CS 232: Artificial Intelligence

Fall 2023

Prof. Carolyn Anderson
Wellesley College

Reminders

- I have help hours Monday from 4-5:30pm
- Lepei's help hours: Sundays 6-8pm
- Lyra's help hours: Wednesdays 2-4pm
- Reading for next Tuesday: YLLATAILY 3-4

Recap

Defining A Search Problem

States: a representation of physical configuration

Nodes: a data structure representing:

<state, parent-node, children, action, path-cost, depth>

Goal: the state(s) we're trying to reach

Start State: initial starting point

Solution: a sequence of states that take us from the start state to the goal state.

Optimal Solution: the shortest solution

Graph Search vs Tree Search

function TREE-SEARCH(*problem*) **returns** a solution, or failure
initialize the frontier using the initial state of *problem*
loop do
if the frontier is empty **then return** failure
choose a leaf nose and remove it from the frontier
if the node contains a goal state **then return** the corresponding solution
expand the chosen node, adding the resulting nodes to the frontier

function GRAPH-SEARCH(*problem*) returns a solution, or failure
initialize the frontier using the initial state of *problem*
initialize the explored set to be empty
loop do
if the frontier is empty **then return** failure
choose a leaf node and remove it from the frontier
if the node contains a goal state **then return** the corresponding solution
add node to the explored set
expand the chosen node, adding the resulting nodes to the frontier
only if not in the frontier OR ***explored set***

Slides adapted from Chris Callison-Burch

Search Strategies

Review: ***Strategy*** = order of tree expansion

- Implemented by different queue structures (LIFO, FIFO, priority)

Dimensions for evaluation

- ***Completeness*** - always find the solution?
- ***Optimality*** - finds a least cost solution (lowest path cost) first?
- ***Time complexity*** - # of nodes generated *(worst case)*
- ***Space complexity*** **- # of nodes simultaneously in memory** *(worst case)*

Time/space complexity variables

- *b*, *maximum branching factor* of search tree
- *d*, *depth* of the shallowest goal node
- *m*, maximum length of any path in the state space (potentially ∞)

Slides adapted from Chris Callison-Burch

Uninformed Search

Uses only information available in problem definition

Informally:

Uninformed search: All non-goal nodes in frontier look equally good

Informed search: Some non-goal nodes can be ranked above others.

Breadth-first search

Idea:

- Expand *shallowest* unexpanded node

<u>Implementation:</u>

- *frontier* is FIFO (First-In-First-Out) Queue:
  - Put successors at the *end* of *frontier* successor list.

Properties of breadth-first search

Complete? Yes (if d is finite)

Optimal? Yes! (assuming we measure cost as # of steps)

Time Complexity? $O(b^d)$

Space Complexity? $O(b^d)$

Exponential Space (and time) Is Not Good...

- Exponential complexity uninformed search problems *cannot* be solved for any but the smallest instances.
- *(Memory* requirements are a bigger problem than *execution* time.)

DEPTH	NODES	TIME	MEMORY
2	*110*	*0.11 milliseconds*	*107 kilobytes*
4	*11110*	*11 milliseconds*	*10.6 megabytes*
6	10^{6}	*1.1 seconds*	*1 gigabytes*
8	10^{8}	*2 minutes*	*103 gigabytes*
10	10^{10}	*3 hours*	*10 terabytes*
12	10^{12}	*13 days*	*1 petabytes*
14	10^{14}	*3.5 years*	*99 petabytles*

Assumes b=10, 1M nodes/sec, 1000 bytes/node

Depth-First Search

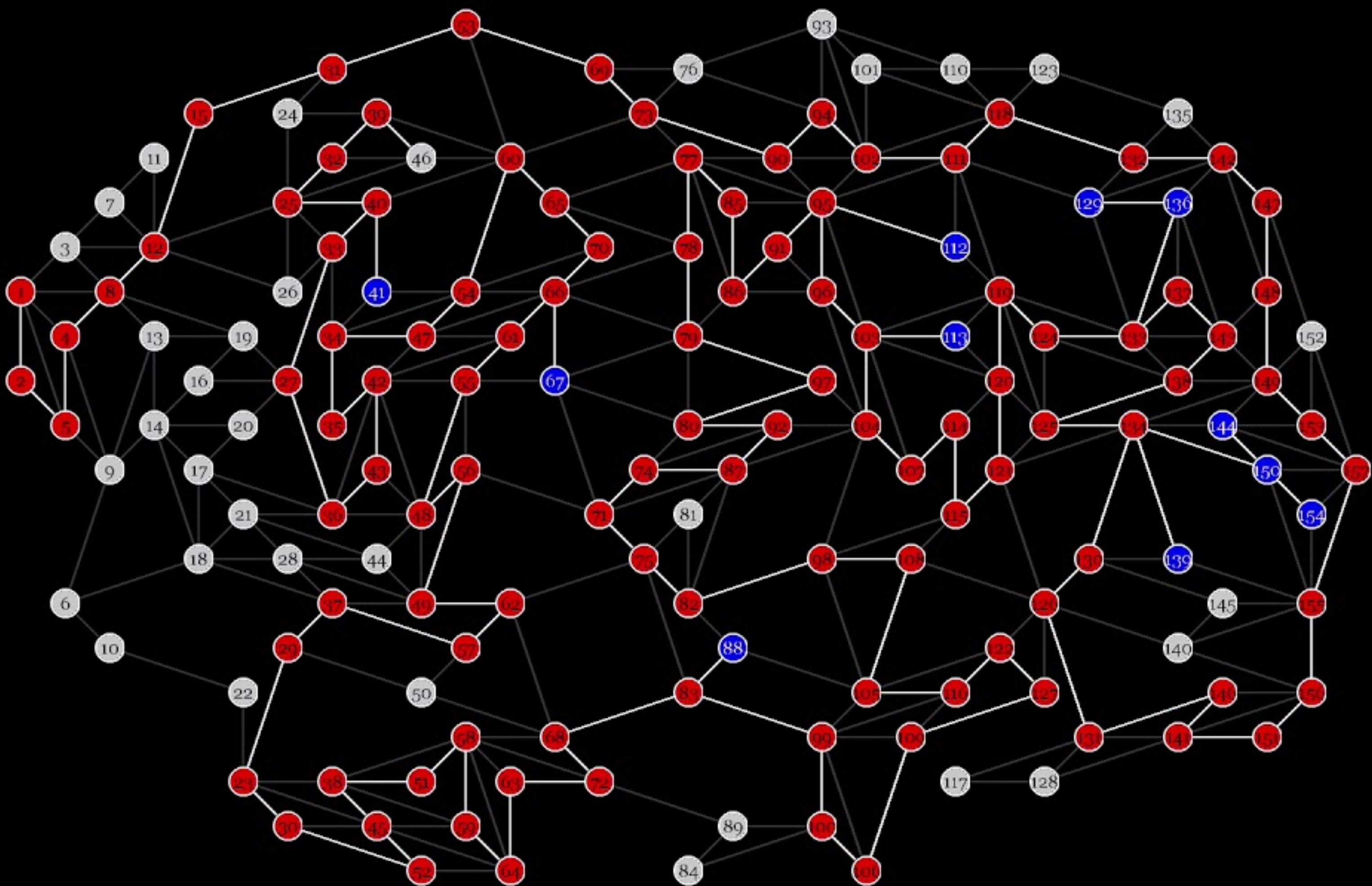
Depth-first search

Idea:

- Expand *deepest* unexpanded node

Implementation:

- *frontier* is LIFO (Last-In-First-Out) Queue:
  - Put successors at the *front* of *frontier* successor list.



current x

discovered y

node done

Undiscovered edge

Discovered edge

```
x = start vertex(1)
dfs(x)

def dfs(x):
   mark x as visited
   for each y in x connections:
      if y not visited then
         dfs(y)
```

Please subscribe @youtube.com/gjenkinslbcc or with icon in lower right >>>

Properties of depth-first search

Complete? Yes if tree is finite

Optimal? No

Time Complexity? $O(b^m)$

Space Complexity? $O(b*m)$

m = maximum depth of search space

Depth-first vs Breadth-first

Use depth-first if

- *Space is restricted*
- There are many possible solutions with long paths and wrong paths are usually terminated quickly
- Search can be fine-tuned quickly

Use breadth-first if

- *Possible infinite paths*
- Some solutions have short paths
- Can quickly discard unlikely paths

Search Conundrum

Breadth-first

- ☑ Complete,
- ☑ Optimal
- ☒ *but* uses $O(b^d)$ space

Depth-first

- ☒ Not complete *unless m is bounded*
- ☒ Not optimal
- ☒ Uses $O(b^m)$ time; terrible if m >> d
- ☑ *but* only uses *O(**b*m) space***

Depth-limited search: A building block

Depth-First search *but with depth limit l.*

- i.e. nodes at depth l *have no successors.*
- No infinite-path problem!

If $l = d$ (by luck!), then optimal

- But:
  - **If $l < d$ then incomplete** ☹
  - **If $l > d$ then not optimal** ☹

Time complexity: $O(b^l)$

Space complexity: $O(bl)$ ☺

Summary of algorithms

Criterion	Breadth-First	Depth-First	Depth-limited	Iterative deepening
Complete?	YES	NO	NO	YES
Time	b^d	b^m	b^l	b^d
Space	b^d	bm	bl	bd
Optimal?	YES	NO	NO	YES

Slides adapted from Chris Callison-Burch