Homework 1

Maxwell Pung
CSCI4511W Artificial Intelligence,
University of Minnesota
Minneapolis, Minnesota

February 2, 2017

Depth-first search = DFS
Breadth-first search = BFS
Limited depth-first search = LDFS
Iteratively deepening depth-first search = IDDFS
Uniform cost search = UCS

1.1 The **initial state** of this agent would be a full dictionary of words to choose from and a crossword puzzle with all empty slots. The **goal test** of this agent says that if all horizontal and vertical slots are populated with letters, then the puzzle has been solved. The **states** of this agent include...

- Slot empty
- Slot has letter

The **actions** of this agent include...

- Fill in word corresponding to slot
- Remove word intersected with slot
- Skip slot

The **cost of actions** for this agent include...

- Number of letters in word
- No cost for skipping

1.2 The state space is a directed acyclic graph. This is because from the 'Slot has letter' state, the next state to be encountered could be 'Slot empty' or 'Slot has letter', hence forming an directed acyclic graph.

2.1 DFS is suitable for applications that require an entire tree to be selected since it requires less state. DFS is not suitable for applications that only need to partially search a tree because it may go to unnecessary depths.

2.2 BFS is suitable for applications that need to find the shortest path from a starting node to a given node since it yields an optimal solution. BFS is not practical for applications with solutions that occur frequently, deeply in a tree.

2.3 UCS is useful for applications who's decisions have varyting costs. It is less useful for memory intensive applications since each level of the tree must be saved in memory to get to the next.

2.4 LDFS is suitable for applications whose solutions are likely to be found near the root of the tree, it could potentially make a better decision than regular DFS. It is not suitable for applications where we are entirely clueless about how deep our solutions are going to occur, otherwise we may not find it if it is deeper than the limit.

2.5 IDDFS is excellent for the same applications as BFS but can offer better space efficiency. It is less suitable for applications where it is a significant disadvantage to explore the same path multiple times.

3.1 Algorithms that limit memory requirements include DFS, LDFS, and IDDFS. I would choose these algorithms because they require space in memory of size O(h) where h is the height of the given tree. The other algorithms like UCS and BFS require space in memory of size O(w) where w is the width of the given tree.

3.2 Limiting the time it takes to search a tree really depends on the characteristics of the given tree such as it's structure and location of solutions. BFS would be suitable for applications that are looking for solutions that are expected to be close to root of the tree. In such a situation, BFS performs better on average that DFS or one of its variants. DFS would limit the time it takes to find a solution if the solution is going to be found deeper in the tree. If it's possible that there is a cycle in the graph being represented as a tree and solutions are expected to be deep in the tree, IDDFS would be the best choice since DFS could potentially run infinitely.

3.3 The algorithm chosen is dependent on whether actions/edges have weights or not. If they have weights, UCS would be the algorithm of choice because at each step it chooses the choice with the lowest cost. If the choices do not have weights, BFS or IDDFS would be the algorithms of choice because they both have better worst-cose performance than UC.

3.4 The algorithm of choice here would be IDDFS. This algorithm requires less many than BFS but is still guranteed to find the optimal solution.

3.5 I would use IDDFS. This is because different branches could be searched concurrently (unlike BFS). Also, it would be able to handle infinite cycles (unlike DFS). It would recieve the most noticeable performance boost from such parallelism.

4.1 The optimal solution would not change if a graphs costs were all increased by 10. This is because the agent searching would not care about the initial value of nodes and since all costs are changing by the same factor, the optimal solution will be the same but with a value of 10 greater than it's original value

4.2 The optimal solution would not be subject to change if costs were doubled. Such a change would not effect the agents because the change being done across all nodes is a linear change.

5 A finite state space does not always lead to a finite search tree. If a finite state space contains cycles in it's graph representation, then the corresponding search tree could be traversed infinitely if the right search algorithm is not used. If the finite state space is a tree, then it will not contain any cycles and it will therefore lead to a finite search tree. State spaces that do not contain cycles in their graph representation will gurantee the search tree to be finite.

6 If the closed list is not used in an algorithm that it should be used by, then paths could be explored redundantly since the algorithm forgot the nodes it has visited. For DFS and LDFS, without a closed list they would re-explore paths every time that they backtrack. For BFS, without a closed list it would re-explore the paths found at previous depths of the tree everytime it moves on to a new depth. For UCS, without a closed list the algorithm would re-explore paths whenever a new optimal solution is found. For IDDFS, without a closed list the algorithm would re-explore paths of previous depths on every deepening iteration.