Uninformed Search

CPSC 322 Lecture 5

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.

Generic 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 };

while *frontier* is not empty:

select and **remove** path ⟨$n_0, n_1, \ldots, n_k$⟩ from *frontier*;

if *goal*(n_k)

return ⟨n_k⟩ ;

for every neighbor *n* of n_k

add ⟨ $n_0, n_1, \ldots, n_k$ ⟩ to *frontier*;

return NULL

Generic Search Algorithm

Inputs: a graph, a start node n_o, Boolean procedure *goal(n)* that tests if *n* is a goal node

frontier:= [$<n_o>$: n_o is a start node];

While *frontier* is not empty:

  select and **remove** path $<n_o, \ldots, n_k>$ from *frontier;*

  If *goal*(n_k)

    return $<n_o, \ldots, n_k>$;

  For every neighbor *n* of n_k

    add $<n_o, \ldots, n_k, n>$ to *frontier;*

return NULL

- The *goal* function defines what constitutes a solution
- The *neighbor* relationship defines the graph
- How paths are selected from the frontier defines the search strategy
- The order in which paths are added to the frontier is not specified

Learning Goals for this class

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

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

in upcoming lectures

Lecture Overview

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

Comparing Searching Algorithms: will it find *any* solution? the *best* solution?

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 returns a solution, it is the best solution (i.e. there is no better 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 paths*)

- 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 the path most recently added to the frontier.

Example:

top of stack

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

- What happens?
  - p_1 is selected, and its end is tested for being a goal. If it is not…
  - New paths are created attaching $\{n_1, \ldots, n_k\}$ *to* p_1
  - These "replace" p_1 at the beginning of the frontier.
  - Thus, the frontier is now $[(p_1, n_1), \ldots, (p_1, n_k), p_2, \ldots, p_r]$.
  - *NOTE:* p_2 is only selected when all paths extending p_1 have been explored.

Analysis of DFS

Def. : A search 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? A. Yes B. 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, go back to “Solve” and see what happens

Analysis of DFS

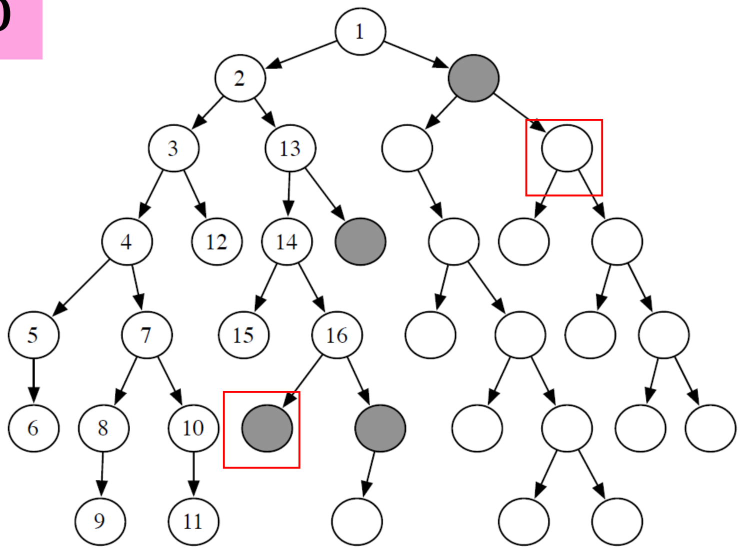
Def.: A search algorithm is optimal if when it finds a solution, it is the best one (e.g., the shortest)

Is DFS optimal?

A. Yes

B. No

goal nodes are red boxes



- It can “stumble” onto longer solution paths before it gets to shorter ones.

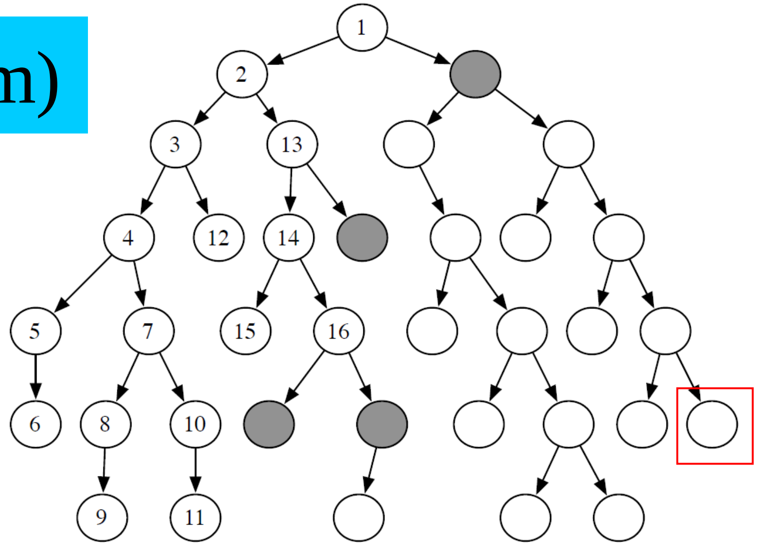
- see this in AISpace by loading “Extended Tree Graph” and set N6 as a goal
  - e.g., click on “Create” tab, right-click on N6 and select “set as a goal node”

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 ?

$O(b^m)$	$O(m^b)$	$O(bm)$	$O(b+m)$
A	B	C	D



- In the worst case, must examine every node in the tree

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.

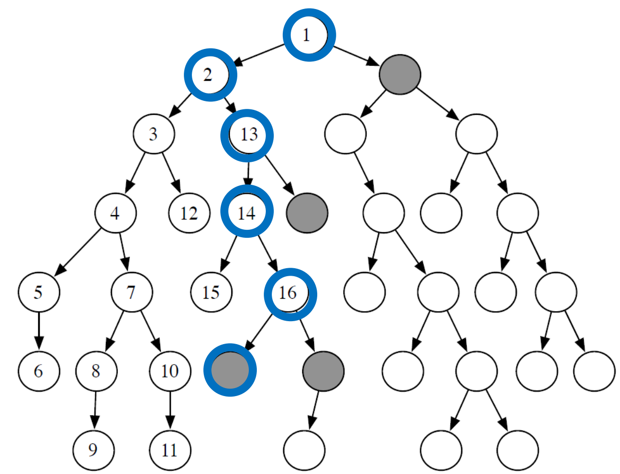
- What is DFS's space complexity, in terms of m and b ?

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

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*.



- 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: Analysis of DFS Summary

- Is DFS complete? ______
  - May not halt on graphs with cycles.
  - However, DFS ***is*** complete for finite acyclic graphs.
- Is DFS optimal? ______
  - It may stumble on a suboptimal solution first
- What is the time complexity, if the maximum path length is *m* and the maximum branching factor is *b* ?
  - Time complexity is _____: may need to examine every node in the tree.
- What is the space complexity?
  - Space complexity is _____: the longest possible path is *m*, and for every node in that path we must maintain a "fringe" of size *b*.

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
E. You have to hand in your code in 5 minutes

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

Inappropriate

- Cycles
- There are shallow solutions
- If you care about optimality

Why bother studying/understanding DFS at all?

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

- 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:

front of queue end of queue

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

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

Breadth-first Search: Analysis of BFS

- Is BFS complete?
- Is BFS optimal?

Breadth-first Search: Analysis of BFS

i>clicker.

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

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

i>clicker.

- What is the space complexity?

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

Analysis of Breadth-First Search

- Is BFS complete?
  - Does not get stuck in cycles
- Is BFS optimal?
  - 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 _______: may need to examine every node in the tree
- What is the space complexity?
  - Space complexity is ________: frontier contains all paths of the relevant length (which is <= the shortest path length to a goal node)

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

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

When to use BFS vs. DFS?

1. The search graph has cycles or is infinite

   BFS DFS

2. We need the shortest path to a solution

   BFS DFS

3. There are only solutions at great depth

   BFS DFS

4. There are some solutions at shallow depth

   BFS DFS

5. Memory is limited

   BFS DFS

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

We started from a very simple one, **the deterministic, goal-driven agent** for which: the sequence of actions and their appropriate ordering is the solution

We have looked at two search strategies (DFS & BFS):

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

Search Summary Table

	complete?	optimal?	time O( )	space O( )
DFS	False	False	b^m	mb
BFS	True	True*	b^m	b^m

*Assuming arcs all have the same cost (we'll get to this later)

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 costs

(read textbook.: 3.7.3, 3.5.3)