Using DFS, BFS and UCS for Sokoban

Truong Minh Chau – 19521281

1. Modeling Sokoban

Sokoban is a logic puzzle which can be modeled as a maze-path-finding problem or as a search problem, so we can use some algorithms such as DFS, BFS, UCS and single-agent search to solve. But here all three DFS, BFS and UCS were chosen for finding the solution the Sokoban. The search problem consists of:

- A state space
- A start state
- A goal state
- A set of legal actions
- A successor function

1.1 State space

A state space is a set of all states reachable from the initial state of a problem. In Sokoban, state space defined is a tuple of all possible position of both box and player.

1.2 Start state

A start state is the initial position of both boxes and player. We can pick randomly a state from state space to make it be a start space.

1.3 Goal state

A goal state is a state which boxes are at target positions and often specified as a Boolean function to output logical (true, false) values for current states.

1.4 A set of legal actions

This is a set of actions taken from current state which not repeat the state has been recorded. A legal action is changing the position of player to the left, right, up or down and it is considered as an illegal action if the player move to wall position or pull boxes to wall position.

1.5 A successor function

A successor function is a function that generates a next state from the current state, plus preference choices depending on each algorithm that affect state changes.

2. Using DFS, BFS and UCS for Sokoban

2.1 DFS

This algorithm outputs a solution which is very long and not optimal. Because it starts at the root node and explores the deepest node first and return the solution. So when we run Sokoban using DFS, we can easily see the path of player is longer than we expected and have to wait for a long time to get the solution.

2.2 BFS

This algorithm is optimal in term of finding the solution which is a shortest path as possible. It takes every legal actions from current state, that is to say it explores neighboring nodes in the branches in turn and stop if found the goal. In Sokoban, the path after reaching the destination is very short so we can also say it is an optimal result. In some levels it takes us few minutes to get the solution although there are only simple maps because of an increasing number of states discovered to the goal.

2.3 UCS

This is also a optimal algorithm in term of returning solution has the least cost to goal. Continue to Sokoban, it is clear to see that both solutions of BFS and UCS are identical because of the same step cost. But sometimes UCS can help it run faster due to expanding nodes based on cost.

2.4 Comparision

After running Sokoban, I have some comments:

- Output of both BFS and UCS is an optimal solution while the opposite was true for DFS.
- As regards time, UCS seems a bit faster than BFS but in most levels that difference is negligible. DFS could be fast or slow up to the problem and node chose first.
- About space complexity will be sorted in descending order from DFS to UCS and finally to BFS. By that I mean DFS is an algorithm uses the least

amount of space for storage during search while BFS experienced an opposite pattern.

- All above three algorithms, they can complete the mission and then give solution if it exist.

In conclusion, according to my view, UCS is the best of three algorithms for running Sokoban.