Uninformed Search

Computer Science cpsc322, Lecture 5

(Textbook Chpt 3.5)

May 18, 2017

Recap

- **Search** is a key computational mechanism in many **AI agents**
- We will study the basic principles of search on the simple **deterministic planning agent model**

Generic search approach:

- define a search space graph,
- start from current state,
- incrementally explore paths from current state until goal state is reached.

Searching: Graph Search Algorithm with three bugs ☹

Input: a graph,

a start node,

Boolean procedure *goal(n)* that tests if *n* is a goal node.

frontier := { ⟨g⟩: g is a goal node }; ← should be initiated with start node

while *frontier* is not empty:

select and **remove** path $\langle n_0, n_1, \cdots, n_k \rangle$ from *frontier*;

if *goal*(n_k)

return $\langle n_k \rangle$; ← should return the path

for every neighbor *n* of n_k

add $\langle n_0, n_1, \cdots, n_k \rangle$ n ⟩ to *frontier*;

end while

No solution found

- The *goal* function defines what is a solution.
- The *neighbor* relationship defines the graph.
- Which path is selected from the frontier defines the search strategy.

Lecture Overview

- Recap
- Criteria to compare Search Strategies
- Simple (Uninformed) Search Strategies
  - Depth First
  - Breadth First

start playing with

Comparing Searching Algorithms: will it find a solution? the best one?

Def. (complete): A search algorithm is **complete** if, whenever at least one solution exists, the algorithm **is guaranteed to find a solution** within a finite amount of time.

Def. (optimal): A search algorithm is **optimal** if, when it finds a solution , it is the best solution

Comparing Searching Algorithms: Complexity

Def. (time complexity)

The time complexity of a search algorithm is an expression for the **worst-case** amount of time it will take to run,

- expressed in terms of the **maximum path length** *m* and the **maximum branching factor** *b*.

Def. (space complexity) : The space complexity of a search algorithm is an expression for the **worst-case** amount of memory that the algorithm will use (*number of nodes*),

- Also expressed in terms of *m* **and** *b*.

Lecture Overview

- **Recap**
- Criteria to compare Search Strategies
- Simple (Uninformed) Search Strategies
  - Depth First
  - Breadth First

Depth-first Search: DFS

- **Depth-first search** treats the frontier as a **stack**
- It always selects one of the last elements added to the frontier

Example:

push / pop

order in which these are added is not specified in pure DFS

- the frontier is $[p_1, p_2, \cdots, p_r]$
- neighbors of last node of p_1 (its end) are $\{n_1, \cdots, n_k\}$

- What happens?
  - p_1 is selected, and its end is tested for being a goal.

    If not...

  - New paths are created attaching $\{n_1, \cdots, n_k\}$ to p_1

    k new paths

  - These "replace" p_1 at the beginning of the frontier
  - Thus, the frontier is now $[(p_1, n_1), \cdots, (p_1, n_k), p_2, \cdots, p_r]$.
  - *NOTE:* p_2 is only selected when all paths extending p_1 have been explored.

Depth-first Search: Analysis of DFS

- Is DFS complete?
- Is DFS optimal?

Depth-first Search: Analysis of DFS

- What is the time complexity if the maximum path length is *m* and the maximum branching factor is *b* ?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

- What is the space complexity?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

Depth–first Search: Analysis of DFS Summary

Is DFS complete?

- Depth–first search isn't guaranteed to halt on graphs with cycles.
- However, DFS *is* complete for finite acyclic graphs.

Is DFS optimal?

What is the time complexity, if the maximum path length is *m* and the maximum branching factor is *b* ?

- The time complexity is ? $O(b^m)$?: must examine every node in the tree.
- Search is unconstrained by the goal until it happens to stumble on the goal.

What is the *space complexity*?

- Space complexity is ? $O(mb)$? the longest possible path is *m*, and for every node in that path must maintain a fringe of size *b*.

Analysis of DFS

Def. :Asearch algorithm is complete if whenever there is at least one solution, the algorithm is guaranteed to find it within a finite amount of time.

Is DFS complete?

No

- If there are cycles in the graph, DFS may get "stuck" in one of them
- see this in AISpace by adding a cycle to "Simple Tree"
  - e.g., click on "Create" tab, create a new edge from N7 to N1, g back to "Solve" and see what happens

Analysis of DFS

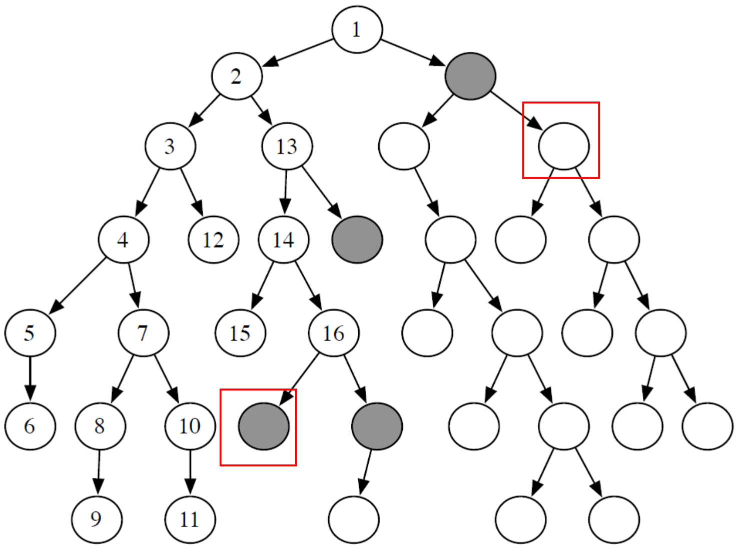
Def.: A search algorithm is optimal if when it finds a solution, it is the best one (e.g., the shortest)

Is DFS optimal?

Yes

No

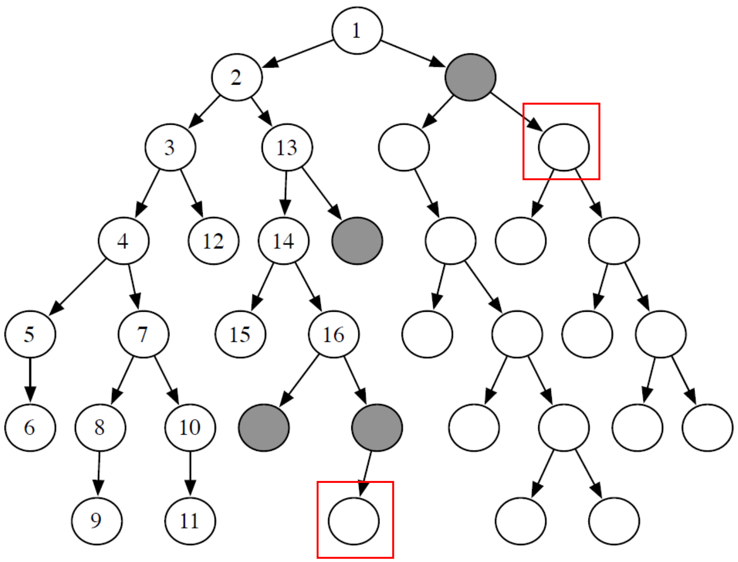
- E.g., goal nodes: red boxes



Analysis of DFS

Def.: A search algorithm is optimal if when it finds a solution, it is the best one (e.g., the shortest)

Is DFS optimal? No



- It can "stumble" on longer solution paths before it gets to shorter ones.
    - E.g., goal nodes: red boxes

- see this in AISpace by loading "Extended Tree Graph" and set N6 as a goal
    - e.g., click on "Create" tab, right-click on N6 and select "set as a goal node"

Analysis of DFS

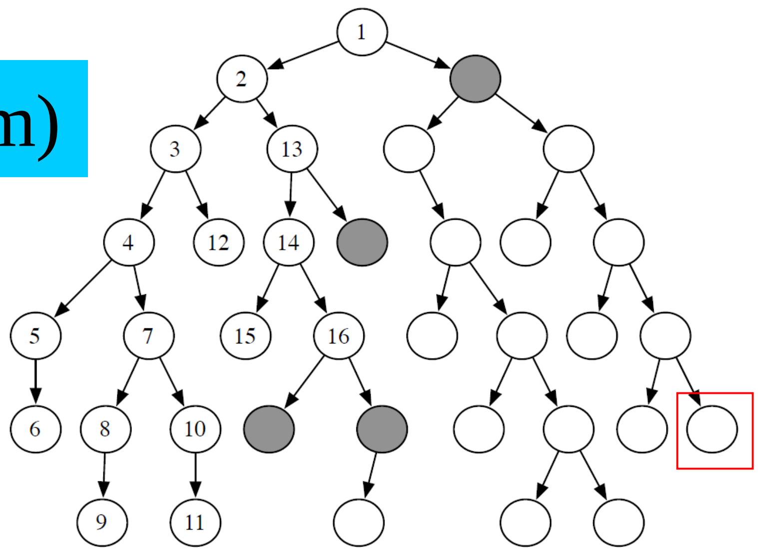
Def.: The time complexity of a search algorithm is

the worst-case amount of time it will take to run, expressed in terms of

- maximum path length *m*
- maximum forward branching factor *b*.

- What is DFS's time complexity, in terms of m and b ?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$



- E.g., single goal node -> red box

Analysis of DFS

Def.: The time complexity of a search algorithm is

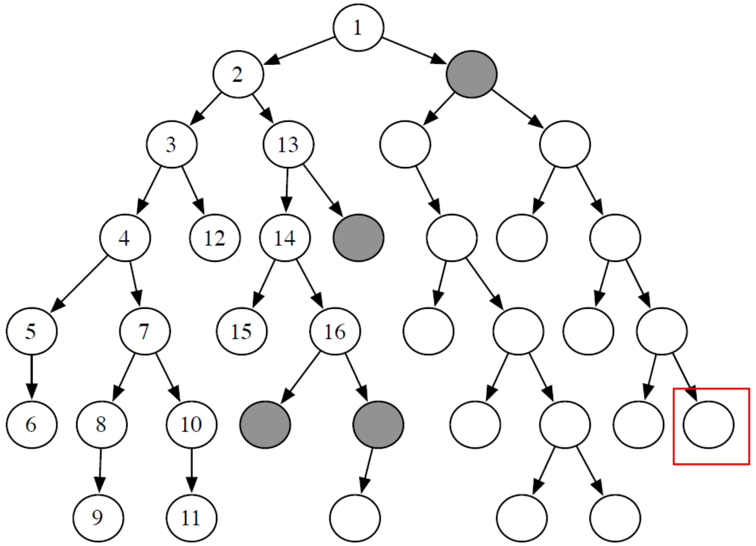
the worst-case amount of time it will take to run, expressed in terms of

- maximum path length *m*
- maximum forward branching factor *b*.

- What is DFS's time complexity, in terms of m and b ?

$O(b^m)$

- In the worst case, must examine every node in the tree
  - E.g., single goal node -> red box



Analysis of DFS

Def.: The space complexity of a search algorithm is the worst–case amount of memory that the algorithm will use (i.e., the maximal number of nodes on the frontier), expressed in terms of

- maximum path length *m*
- maximum forward branching factor *b*.

- What is DFS's space complexity, in terms of m and b ?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

See how this works in AIspace

Analysis of DFS

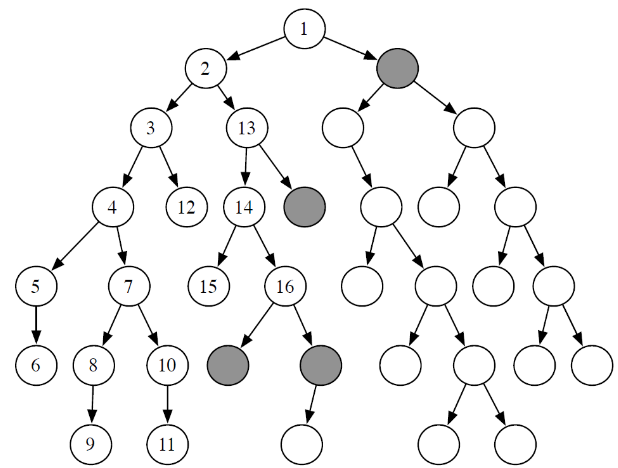
Def.: The space complexity of a search algorithm is the

worst-case amount of memory that the algorithm will use (i.e., the maximum number of nodes on the frontier), expressed in terms of

- maximum path length *m*
- maximum forward branching factor *b*.

- What is DFS's space complexity, in terms of m and b ?

O(bm)



- for every node in the path currently explored, DFS maintains a path to its unexplored siblings in the search tree
  - Alternative paths that DFS needs to explore
- The longest possible path is m, with a maximum of b-1 alterative paths per node

See how this works in AIspace

Depth-first Search: When it is appropriate?

A. There are cycles
B. Space is restricted (complex state representation e.g., robotics)
C. There are shallow solutions
D. You care about optimality

Depth-first Search: When it is appropriate?

Appropriate

- Space is restricted (complex state representation e.g., robotics)
- There are many solutions, perhaps with long path lengths, particularly for the case in which all paths lead to a solution

AIspace

clicker question?

Inappropriate

- Cycles
- There are shallow solutions
- if you care about optimality!

Why DFS need to be studied and understood?

- It is simple enough to allow you to learn the basic aspects of searching (When compared with breadth first)

- It is the basis for a number of more sophisticated / useful search algorithms

Lecture Overview

- Recap

- Simple (Uninformed) Search Strategies
  - Depth First
  - Breadth First

Breadth-first Search: BFS

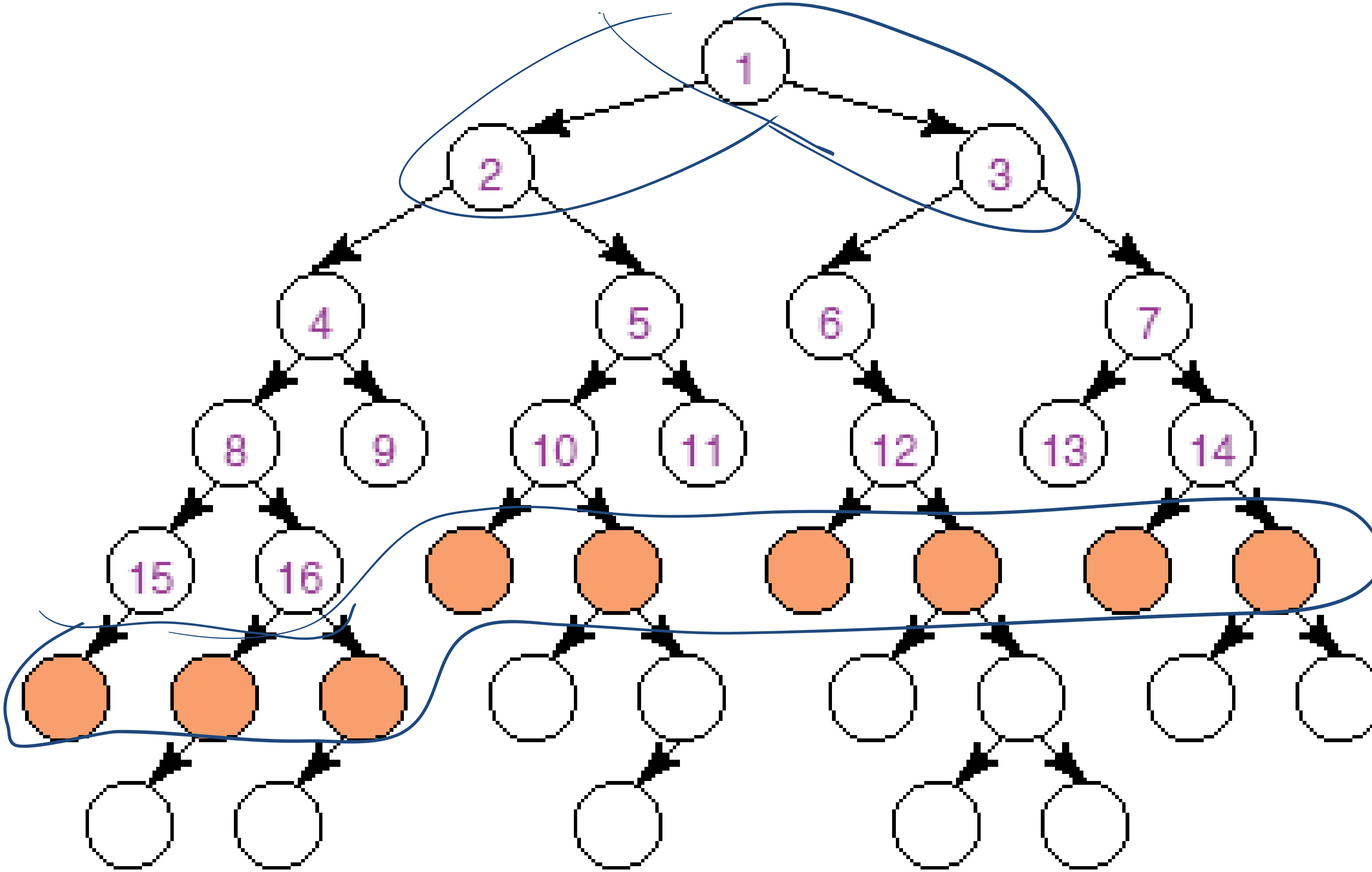
- Breadth-first search treats the frontier as a **queue**
  - it always selects one of the earliest elements added to the frontier

Example:

- the frontier is $[p_1, p_2, \cdots, p_r]$
- neighbors of the last node of p_1 are $\{n_1, \cdots, n_k\}$

- What happens?
  - p_1 is selected, and its end tested for being a path to the goal.
  - New paths are created attaching $\{n_1, \cdots, n_k\}$ to p_1
  - These follow p_r at the end of the frontier
  - Thus, the frontier is now $[p_2, \cdots, p_r, (p_1, n_1), \cdots, (p_1, n_k)]$.
  - p_2 is selected next.

Illustrative Graph - Breadth-first Search



Breadth-first Search: Analysis of BFS

- Is BFS complete?
- Is BFS optimal?

Breadth-first Search: Analysis of BFS

- What is the time complexity if the maximum path length is *m* and the maximum branching factor is *b* ?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

- What is the space complexity?

$O(b^m)$ $O(m^b)$ $O(bm)$ $O(b+m)$

Analysis of Breadth-First Search

- Is BFS complete?
  - Yes
  - In fact, BFS is guaranteed to find the path that involves the fewest arcs (why?)
- What is the time complexity, if the maximum path length is *m* and the maximum branching factor is *b*?
  - The time complexity is ? $O(b^m)$? must examine every node in the tree.
  - The order in which we examine nodes (BFS or DFS) makes no difference to the worst case: search is unconstrained by the goal.
- What is the space complexity?
  - Space complexity is ? $O(b^m)$?

Using Breadth–first Search

- When is BFS appropriate?
  - space is not a problem
  - it's necessary to find the solution with the fewest arcs
  - although all solutions may not be shallow, at least some are

optimality

- When is BFS inappropriate?
  - space is limited
  - all solutions tend to be located deep in the tree
  - the branching factor is very large

What have we done so far?

GOAL: study search**, a set of basic methods underlying many intelligent agents**

AI agents can be very complex and sophisticated

Let's start from a very simple one, **the deterministic, goal-driven agent** for which: he sequence of actions and their appropriate ordering is the solution

We have looked at two search strategies DFS and BFS:

- To understand key properties of a search strategy
- They represent the basis for more sophisticated (heuristic / intelligent) search

Learning Goals for today's class

- Apply basic properties of search algorithms: completeness, optimality, time and space complexity of search algorithms.

	comp	opt	time	space
DFS	False	False	b^m	mb
BFS	True	True	b^m	b^m

- Select the most appropriate search algorithms for specific problems.
  - BFS vs DFS vs IDS vs ~~BidirS~~ –
  - LCFS vs. BFS –
  - A* vs. B&B vs IDA* vs MBA*

next 4 lectures

informed

To test your understanding of today's class

- Work on **Practice Exercise** 3.B
- http://www.aispace.org/exercises.shtml

Next Class

- Iterative Deepening
- Search with cost

(read textbook.: 3.7.3, 3.5.3)

- (maybe) Start Heuristic Search

(textbook.: start 3.6)

Recap: Comparison of DFS and BFS

	Complete	Optimal	Time	Space
DFS	− N (Y if no cycles)	− N	+/− $O(b^m)$	+ $O(bm)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$