CPSC 322
Week II

Jeremi Do Dinh - 61985628

April 12, 2020

Lecture IV

Comparing Searching Algorithms

Def 1. 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 2. A search algorithm is **optimal** if, when it returns a solution, it is the best solution (i.e. there is no better solution)

Def 3. 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 4. 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.

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 1.
- *The frontier is* $[p_1, p_2, ..., p_r]$
- *neighbors of last node of* p_1 *(its end) are* $\{n_1, ..., n_k\}$
What happens:
- p_1 *is selected, and its end is tested for being a goal. If it is not...*
- *New paths are created attaching* $\{n_1, ..., n_k\}$ *to* p_1
- *These "replace"* p_1 *at the beginning of the frontier.*
- *Thus, the frontier is now* $[(p_1, n_1), ..., (p_1, n_k), p_2, ..., p_r]$ *.*
- *NOTE:* p_2 *is only selected when all paths extending* p_1 *have been explored.*

Analysis of DFS Summary

- Is DFS complete? **No**
  - May not halt on graphs with cycles.
  - However, DFS is complete for finite acyclic graphs.
- Is DFS optimal? No

  - 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 $O(b^m)$: may need to examine every node in the tree.
- What is the space complexity?
  - Space complexity is $O(bm)$: the longest possible path is m, and for every node in that path we must maintain a "fringe" of size b.

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

Breadth-first Search: BFS

Breadth-first search treats the frontier as a queue.

Example 2.

- *the frontier is* $[p_1, p_2, ..., p_r]$
- *neighbors of the last node of p1 are* $\{n_1, ..., n_k\}$

What happens?

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

Analysis of BFS Summary

- Is BFS complete? **Yes**
  - Does not get stuck in cycles
- Is BFS optimal? **Yes**
  - 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 ?
  - Time complexity is $O(b^m)$: may need to examine every node in the tree.
- What is the space complexity?
  - Space complexity is $O(b^m)$: frontier contains all paths of the relevant length (which is $\leq$ to the shortest path length to a goal node)

When it is appropriate?

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

Inappropriate

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

Iterative Deepening Search

Essence:

- Look with **DFS** for solutions at depth 1, then 2, then 3, etc.
- If a solution cannot be found at depth D, look for a solution at depth $D + 1$.
- You need a depth-bounded depth-first searcher.
- Given a bound B you simply assume that paths of length B cannot be expanded....

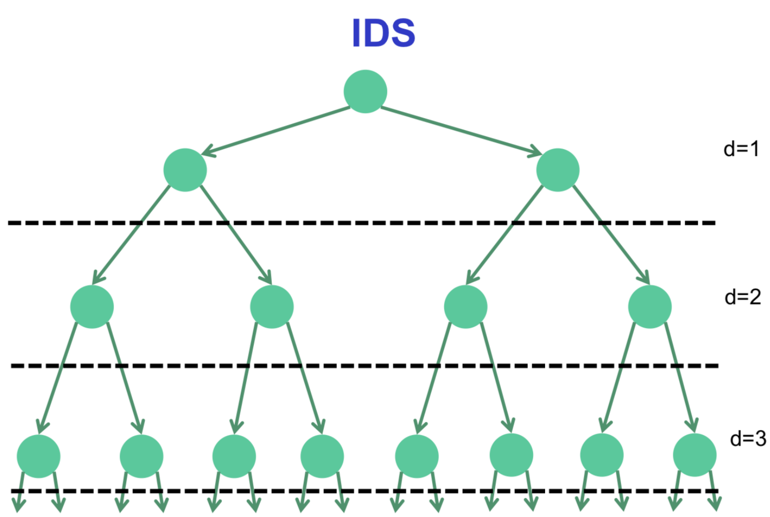

Figure 1: IDS essence