

Application of DFS, BFS, UCS for Sokoban Game

Quan Hoang Ngoc - 22521178

3/2024

In this project, we install search algorithms such as DFS, BFS, UCS to apply them to solve the sokoban game by modeling them as a search problem. We also conduct experiments and statistics to compare algorithms as well as evaluate the difficulty of the game.

A. Application of DFS, BFS, UCS for Sokoban Game

- To understand the application of search algorithms (DFS, BFS, UCS) for the Sokoban Game, we need to understand the structure of programs.
- Whenever the function `game.py/Game/auto_move` (or user click 'auto move' mode) then the function `solver.py/get_move` will be activated and call one of three algorithms include **DFS**, **BFS** and **UCS** to search a solution which lead to a goal state *to win the game*.

B. Modeling Sokoban Game

- Solving a Sokoban task can be modeling as a **search problem** which we need to find a solution (path = sequence of actions including 'up', 'down', 'left', 'right') so that from initialize state of game we can go to the end-state that wins the game.
- Thus, we can modeling the Sokoban World by some concepts such as:
 - **State**: information about position of the player and position of boxes.
 - **Start state**: the state when we perform one of three functions (DFS, BFS, UCS). Typically, it is when we start a level of game.
 - **End state**: the state where we win the game (all boxes are in the correct position).
 - **State Space**: all states that can be reached from the start state. It is the totality of possible positions of player and boxes. It is frequently represented as a graph.
 - **Action**: 'up', 'down', 'left', 'right' correspond to moving the player up, down, left and right.
 - **Legal Action**: an action is called 'Legal' if it makes the function `solver.py/isLegalAction` true. In other words, it does not cause the player to move into walls or boxes. Beside, in algorithms, an action only be considered if it is not *Failed*. It does not cause a box that is not in the correct position but it can not move anymore.
 - **Successor function**: in this setting code, it is function `solver.py/legalAction`. A successor function describes the agent's options: given a state, it returns a set of (action, next state) pairs. it indicates actions and states corresponding can happen to an input state.

C. Statistics and Evaluation

- Let's evaluate the performance of algorithms, we need to measure some quantity such as: length of solution (number of actions that need to be performed) - L, number of nodes (state) that can be processed by algorithm - N.

Statistics Algorithms Table-[1]

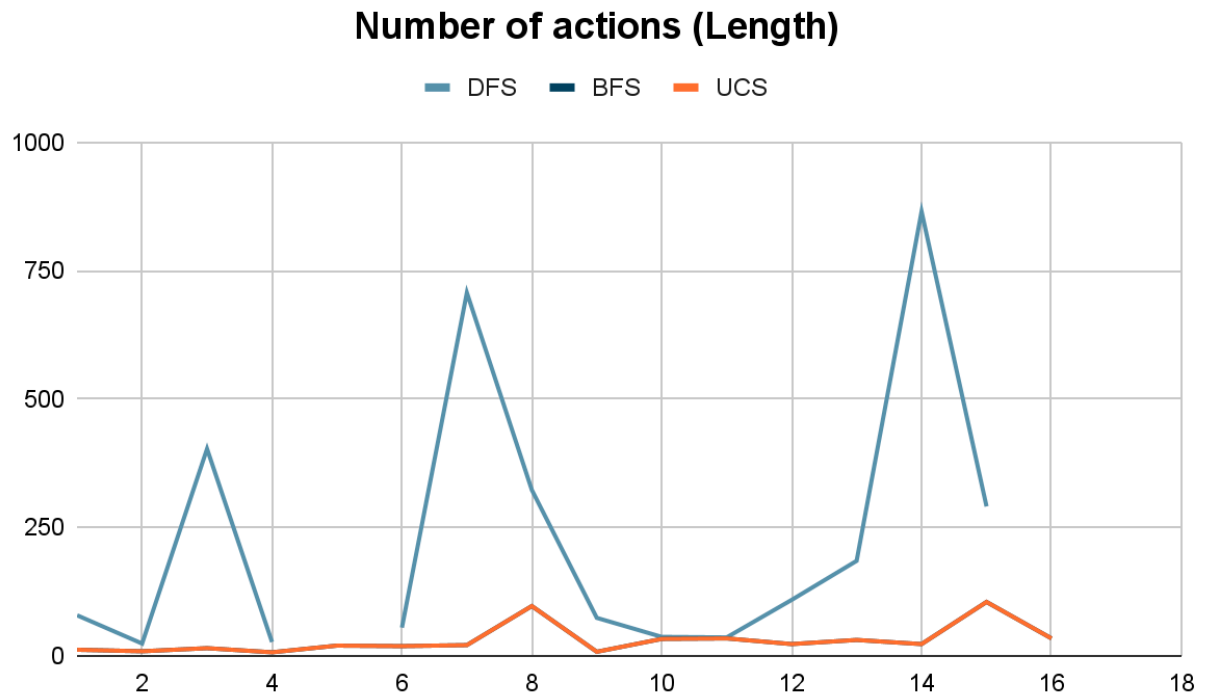
Level of game	DFS		BFS		UCS	
	L	N	L	N	L	N
1	79	2177	12	2552	12	1634
2	24	<u>55</u>	9	186	9	181
3	403	4582	15	2137	15	1218
4	27	<u>41</u>	7	230	7	171
5		OOM	20	2253288	20	1278998
6	55	<u>524</u>	19	612	19	595
7	707	<u>16221</u>	21	27307	21	16767
8	323	<u>1756</u>	97	5561	97	5356
9	74	7142	8	141	8	174
10	37	<u>392</u>	33	<u>509</u>	33	524
11	36	<u>638</u>	34	671	34	657
12	109	5116	23	2959	23	2966
13	185	6341	31	4905	31	6011
14	865	88800	23	65221	23	69842
15	291	<u>3842</u>	105	6362	105	6325
16		OOM	34	160839	34	141814
17	x	169158	x	169158	x	169158
18		OOM		OOM		OOM

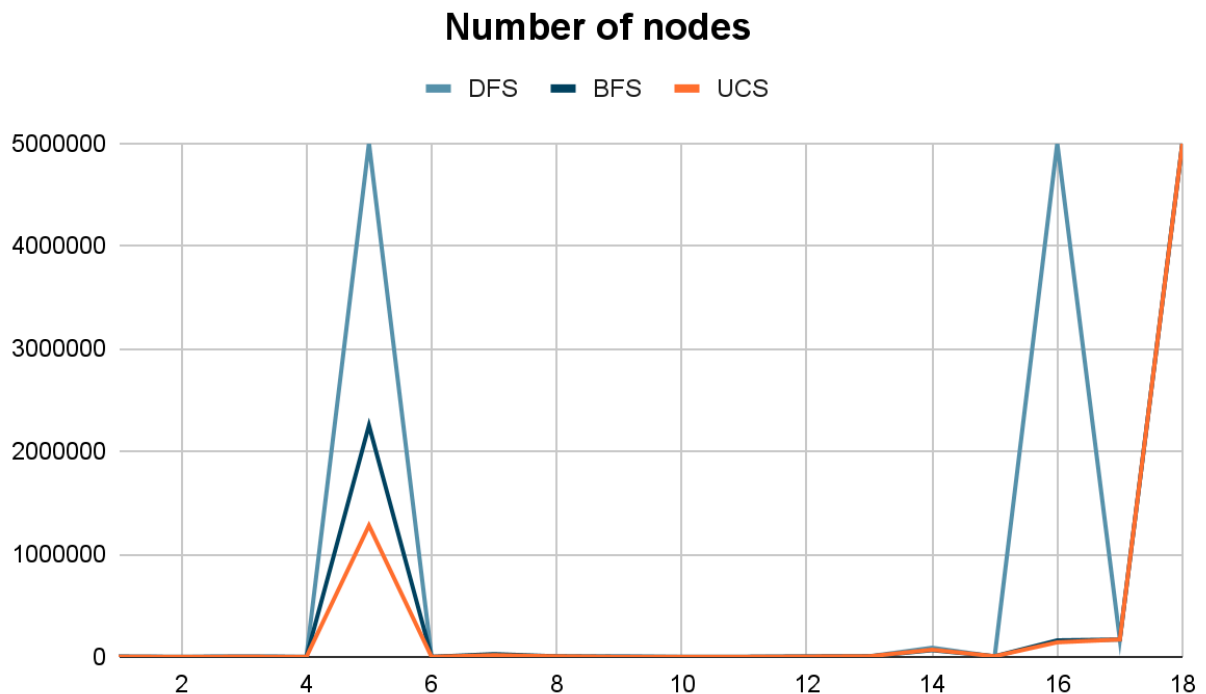
Notion:

[x]: not find a solution

[OOM]: Out of Memory

Computer used for statistics is **Kaggle Colab P100: RAM 29GB.**





1. Evaluate about Length (number of actions)

- In these maps of the Sokoban Game, both **BFS and UCS** find the **shortest path** (minimum number of actions) if there is a solution. But typically, UCS will search for the solution in a shorter time, because it processes fewer nodes.
- **DFS does not find a shortest path**, the path can be very long. But, in some maps that have many solutions to win and length of solution is not deep, it can be effective.

2. Evaluate about Performance of Algorithms

- We expect an algorithm to find the exact solution (shortest path) and the execution time is minimal. The accuracy of the solution is expressed by the length of the solution and the execution time is expressed by the number of nodes that can be processed by the algorithm.
- We define a quantity as the goodness of the algorithm as follows: $G = L + 2N$
- Because, we expect to find the solution faster, so the weight of N will be larger than L .
- But L and N are two different quantities (they are not in the same range) so we need to normalize them:
 - $L = (L - 7)/(865 - 7)$
 - $N = (N - 41)/(2253288 - 41)$
 - If it OOM, $G = 3$.

[Comparison Table-\[2\]](#)

	DFS	BFS	UCS	
1	0.085812015	0.008056289	0.005791188	0.099659492

2	0.019825946	0.002459705	0.002455267	0.024740919
3	0.465569089	0.011184436	0.010368724	0.487122249
4	0.023310023	0.000167758	0.000115389	0.02359317
5	3	2.015151515	1.150363944	6.165515459
6	0.056372771	0.014492838	0.014477749	0.085343358
7	0.830212313	0.040518535	0.031163147	0.901893995
8	0.369820616	0.109794701	0.109612741	0.589228058
9	0.084391482	0.001254262	0.001283553	0.086929297
10	3	0.030718431	0.030731745	3.061450176
11	0.034329436	0.032027724	0.032015298	0.098372458
12	0.123385729	0.021238059	0.021244272	0.16586806
13	0.213051138	0.032289353	0.033271048	0.278611538
14	1.078783196	0.076502306	0.080603943	1.235889445
15	3	0.119829684	0.119796843	3.239626527
16	3	0.174194118	0.157307376	3.331501495
17	1.150109597	1.150109597	1.150109597	3.450328792
18	3	3	3	9
	19.53497335	6.839989313	5.950711825	

- If G is larger, it shows that the algorithm is not good, so the best algorithm is the algorithm with the smallest total G. In other words, **UCS is the best algorithm in these cases.**

3. Evaluate about Game Difficulty

- The map that is most difficult for a computer to solve will be the one that takes the longest to solve.
- The larger the number of nodes an algorithm needs to process, the longer it will take to run the algorithm.
- So, following figure 2, **level 18 is most difficult.** It causes all 3 algorithms to suffer Out of Memory (29 GB).
- But, if we ignore the OOM case, **maybe that level 5 is most difficult.**

D. Conclusion & Discussion

- We can model Sokoban World by a search problem and apply algorithms such as DFS, BFS, UCS to search for a solution (a path) that can win the game.
- In these cases, UCS appears to be the better algorithm.
- There are many different ways to define the difficulty of the game, but if measured in terms of time, it seems that levels 18 and 5 are the most difficult.

- In the source code of this project, we use Python Programming Language to implement the algorithms, but the run time of programs is too long and slow (up hours). Instead of, you can use C++ as the backend of algorithms, it will significantly improve speed of algorithms based on data structure and control structure, thus reducing the time and resources needed to experiment.

E. References

- All about source code, experiment process documentation and youtube demo (if there) in this github link:
https://github.com/QuanHoangNgoc/Application_DFS_BFS_UCS_for_Sokoban_Game
- Author: Quan Hoang Ngoc - 22521178. Thank you.