  - All nodes up to length m are generated before the solution is found
- Space complexity
  - Only a single path from the root to a leaf node needs to be stored

DFS: Computational complexity

Depth	Time complexity **N. of generated nodes**	Space complexity **N. of stored nodes**
0	1 (root node)	1 (root node)
1	b	b
2	b^2	b
...	...	...
m	b^m	b
Total:	$1 + b + \ldots + b^m = \mathcal{O}(b^m)$	$1 + mb = \mathcal{O}(m)$

- Time complexity exponential w.r.t. depth m
- Space complexity linear

Properties of DFS

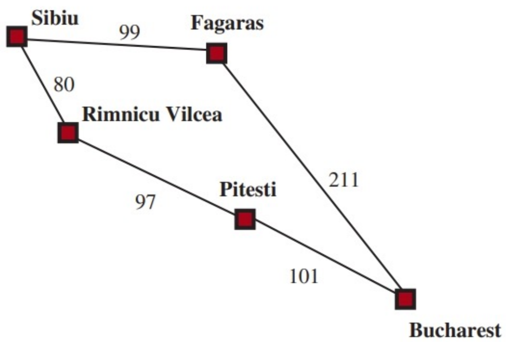
- Complete
  - Unless there are infinite paths
- Non-optimal
  - A deeper, suboptimal solution can be found along a path that is explored before an optimal solution path at a smaller depth
- Exponential time complexity and linear space complexity

Other strategies

- Uniform-cost
  - Expands the leaf node with the lowest path cost
- Depth-limited
  - Depth-first search with a predefined depth limit (avoid infinite paths, but not complete)
- Iterative-deepening depth-first
  - Repeated depth-limited search with depth limit 1, 2, 3, …, until a solution is found (avoid infinite paths and complete)
- Bidirectional
  - Simultaneously searching forward from the initial state and backwards from the goal sate, until the two searches meet

Uniform-cost Search

- When actions have different costs the node to expand is the one with minimal cost, where the cost of the path from the root to the current node is considered
- Expand the least-cost unexpanded node
  - The frontier is ordered by path cost, the lowest first
  - Equivalent to BFS if step costs are all equal



Depth-limited Search

- The depth-limited search keeps DFS from wandering down an infinite path, setting a depth limit l
  - It treats all nodes at depth l as if they had no further nodes to move on
- Sometimes a good depth limit can be chosen based on knowledge of the problem
  - For example, on the map of Romania there are 20 cities, so l=19 is a valid limit
  - However, any city can be reached from any other city in at most 9 actions. This number, known as the diameter of the state-space graph, gives us a better depth limit
- For most problems, we will not know a good depth limit until we have solved the problem

Iterative Deepening Search

- Iterative deepening search solves the problem of picking a good value for l by trying all values: first 0, then 1, then 2, and so on
  - until either a solution is found, or the depth-limited search returns the failure value
- Iterative deepening combines many of the benefits of depth-first and breadth-first search
- In general, iterative deepening is the preferred uninformed search method when
  - the search state space is larger than can fit in memory and
  - the depth of the solution is not known

Bidirectional Search

- An alternative approach called bidirectional search simultaneously searches forward from the initial state and backward from the goal state(s), hoping that the two searches will meet
- We need to track of two frontiers and two tables of explored states
- Reasoning backwards
  - if state **t** is a successor of **s** in the forward direction, then we need to know that s is a successor of **t** in the backward direction
  - A solution is when the two frontiers collide

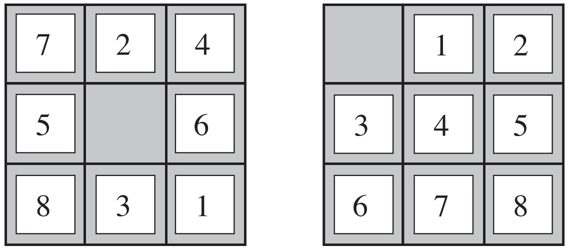
Uninformed search algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
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	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: [1] complete if b is finite, and the state space either has a solution or is finite. [2] complete if all action costs are $\geq \epsilon > 0$; [3] cost-optimal if action costs are all identical; [4] if both directions are breadth-first or uniform-cost.

Effectiveness of uninformed search

- One may think that the high computational complexity of uninformed search strategies is an issue only for real-world problems, not for toy ones
- Consider again 8-puzzle, apparently a very simple toy problem:



Start State Goal State

- How long does it take to solve it using, for instance, BFS?

Effectiveness of uninformed search

- 8-puzzle
  - the state space contains 9! = 362.880 distinct states (only 9!/2 = 181.440 are reachable from any given initial state)
  - it can be shown that the average solution depth (over all possible pairs of initial and goal states) is about 22
  - the average branching factor *b* (over all possible states) is about 3 (note that from each state 2 to 4 actions can be performed)
- How many nodes does BFS generate and store, when the shallowest solution has depth d = 22 (i.e., in the **average** case)?
- Remember that the worst-case time and space complexity of BFS is $O(b^{d+1})$, which in this case amounts to $3^{23} \approx 9 \times 10^{10}$ ...
- For instance, considering that representing a state requires at least $\lceil \log_2 9! \rceil = 19$ bits, storing 3^{23} states requires about $19 \times 9 \times 10^{10}$ bits, i.e., more than 200 GB...

Suggested exercises

1. Implement the **general tree-search algorithm**, and the related data structures, in Python
2. Implement the **additional**, specific functions for breadth-first, depth-first and uniform-cost search
3. Implement the **additional**, specific data structures and functions for the 8-puzzle problem, and the route-finding problem in the Romania map
4. Run the above search algorithms on specific problem instances, and evaluate the number of generated and stored nodes