Artificial Intelligence

Week 3: Uninformed Search

COMP30024

April 1, 2021

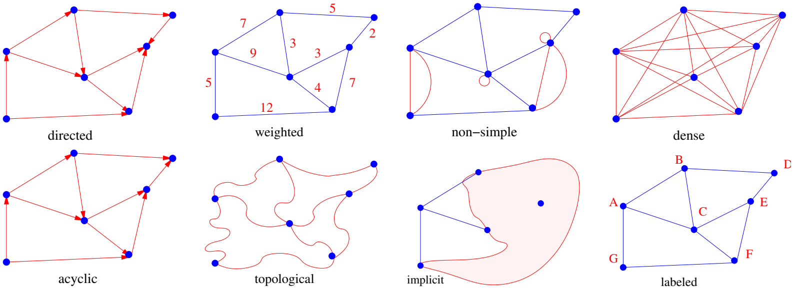
Search

Search

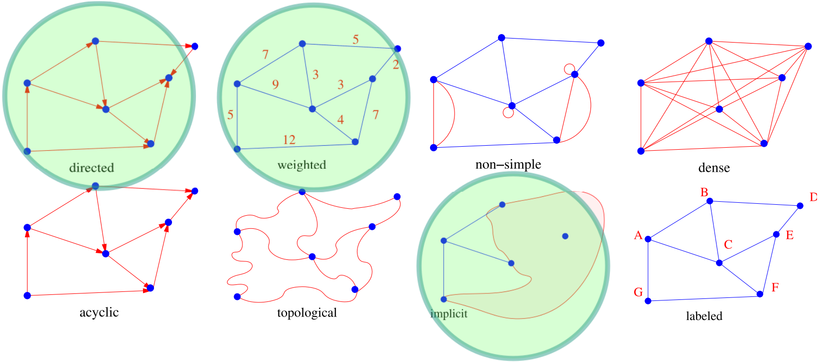
Select action sequences to achieve a goal state in environments that are deterministic, observable, static with a completely specified state space.

- Uninformed search (today): Only given problem statement.
  - RN 3.5.
- Informed search (next week): Incorporate domain knowledge to strive for optimality.
  - RN 4.*.

Graphs


directed
7
5
2
3
9
3
5
7
4
12
weighted
non−simple
dense
acyclic
topological
implicit
B
D
A
E
C
G
F
labeled

Graphs



Game-playing: built as we use them - not explicitly constructed and traversed.
`https://github.com/aimacode/aima-python/blob/master/search.py`

Breadth-First Search

Expand shallowest unexpanded node - FIFO queue for frontier (set of unexpanded visited nodes).

- Completeness ✔
- Optimality ✗ (Only if $g(\sigma) = g(d)$ is non-decreasing)
- b : Branching factor, d : goal depth
- Time complexity (generated states) $\mathcal{O}(b^d) = O(|V|)$
- Space complexity (cached states) $\mathcal{O}(b^d) = O(|V|)$

Breadth-First Search

BFS Pseudo-code

```
def BFS(graph, start, end):
    visited = [start]
    frontier = [start]

    while frontier is not empty:
        node = frontier.pop_first()
        if node == end:
            return (node, True)
        for child in graph[node]:  # check against hash table
            if child not in visited:
                visited.append(child)
                frontier.push_back(child)  # add new candidate
    return (None, False)
```

Depth-First Search

DFS Pseudo-code

```
def DFS(graph, node, visited, end):
    if node == end:
        return True
    if node not in visited:
        visited.append(node)
        for child in graph[node]:
            dfs(graph, child, visited, end)
    return False
```

- Expand most recently generated node (deepest unexpanded visited node).
- Completeness ✗
- Optimality ✗ - pick a direction and hope.
- Time complexity $\mathcal{O}(b^d)$
- Space complexity (cached states) $\mathcal{O}(bd)$ - cache single path from root to node (+ unexpanded siblings).

Iterative Deepening Search

IDS Pseudo-code

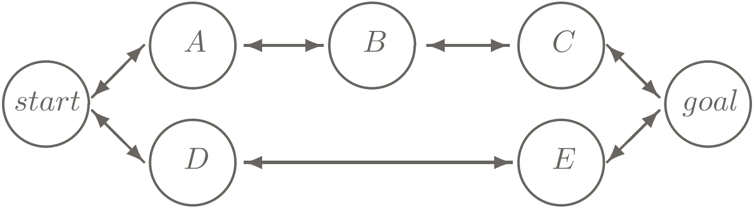
```
def IDS(graph, node, visited, end):
    for d in range(max_depth):
        result = # DFS with recursion limit d
    return result
```

- Iteratively increase depth limit of DFS until goal is found
- Completeness ✔
- Optimality ✔ (if path cost is nondecreasing function of depth).
- Space complexity (cached states): $\mathcal{O}(bd)$
- Time complexity:

$$N(IDS) = db + (d-1)b^2 + \ldots 2b^{d-1} + b^d \in \mathcal{O}(b^d)$$

Bidirectional Search

- Identify goal state(s) and start state, search simultaneously forward and backward.
- Terminate upon forward/backward frontier intersection.
- Caveats:
  - Must know goal states explicitly beforehand.
  - Must efficiently generate predecessor states.
- Time complexity $\mathcal{O}(b^{d/2})$ - exponential savings v. BFS/DFS!
- Space complexity - Q2.4.
- Completeness, optimality depend on type of search algorithm used.



$start$
A
B
C
D
E
$goal$