Lab Assignment 2
State Space Mode

T-622-ARTI,Artificial Intelligence, 2023-1

Reykjavik University - School of Computer Science, Menntavegi 1, IS-101 Reykjavík, Iceland

Jonas Hagg `jonash22@ru.is`
Philipp Zimmermann `philipp22@ru.is`
Jonathan Lutz `jonathan22@ru.is`

January 20, 2023

Abstract

This is the report for the second lab assignment of the artificial intelligence course.

1 Tasks

1.1 Task 1

1. Depth-First benefits from detecting repeated states because the search becomes faster if the same state is not queued multiple times. If the graph is cyclic and the search algorithm does not detect repeated states, the Depth-First search does not terminate.
2. Breadth-first search benefits from detecting repeated states because repeated states do not have to be stored in the queue, thus reducing the required memory. They also do not have to be processed, reducing the processing time required.
3. Uniform-Cost search can not benefit from detecting repeated states, because we must visit a state multiple times if multiple paths lead to this state. Otherwise, we could not compare the different paths by there cost and we could therefore not determine which path we should prefer.

1.2 Task 2

A commonly used data structure for detecting repeated states in search algorithms is a hash map, as it provides efficient operations for adding and checking for the presence of elements. We can use this hash map to store all visited states. This allows the algorithm to quickly check if a state has already been visited, without the need to search through all previously visited states.

1.3 Task 3

The state consists of the following data:

1. The current orientation of the vacuum.
2. The current position of the vacuum (in terms of coordinates).
3. A list of cells which contain dust (i.e. a list of coordinates).
4. A flag telling if the vacuum is turned on.

Legal transitions are:

1. Turn on is the only legal transition if the agent is turned off
2. Turn off is legal only if the agent is turned on
3. If the agent is turned on, *TURN_LEFT*, *TURN_RIGHT*, *GO* and *SUCK* are legal.

1.4 Task 4

The agent can be in $H \cdot W$ cells with every orientation ($H \cdot W \cdot 4$). Besides, $\sum_{n=0}^{D} \frac{!(HW)}{!(HW-n)}$ states exist for the dirty cells where D is the number of dirty cells at the beginning. This includes all possible combinations from a dirty room to a already cleaned room. Since the dirt can also be sucked, we compute the sum for up to D dirty spots. Also the agent can always be turned on or off (factor of two). Thus, the upper bound of the possible states is:

$$2 \cdot H \cdot W \cdot 4 \cdot (\sum_{n=0}^{D} \frac{!(HW)}{!(HW-n)}) \tag{1}$$

Note: The actual size of the State Space of a given environment will be lower than this estimate. Reason for this is, that the number of States is reduced as soon as the dust positions are fixed. Since the Robot is only turned off when all dust has been removed, there are no States with a turned off robot and dirty cells in the room (excep the Stating State before the robot has been turned on).

1.5 Task 5

see python implementation

1.6 Task 6

see python implementation

1.7 Task 7

We observed the following results while running `python3 env_tester.py`:

1. 3205 reachable states

2. 3205 unique hashes (0.0% hash collisions) This is good because it implies that all reached states have unique hashes.

   If a State has been visited during the search, its hash value is added to a list. Occurs the same hash value again, this re-appearing State will be skipped/excluded from the Search to avoid loops or increase the performance of the search. But if two or more different States would have the same hash value it is impossible to differentiate them. When one hash value of those States gets added to the list, all the other States would be unable to reach from this point on.

   Therefore all States can be exactly identified and reached, when the hash collision is at 0%.

3. 2689 unique hash indizes (16.1% index collision). A hash index collision occurs when two or more keys have the same hash value in a hash index. When this happens, the index must use some method to handle the duplicate keys. Thus we want to have as few hash index collisions as possible.

1.8 Task 8

If hashing is used to detect repeated states we want to have few (or at best none) hash collisions. Assume we have different states A and B which are hashed to the same value. Once the search algorithm has detected state A, state B will never be detected since we already detected state A. Thus, the search algorithm will not detect the valid state B meaning its output is incomplete.