CS 1571 Introduction to AI
Lecture 4

Uninformed search methods (cont.)
Informed search methods

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Administration

Office hours for the TA of the course:

- **Thursday 2:00-3:30pm, SNQ 6516**
- **Friday 10:00-11:30am , SNQ 6516**

- **The first part of each session has recitation format**

Topics

Uninformed search methods

- Review of uninformed search methods
- **Checking state repeats**
- **Uniform cost search**

Informed search methods

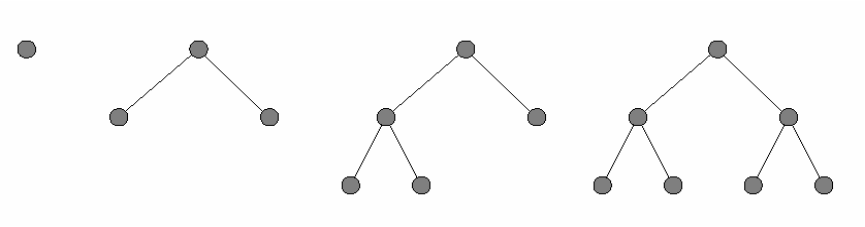
- Incorporating additional information to guide the search
- **Best first search**
  - Greedy methods
  - A* search
  - IDA*
- Heuristics

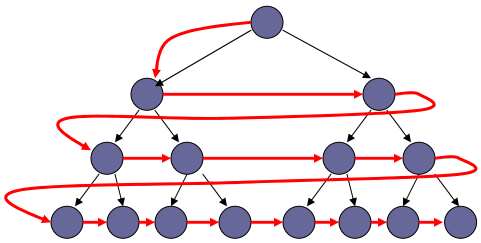
Uninformed methods

- Uninformed search methods use only information available in the problem definition
  - **Breadth-first search (BFS)**
  - **Depth-first search (DFS)**
  - **Iterative deepening (IDA)**
  - **Bi-directional search**
- **For the minimum cost path problem:**
  - **Uniform cost search**

Breadth first search (BFS)

- The shallowest node is expanded first





Properties of breadth-first search

- **Completeness:** **Yes.** The solution is reached if it exists.
- **Optimality:** **Yes**, for the shortest path.
- **Time complexity:**

$$1 + b + b^2 + \ldots + b^d = O(b^d)$$

exponential in the depth of the solution *d*

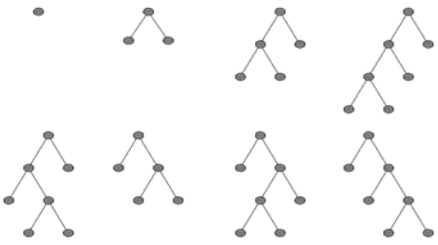
- **Memory (space) complexity:**

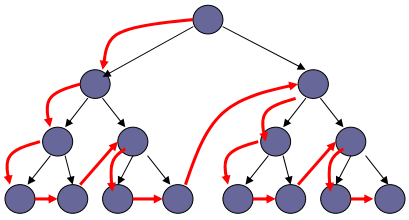
$$O(b^d)$$

same as time - every node is kept in the memory

Depth-first search (DFS)

- The deepest node is expanded first
- Backtrack when the path cannot be further expanded





Properties of depth-first search

- **Completeness:** **No.** Infinite loops can occur.
- **Optimality:** **No.** Solution found first may not be the shortest possible.
- **Time complexity:**

  $O(b^m)$

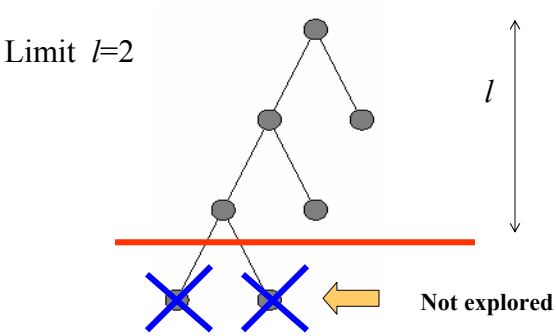
  exponential in the maximum depth of the search tree *m*
- **Memory (space) complexity:**

  $O(bm)$

  linear in the maximum depth of the search tree *m*

Limited-depth depth first search

- The limit *(l)* on the depth of the depth-first exploration

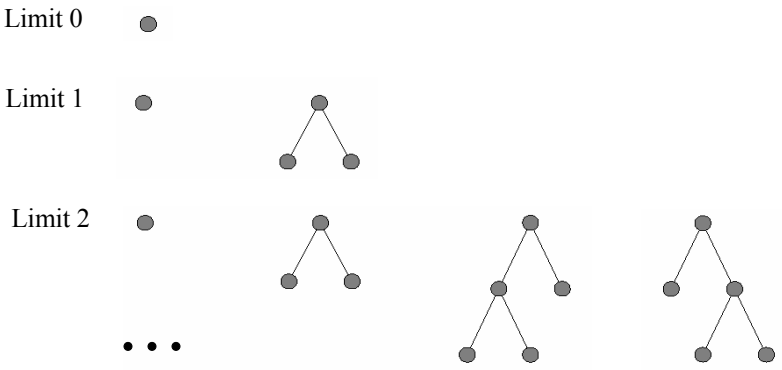


- **Time complexity:** $O(b^l)$
- **Memory complexity**: $O(bl)$

l - is the given limit

Iterative deepening algorithm (IDA)

- Progressively increases the limit of the limited-depth depth-first search



Properties of IDA

- **Completeness:** **Yes.** The solution is reached if it exists.
  (the same as BFS)
- **Optimality:** **Yes**, for the shortest path.
  (the same as BFS)
- **Time complexity:**

$$O(1) + O(b^1) + O(b^2) + \ldots + O(b^d) = O(b^d)$$

exponential in the depth of the solution *d*

worse than BFS, but asymptotically the same

- **Memory (space) complexity:**

$$O(db)$$

linear in the depth of the solution - much better than BFS

Elimination of state repeats

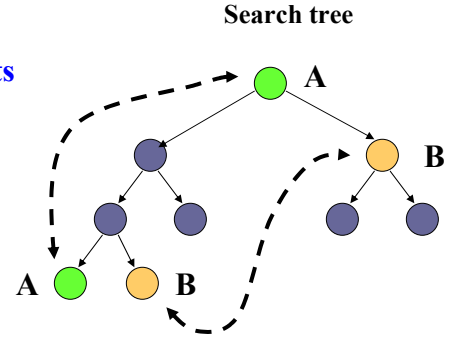
While searching the state space for the solution we can encounter the same state many times.

Question: Is it necessary to keep and expand all copies of states in the search tree?

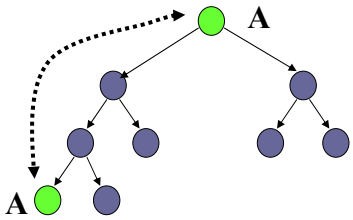
Two possible cases:

(A) Cyclic state repeats

(B) Non-cyclic state repeats



Elimination of cycles

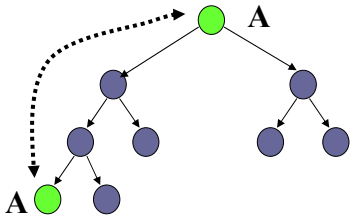


Case A: Corresponds to the path with a cycle

Can the branch (path) in which the same state is visited twice ever be a part of the optimal (shortest) path between the initial state and the goal? **No !!**

Branches representing cycles cannot be the part of the shortest solution and can be eliminated.

Elimination of cycles

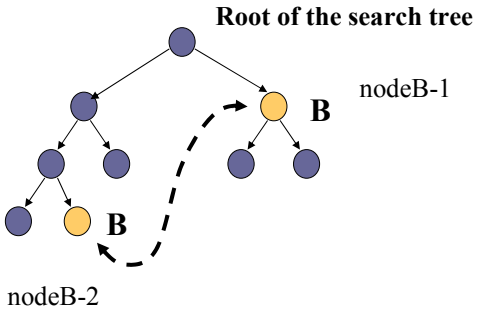


How to check for cyclic state repeats:

Check ancestors in the tree structure.

Do not expand the node with the state that is the same as the state in one of its ancestors.

Elimination of non-cyclic state repeats

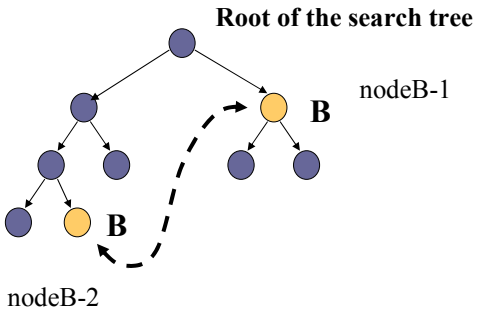


Case B: nodes with the same state are not on the same path from the initial state

Is one of the nodes nodeB-1, nodeB-2 better or preferable?

Yes. nodeB-1 represents the shorter path between the initial state and B

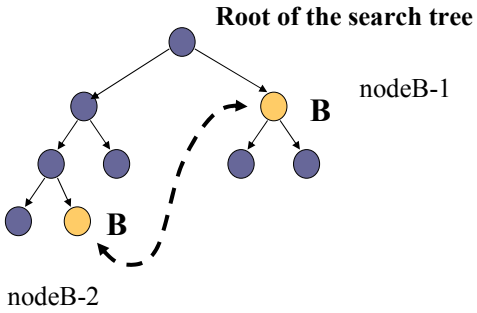
Elimination of non-cyclic state repeats



Since we are happy with the optimal solution nodeB-2 can be eliminated. It does not affect the optimality of the solution.

Problem: Nodes can be encountered in different order during different search strategies.

Elimination of non-cyclic state repeats with BFS



Breadth FS is well behaved with regard to non-cyclic state repeats: nodeB-1 is always expanded before nodeB-2

- Order of expansion determines the correct elimination strategy
- we can safely eliminate the node that is associated with the state that has been expanded before

Elimination of state repeats for the BFS

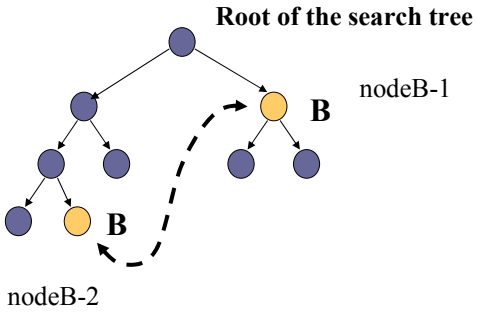
For **the breadth-first search (BFS)**

- we can safely eliminate all second, third, fourth, etc. occurrences of the same state
- this rule covers both the cyclic and non-cyclic repeats !!!

Implementation of all state repeat elimination through **marking**:

- All expanded states are marked
- All marked states are stored in a special data structure (a hash table)
- Checking if the node has ever been expanded corresponds to the mark structure lookup

Elimination of non-cyclic state repeats with DFS



Depth FS: nodeB-2 is expanded before nodeB-1

- The order of node expansion does not work imply correct elimination strategy
- we need to remember the length of the path between nodes to safely eliminate them

Elimination of all state redundancies

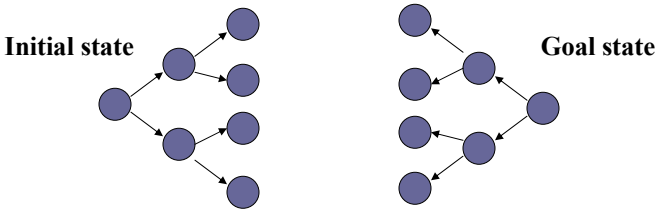
- **General strategy:** A node is redundant if there is another node with exactly the same state and a shorter path from the initial state
  - Works for any search method
  - Uses additional path length information

Implementation: marking with the minimum path value:

- The new node is redundant and can be eliminated if
  - it is in the hash table (it is marked), and
  - its path is longer or equal to the value stored.
- Otherwise the new node cannot be eliminated and it is entered together with its value into the hash table. (if the state was in the hash table the new path value is better and needs to be overwritten.)

Bi-directional search

- In some search problems we want to find the path from the initial state to the unique goal state (e.g. traveler problem)
- **Bi-directional search:**



  - Search both from the initial state and the goal state;
  - Use inverse operators for the goal-directed search.

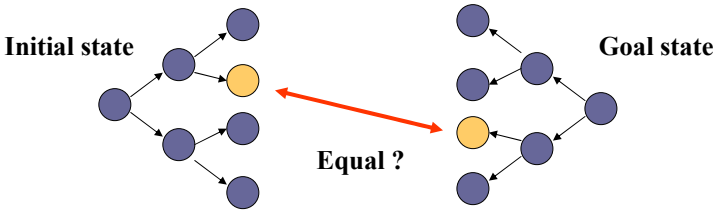
Bi-directional search

When does it help?

- It cuts the size of the search tree by half.

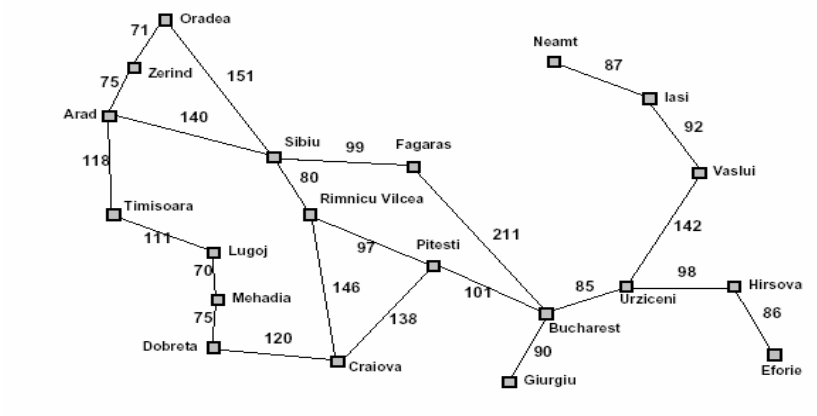
What is necessary?

- Merge the solutions.



Minimum cost path search

Traveler example with distances [km]



Optimal path: the shortest distance path from Arad to Bucharest

Searching for the minimum cost path

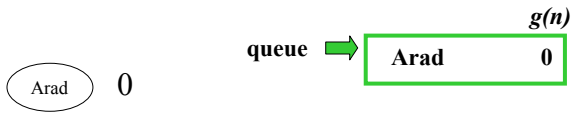
- **General minimum cost path-search problem:**
  - **adds weights or costs** to operators (links)
- "intelligent" expansion of the search tree should be driven by the cost of the current (partially) built path

Path cost function $g(n)$; path cost from the initial state to n

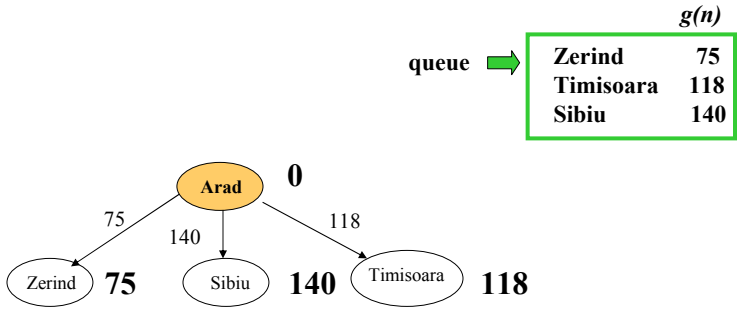
- Expand the leaf node with the minimum $g(n)$ first.
  - This is what the breadth first search does when operator costs are all equal to 1.
- The basic algorithm for finding the minimum cost path:
  - **Dijkstra's shortest path**
- In AI, the strategy goes under the name
  - **Uniform cost search**

Uniform cost search

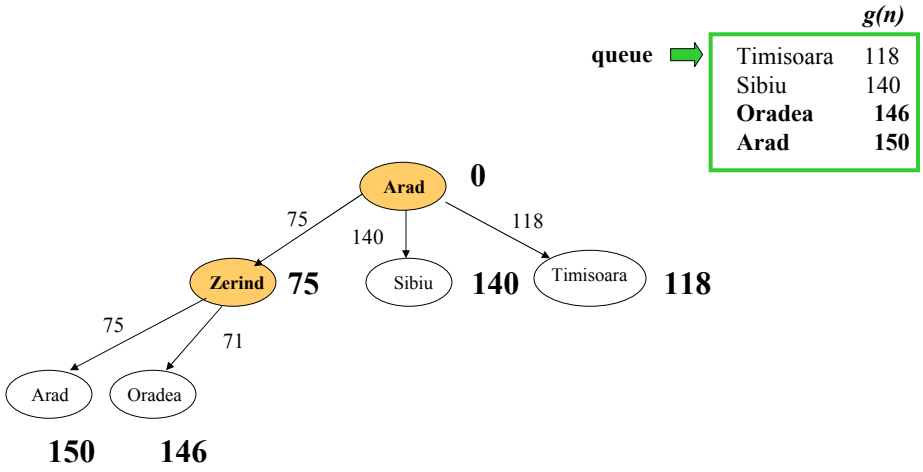
- Expand the node with the minimum path cost first
- **Implementation:** priority queue



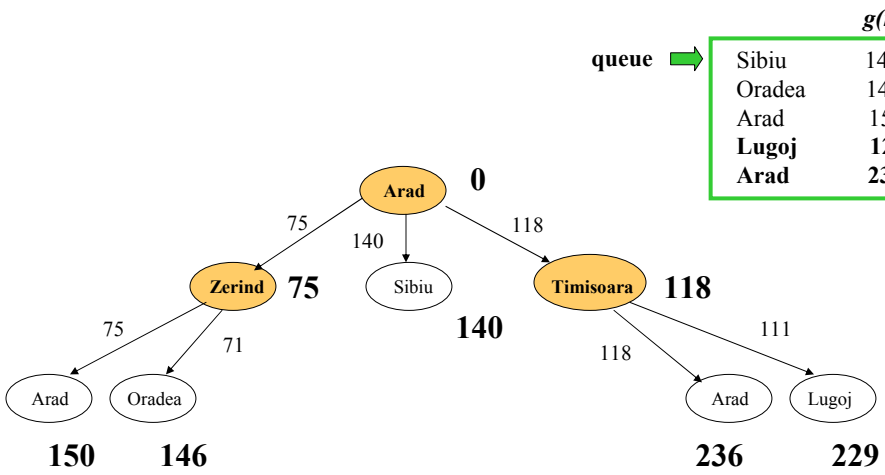
Uniform cost search



Uniform cost search



Uniform cost search



Properties of the uniform cost search

- **Completeness:** **Yes**, assuming that operator costs are non-negative (the cost of path never decreases)

$$g(n) \leq g(\text{successor } (n))$$

- **Optimality:** **Yes.** Returns the least-cost path.

- **Time complexity:**
  number of nodes with the cost *g(n)* smaller than the optimal cost

- **Memory (space) complexity:**
  number of nodes with the cost *g(n)* smaller than the optimal cost

Elimination of redundant tree nodes

- **The path-cost method for the safe elimination of all redundant state repeats works also for the uniform cost search:**
  - A node is redundant if there is another node with exactly the same state and a shorter path from the initial state

Implementation: marking with the minimum path value:

- The new node is redundant and can be eliminated if
  - it is in the hash table (it is marked), and
  - its path is longer or equal to the value stored.
- Otherwise the new node cannot be eliminated and it is entered together with its value into the hash table. (if the state was in the hash table the new path value is better and needs to be overwritten.

Informed search method

Additional information to guide the search

- **Uninformed search methods**
  - use only the information from the problem definition; and
  - past explorations, e.g. cost of the path generated so far.

- **Informed search methods**
  - incorporate additional measure of a potential of a specific state to reach the goal
  - a potential of a state (node) to reach a goal is measured through a **heuristic function**

- Heuristic function is denoted $h(n)$

Search with a node evaluation function

- A search strategy can be defined in terms of **a node evaluation function**
- **Evaluation function**
  - Denoted $f(n)$
  - measures the desirability of a node to be expanded next
- **Search: expand the node (state) with the best evaluation function value**
- **Implementation:** successors of the expanded node are inserted into the **priority queue** in the decreasing order of their evaluation function value
- **Uniform cost search:**
  - A special case of the search with an evaluation function

$$f(n) = g(n)$$

Best-first search

Best-first search

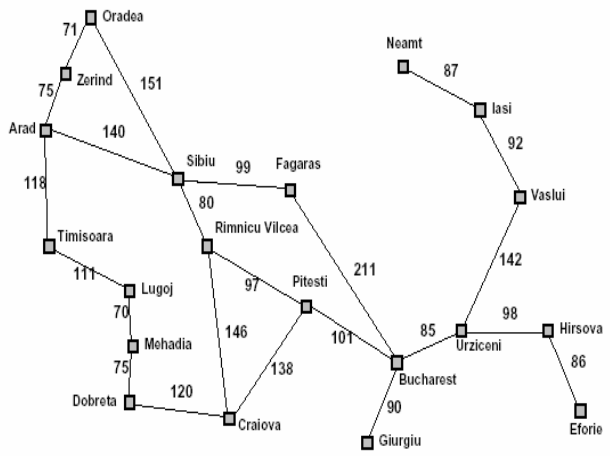
- Relies on the evaluation function $f(n)$ to guide the growth of the search tree and nodes expansions
- incorporates a heuristic function $h(n)$ into the evaluation function.

Special cases (differ in the design of evaluation function):

- **Greedy search** $f(n) = h(n)$
- **A* algorithm** $f(n) = g(n) + h(n)$

+ **iterative deepening** version of A* : **IDA***

Example: traveler problem with straight-line distance information



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- **Straight-line distances** give an estimate of the cost of the path from that city to Bucharest (a good heuristic)