CSE 411: Artificial Intelligence (Elective Course #6)

400 Level, Mechatronics Engineering
2nd Term 2016/2017, Lecture #4

Hazem Shehata

Dept. of Computer & Systems Engineering
Zagazig University

March 20th, 2017

Credits to Dr. Mohamed El Abd for the slides

Adminstrivia

Notes

- Assignment #1:
  - Due today.

Course Info:

- Website: http://hshehata.github.io/courses/zu/cse411/
- Office hours: Sunday 11:30am - 12:30pm

Search

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost).

Search

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost).
  - Algorithms (last week): **BFS**, **DFS**.

Search

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost).
  - Algorithms (last week): **BFS**, **DFS**.
  - Algorithms (this week): **DLS**, **IDS**, **UCS**.

Search

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost).
  - Algorithms (last week): **BFS**, **DFS**.
  - Algorithms (this week): **DLS**, **IDS**, **UCS**.

- **Informed Search:**
  Has additional information that allows it to judge the promise of an action, *i.e.*, the estimated cost from a state to a goal.

Search

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost).
  - Algorithms (last week): **BFS**, **DFS**.
  - Algorithms (this week): **DLS**, **IDS**, **UCS**.
- **Informed Search:**
  Has additional information that allows it to judge the promise of an action, *i.e.*, the estimated cost from a state to a goal.
  - Algorithms: **GBFS**, **A***.

Outline

Outline

Uninformed Search

Uninformed Search algorithms

- Breadth-first search (BFS),
- Depth-first search (DFS),
- Depth-limited search,
- Iterative deepening search (IDS),
- Uniform-cost search

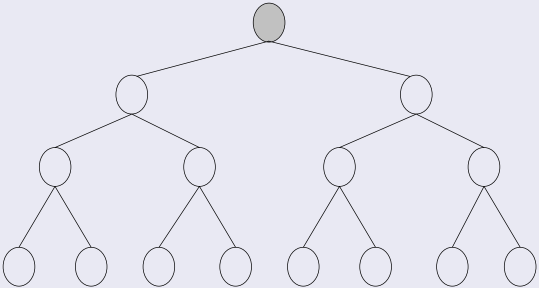
Uninformed Search

Breadth-first search

- The frontier is implemented as a FIFO queue,
- The tree is traversed on a level-by-level basis.

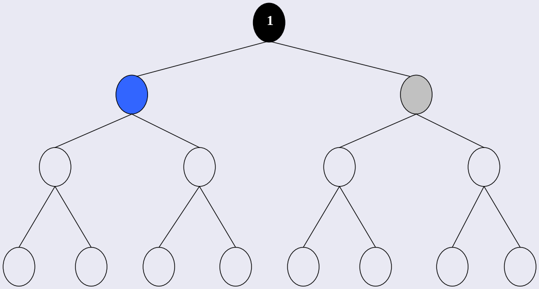
Uninformed Search

Breadth-first search



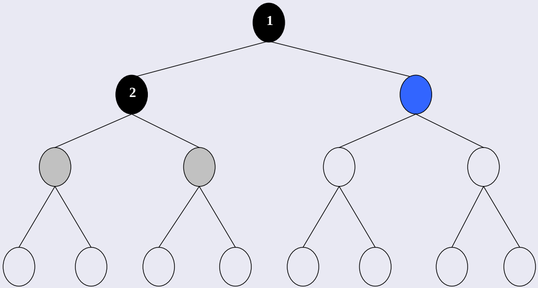
Uninformed Search

Breadth-first search



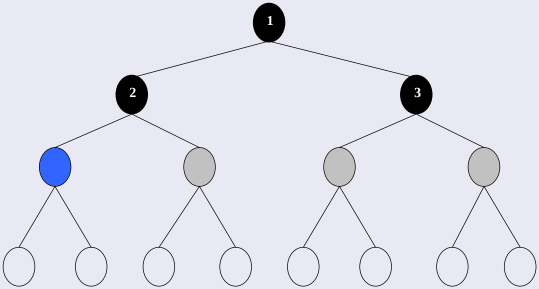
Uninformed Search

Breadth-first search



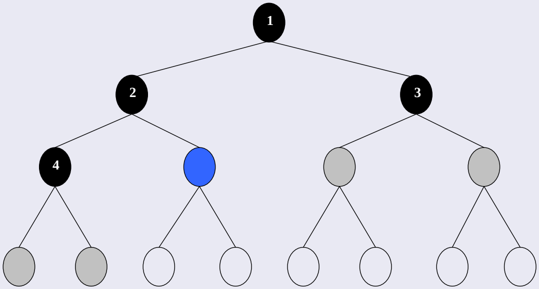
Uninformed Search

Breadth-first search



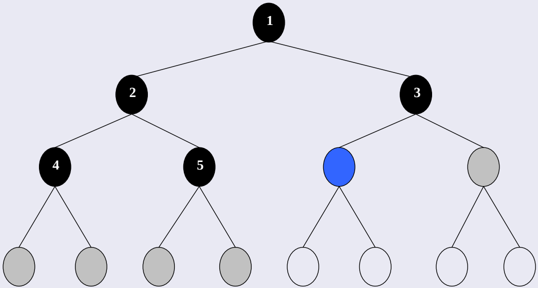
Uninformed Search

Breadth-first search



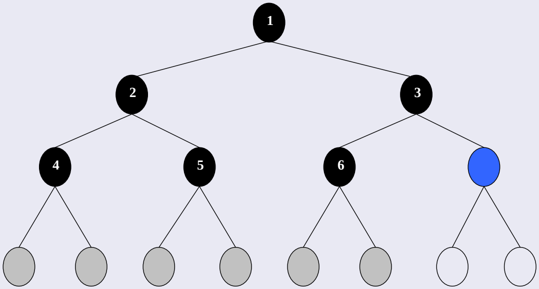
Uninformed Search

Breadth-first search



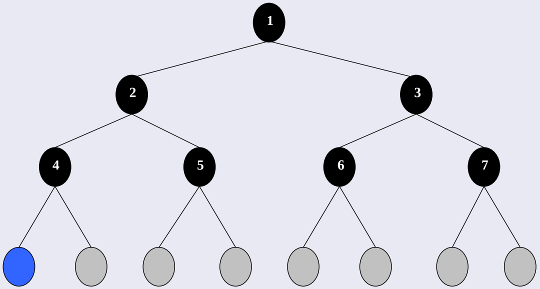
Uninformed Search

Breadth-first search



Uninformed Search

Breadth-first search



Search Algorithms

Breadth-first search (tree version)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE=problem.INITIAL-STATE, PATH-COST=0
  if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
  frontier ← a FIFO queue with node as the only element
  loop do
    if frontier.EMPTY?() then return failure
    node ← frontier.POP()            /* choose shallowest node in frontier */
    for each action in problem.ACTIONS(node.STATE) do
      child ← node.CHILD-NODE(problem, action)
      if problem.GOAL-TEST(child.STATE) then return child.SOLUTION()
      frontier.INSERT(child)
```

Search Algorithms

Breadth-first search (graph version)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE=problem.INITIAL-STATE, PATH-COST=0
  if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if frontier.EMPTY?() then return failure
    node ← frontier.POP()            /* choose shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← node.CHILD-NODE(problem, action)
      if child.STATE is not in explored and not in frontier then
        if problem.GOAL-TEST(child.STATE) then return child.SOLUTION()
        frontier.INSERT(child)
```

Uninformed Search

Breadth-first search

BFS properties:

- Complete (if b is finite).
- Optimal, if path cost is equal to depth:
  - Guaranteed to return the shallowest goal (depth d).
- Time complexity = $O(b^d)$.
- Space complexity = $O(b^d)$.

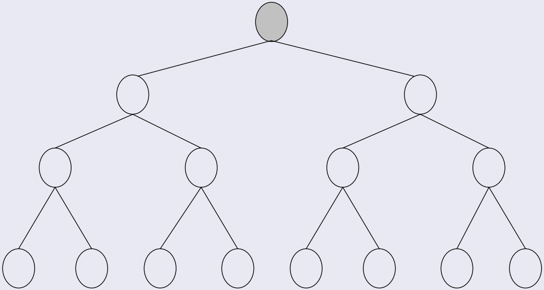
Uninformed Search

Depth-first search

- Algorithm is similar to BFS, except that the frontier is implemented as a LIFO queue (*i.e.*, stack).
- Algorithm always expands deepest unexpanded node in current frontier.

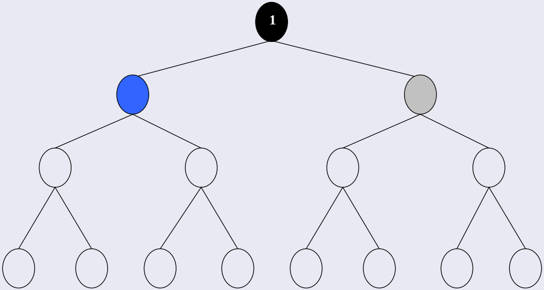
Uninformed Search

Depth-first search



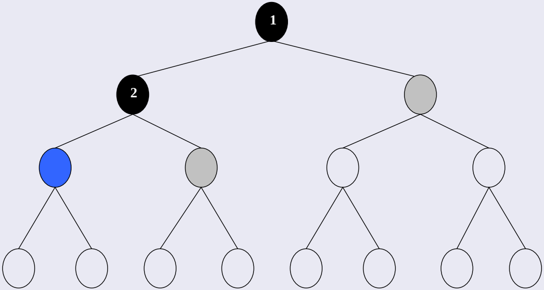
Uninformed Search

Depth-first search



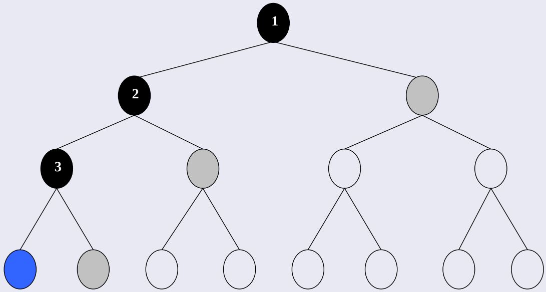
Uninformed Search

Depth-first search



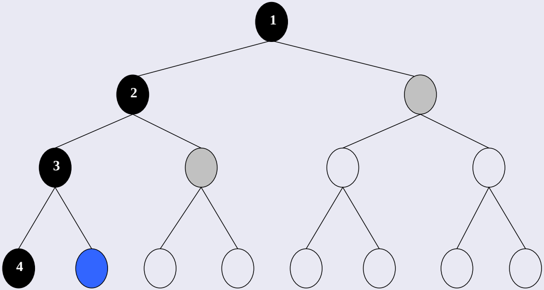
Uninformed Search

Depth-first search



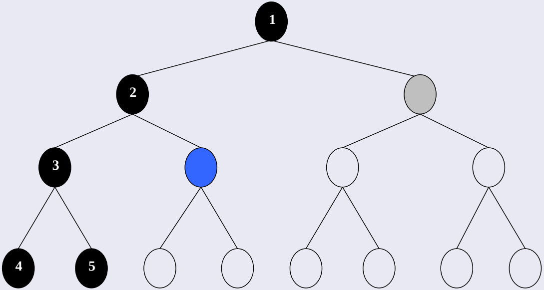
Uninformed Search

Depth-first search



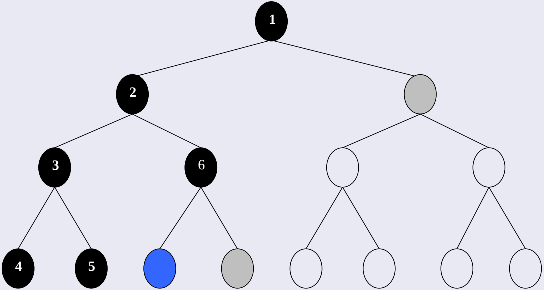
Uninformed Search

Depth-first search



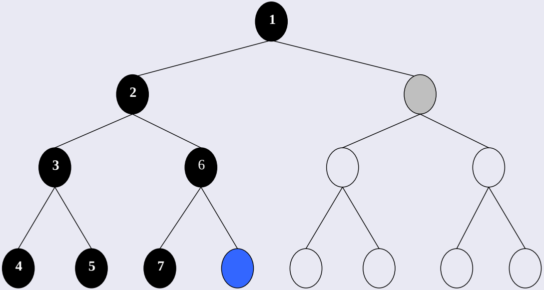
Uninformed Search

Depth-first search



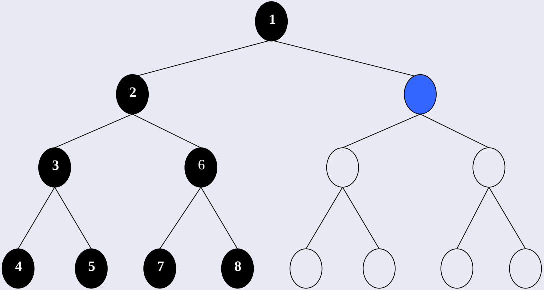
Uninformed Search

Depth-first search



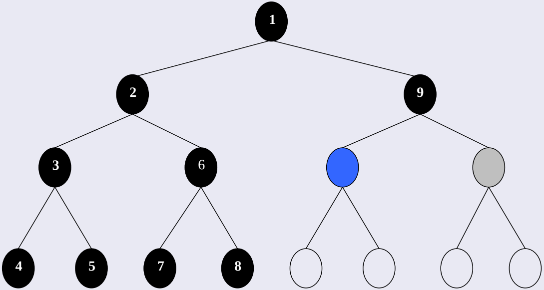
Uninformed Search

Depth-first search



Uninformed Search

Depth-first search



Uninformed Search

Depth-first search

DFS properties:

- Not complete (tree version).
- Not Optimal.
- Time complexity = $O(b^m)$.
- Space complexity = $O(bm)$.

Uninformed Search

Depth-limited search

- Depth-first search with depth limit l.

Uninformed Search

Depth-limited search

- Depth-first search with depth limit l.
- Algorithm is a simple modification to the general tree-search or graph-search algorithm
  - It takes l as an extra argument.
  - It returns one of the following:
    - *solution* (solution found with the depth limit).
    - *failure* (no solution found in entire tree; $l \geq m$).
    - *cutoff* (no solution found within depth limit; $l < m$).

Uninformed Search

Depth-limited search

- Depth-first search with depth limit l.
- Algorithm is a simple modification to the general tree-search or graph-search algorithm
  - It takes l as an extra argument.
  - It returns one of the following:
    - *solution* (solution found with the depth limit).
    - *failure* (no solution found in entire tree; $l \geq m$).
    - *cutoff* (no solution found within depth limit; $l < m$).
- Avoids problems of depth-first search when trees are unbounded.

Uninformed Search

Depth-limited search

- Depth-first search with depth limit l.
- Algorithm is a simple modification to the general tree-search or graph-search algorithm
  - It takes l as an extra argument.
  - It returns one of the following:
    - *solution* (solution found with the depth limit).
    - *failure* (no solution found in entire tree; $l \geq m$).
    - *cutoff* (no solution found within depth limit; $l < m$).
- Avoids problems of depth-first search when trees are unbounded.
- Depth-first search is depth-limited search with $l = \infty$.

Uninformed Search

Depth-limited search

- Not complete (unless $l = d$).

Uninformed Search

Depth-limited search

- Not complete (unless $l = d$).
- Not optimal (unless $l = d$).

Uninformed Search

Depth-limited search

- Not complete (unless $l = d$).
- Not optimal (unless $l = d$).
- Time complexity = $O(b^l)$.

Uninformed Search

Depth-limited search

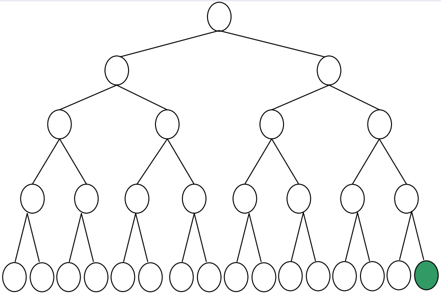
- Not complete (unless $l = d$).
- Not optimal (unless $l = d$).
- Time complexity = $O(b^l)$.
- Space complexity = $O(bl)$.

Uninformed Search

Depth-limited search

DLS properties:

- Upper-bound case for time: goal is last node of last branch:
  - Number of nodes generated: b nodes for each node of l levels (entire tree to depth l).
  - Time complexity: all generated nodes $O(b^l)$.

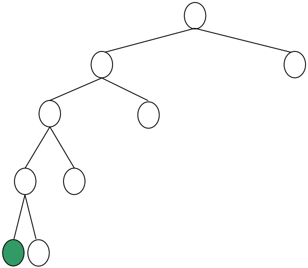


Uninformed Search

Depth-limited search

DLS properties:

- Upper-bound case for space: goal is last node of first branch:
  - Number of generated nodes: b nodes at each of l levels.
  - Space complexity: all generated nodes = $O(bl)$.



Uninformed Search

Iterative deepening search

- Depth-first search with increasing depth limit l: repeat depth-limited search over and over, with $l = l + 1$.

Uninformed Search

Iterative deepening search

- Depth-first search with increasing depth limit l: repeat depth-limited search over and over, with $l = l + 1$.
- Avoids problems of depth-first search when trees are unbounded.

Uninformed Search

Iterative deepening search

- Depth-first search with increasing depth limit l: repeat depth-limited search over and over, with $l = l + 1$.
- Avoids problems of depth-first search when trees are unbounded.
- Avoids problem of depth-limited search when goal depth $d > l$.

Uninformed Search

Iterative deepening search

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
  for *depth*=0 **to** ∞ **do**
    result $\leftarrow$ DEPTH-LIMITED-SEARCH(*problem*,*depth*)
    if *result* $\neq$ cutoff **then return** *result*

Uninformed Search

Iterative deepening search

- Complete (if b is finite).

Uninformed Search

Iterative deepening search

- Complete (if b is finite).
- Optimal.

Uninformed Search

Iterative deepening search

- Complete (if b is finite).
- Optimal.
- Time complexity = $O(b^d)$.

Uninformed Search

Iterative deepening search

- Complete (if b is finite).
- Optimal.
- Time complexity = $O(b^d)$.
- Space complexity = $O(bd)$.

Uninformed Search

Iterative deepening search

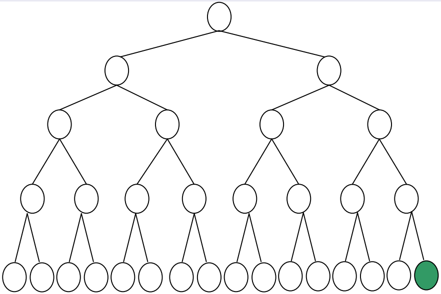
- Complete (if b is finite).
- Optimal.
- Time complexity = $O(b^d)$.
- Space complexity = $O(bd)$.
- Note: nodes on levels above d are generated multiple times.

Uninformed Search

Iterative deepening search

IDS properties:

- Upper-bound case for time: goal is last node of last branch:
  - Number of nodes generated: b nodes for each node of d levels.
  - Time complexity: all generated nodes $O(b^d)$.

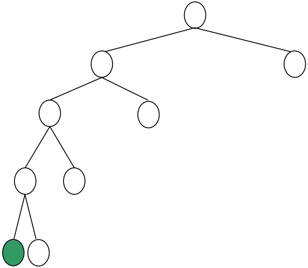


Uninformed Search

Iterative deepening search

IDS properties:

- Upper-bound case for space: goal is last node of first branch:
  - Number of generated nodes: b nodes at each of d levels.
  - Space complexity: all generated nodes = $O(bd)$.



Uninformed Search

Uniform-cost search

- A simple extension of BFS that works for any step-cost function.

Uninformed Search

Uniform-cost search

- A simple extension of BFS that works for any step-cost function.
- Instead of expanding the shallowest node, UCS expands the node n with the lowest path cost $g(n)$.

Uninformed Search

Uniform-cost search

- A simple extension of BFS that works for any step-cost function.
- Instead of expanding the shallowest node, UCS expands the node n with the lowest path cost $g(n)$.
- The frontier is implemented as a priority queue ordered by g.

Uninformed Search

Uniform-cost search

- A simple extension of BFS that works for any step-cost function.
- Instead of expanding the shallowest node, UCS expands the node n with the lowest path cost $g(n)$.
- The frontier is implemented as a priority queue ordered by g.
- Condition: No zero-cost or negative-cost edges, minimum cost is ε.

Uninformed Search

Uniform-cost search

- A simple extension of BFS that works for any step-cost function.
- Instead of expanding the shallowest node, UCS expands the node n with the lowest path cost $g(n)$.
- The frontier is implemented as a priority queue ordered by g.
- Condition: No zero-cost or negative-cost edges, minimum cost is ε.
- UCS algorithm differs from BFS algorithm in 2 aspects:
  - Goal test is applied to a node when it is selected for expansion.
  - In graph version, a test is added in case a better path is found to a node currently on the frontier.

Search Algorithms

Uniform-cost search (tree version)

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE=problem.INITIAL-STATE, PATH-COST=0
  frontier ← a priority queue ordered by PATH-COST, with node as only element
  loop do
    if frontier.EMPTY?() then return failure
    node ← frontier.POP()          /* choose lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
    for each action in problem.ACTIONS(node.STATE) do
      child ← node.CHILD-NODE(problem, action)
      frontier.INSERT(child, child.PATH-COST)
```

Search Algorithms

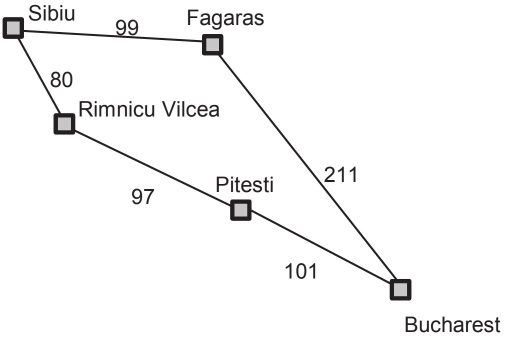
Uniform-cost search (graph version)

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure
  node ← a node with STATE=*problem*.INITIAL-STATE, PATH-COST=0
  frontier ← a **priority queue** ordered by PATH-COST, with *node* as only element
  explored ← an empty set
  loop do
    if *frontier*.EMPTY?() **then return** failure
    node ← *frontier*.POP()   /* choose lowest-cost node in frontier */
    if *problem*.GOAL-TEST(*node*.STATE) **then return** *node*.SOLUTION()
    add *node*.STATE to *explored*
    for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
      child ← *node*.CHILD-NODE(*problem*, *action*)
      if *child*.STATE is not in *explored* and not in *frontier* **then**
        frontier.INSERT(*child*, *child*.PATH-COST)
      else if *child*.STATE is in *frontier* with higher PATH-COST **then**
        replace that frontier node with *child*

Uninformed Search

UCS example - Romania map

Find the shortest path route from Sibiu to Bucharest.



Uninformed Search

Uniform-cost search

Uniform-cost search properties:

- Complete (if b is finite).

Uninformed Search

Uniform-cost search

Uniform-cost search properties:

- Complete (if b is finite).
- Optimal.

Uninformed Search

Uniform-cost search

Uniform-cost search properties:

- Complete (if b is finite).
- Optimal.
- Time complexity = $O(b^{1+\lfloor \frac{C^*}{\varepsilon} \rfloor}) \geq O(b^d)$.

Uninformed Search

Uniform-cost search

Uniform-cost search properties:

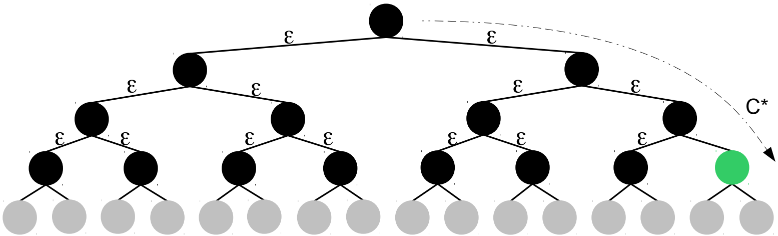
- Complete (if b is finite).
- Optimal.
- Time complexity = $O(b^{1+\lfloor \frac{C^*}{\varepsilon} \rfloor}) \geq O(b^d)$.
- Space complexity = $O(b^{1+\lfloor \frac{C^*}{\varepsilon} \rfloor}) \geq O(b^d)$.

Uninformed Search

Uniform-cost search

Uniform-cost search properties:

- Upper-bound case: goal has path cost C^*, all other actions have minimum cost of ε:

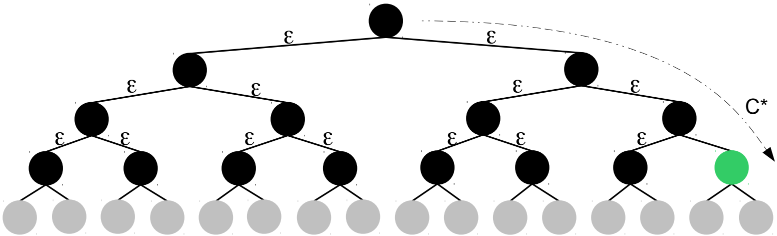


Uninformed Search

Uniform-cost search

Uniform-cost search properties:

- Upper-bound case: goal has path cost C^*, all other actions have minimum cost of ε:
  - Goal is detected once goal node is popped from frontier.

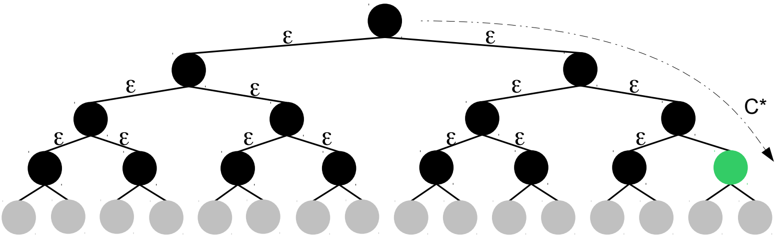


Uninformed Search

Uniform-cost search

Uniform-cost search properties:

- Upper-bound case: goal has path cost C^*, all other actions have minimum cost of ε:
  - Goal is detected once goal node is popped from frontier.
  - Depth explored before popping goal node: $1 + \lfloor \frac{C^*}{\varepsilon} \rfloor$.

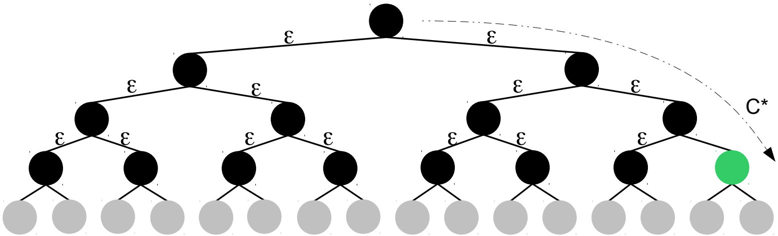


Uninformed Search

Uniform-cost search

Uniform-cost search properties:

- Upper-bound case: goal has path cost C^*, all other actions have minimum cost of ε:
  - Goal is detected once goal node is popped from frontier.
  - Depth explored before popping goal node: $1 + \lfloor \frac{C^*}{\varepsilon} \rfloor$.
  - Number of generated nodes: $O(b^{1+\lfloor \frac{C^*}{\varepsilon} \rfloor})$.

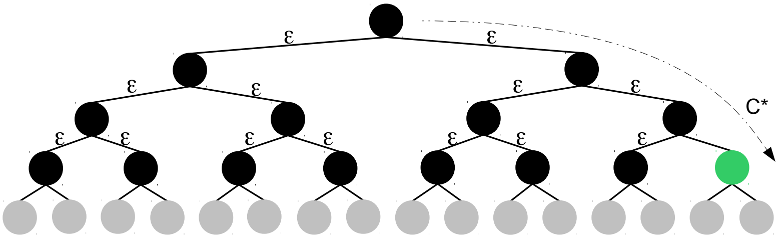


Uninformed Search

Uniform-cost search

Uniform-cost search properties:

- Upper-bound case: goal has path cost C^*, all other actions have minimum cost of ε:
  - Goal is detected once goal node is popped from frontier.
  - Depth explored before popping goal node: $1 + \lfloor \frac{C^*}{\varepsilon} \rfloor$.
  - Number of generated nodes: $O(b^{1+\lfloor \frac{C^*}{\varepsilon} \rfloor})$.
  - Space and time complexity: all generated nodes.



Uninformed Search

Summary of uninformed search algorithms

Criterion	**BFS**	**UCS**	**DFS**	**DLS**	**IDS**
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]
Optimal?	Yes[3]	Yes	No	No	Yes[3]
Time	$O(b^d)$	$O(b^{1+\lfloor \frac{C^*}{\varepsilon} \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor \frac{C^*}{\varepsilon} \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$

- Note 1: assuming finite branching factor b.
- Note 2: assuming minimum step cost $\varepsilon > 0$.
- Note 3: assuming equal step costs.

Outline

1. Uninformed Search (Cont.)

2. Requirements & Reading Material

Requirements

What do I need from you

- When given a certain problem you should be able to:

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the search tree up to a given depth.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the search tree up to a given depth.
  - Traverse the search tree according to a given strategy.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the search tree up to a given depth.
  - Traverse the search tree according to a given strategy.
- Answer descriptive questions.

Reading Material

Which parts of the textbook are covered

- Russell-Norvig, Chapters 3:
  - Pages 87 - 91.