2024 Fall

CS386-02

Assignment 2: Search

Jeongyeon Lim (20816838)

Question #1.

Algorithm	Time Complexity	Space Complexity	Complete?	Optimal?
BFS	$O(b^{d+1})$	$O(b^{d+1})$	O	O
UCS	$O(b^{\lceil \frac{C}{\epsilon} \rceil})$	$O(b^{\lceil \frac{C}{\epsilon} \rceil})$	O	O (if all edge costs are non-negative.)
DFS	$O(b^n)$	$O(bn)$	X	X
DLS	$O(b^l)$ l: limited depth	$O(l)$	X	X
IDS	$O(b^d)$	$O(bd)$	O	O
A*	$O(b^d)$	$O(b^d)$	O	O (in case where the heuristic is admissible)

Question #2.

<bfs: mission_complete>

Goal found

(Location: battery

Sample Extracted?: True

Holding Sample?: False

Charged? True, 'move_to_battery')

Location: sample

Sample Extracted?: True

Holding Sample?: False

Charged? True

Location: sample

Sample Extracted?: True

Holding Sample?: False
Charged? False
Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: False
Holding Sample?: False
Charged? False
None
The number of states generated: 46
(Location: battery
Sample Extracted?: True
Holding Sample?: False
Charged? True, 'move_to_battery')

<dfs: misstion_complete>

Goal found
(Location: battery
Sample Extracted?: True
Holding Sample?: False
Charged? True, 'move_to_battery')
Location: station
Sample Extracted?: True
Holding Sample?: False
Charged? True
Location: station
Sample Extracted?: True
Holding Sample?: False
Charged? True

Location: station
Sample Extracted?: True
Holding Sample?: True
Charged? True
Location: sample
Sample Extracted?: True
Holding Sample?: True
Charged? True
Location: sample
Sample Extracted?: True
Holding Sample?: True
Charged? True
Location: sample
Sample Extracted?: True
Holding Sample?: True
Charged? False
Location: sample
Sample Extracted?: True
Holding Sample?: False
Charged? False
Location: sample
Sample Extracted?: True
Holding Sample?: False
Charged? False
Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: False
Holding Sample?: False
Charged? False

None
The number of states generated: 63
(Location: battery
Sample Extracted?: True
Holding Sample?: False
Charged? True, 'move_to_battery')

<bfs: problem decomposition>

1. move to sample

Goal found
(Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False, 'move_to_sample')
Location: station
Sample Extracted?: False
Holding Sample?: False
Charged? False
None
The number of states generated: 3

2. remove sample

Goal found
(Location: sample
Sample Extracted?: True
Holding Sample?: True
Charged? False, 'pick_up_sample')
Location: sample
Sample Extracted?: True
Holding Sample?: False
Charged? False
Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False

Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: False
Holding Sample?: False
Charged? False
None
The number of states generated: 28

3. return to charger

Goal found
(Location: battery
Sample Extracted?: True
Holding Sample?: True
Charged? False, 'move_to_battery')
Location: sample
Sample Extracted?: True
Holding Sample?: True
Charged? False
Location: sample
Sample Extracted?: True
Holding Sample?: False
Charged? False
Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: False
Holding Sample?: False

Charged? False

None

The number of states generated: 7

<dfs: problem decomposition>

1. move to sample

Goal found

(Location: sample

Sample Extracted?: False

Holding Sample?: False

Charged? False, 'move_to_sample')

Location: station

Sample Extracted?: False

Holding Sample?: False

Charged? False

Location: station

Sample Extracted?: False

Holding Sample?: False

Charged? False

None

The number of states generated: 9

2. remove sample

Goal found

(Location: sample

Sample Extracted?: True

Holding Sample?: True

Charged? False, 'pick_up_sample')

Location: sample

Sample Extracted?: True

Holding Sample?: False

Charged? False

Location: sample

Sample Extracted?: True

Holding Sample?: False

Charged? False

Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: False
Holding Sample?: False
Charged? False
None
The number of states generated: 20

3. return to charger

Goal found
(Location: battery
Sample Extracted?: True
Holding Sample?: False
Charged? False, 'move_to_battery')
Location: station
Sample Extracted?: True
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: True
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: True
Holding Sample?: True
Charged? False
Location: sample
Sample Extracted?: True
Holding Sample?: True

Charged? False
Location: sample
Sample Extracted?: True
Holding Sample?: True
Charged? False
Location: sample
Sample Extracted?: True
Holding Sample?: False
Charged? False
Location: sample
Sample Extracted?: True
Holding Sample?: False
Charged? False
Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: False
Holding Sample?: False
Charged? False
Location: station
Sample Extracted?: False
Holding Sample?: False
Charged? False
None
The number of states generated: 18

Question #3.

<A* Search>

The number of states generated: 32

<Uniform Cost Search>

The number of states generated: 32

Question #4.

Antenna 1: Frequency f2

Antenna 2: Frequency f1

Antenna 3: Frequency f3

Antenna 4: Frequency f3

Antenna 5: Frequency f2

Antenna 6: Frequency f2

Antenna 7: Frequency f1

Antenna 8: Frequency f3

Antenna 9: Frequency f1

Question #5.

a) What were the engineering advances that led to Deep Blue's success? Which of them can be transferred to other problems, and which are specific to chess?

The first advance that led to Deep Blue's success was a single-chip chess search engine. This allowed it to calculate much faster, which is essential for a chess game. However, this advance is difficult to apply to other problems since it is specifically designed for chess.

The second advancement is a massively parallel system with multiple levels of parallelism. Thanks to this system, the model can calculate many chess positions simultaneously, allowing it to evaluate all possible scenarios in a short amount of time. This technology is very useful for solving other problems because it can reduce the time required for computation, which is crucial in fields that involve a large amount of calculations.

Next, a strong emphasis on search extensions supported the success. The chess engine implemented search extensions that allowed it to explore important moves more deeply, going beyond simple search techniques. This approach enabled the discovery of more strategic moves in specific situations. The search extension techniques used in chess can also be applied to make important decisions in other games or optimization problems.

A complex evaluation function is another key reason for Deep Blue's success. This function considers various factors in chess, such as piece positioning, king safety, and piece coordination, allowing for a more nuanced analysis of specific situations. While the specific factors may differ in other problems,

the approach of using multiple variables to develop an evaluation function can be quite beneficial in those contexts as well.

Lastly, we can figure out that effective use of a Grandmaster game database is one of the reasons that led to Deep Blue's success. Massive amounts of data are used for the model. It was just about the chess, so it is unable to apply to other problems.

b) AlphaZero is compared to a number of modern game-playing programs, such as StockFish, which work similarly to Deep Blue. The paper shows that AlphaZero is able to defeat StockFish even when it is given only 1/100 of the computing time. Why is that? Please frame your answer in terms of search and the number of nodes evaluated.

There is a difference between AlphaZero and StockFish in how they search for the best plays. AlphaZero uses the Monte Carlo tree search (MCTS) method, which randomly simulates different plays and chooses the best one. In contrast, StockFish uses alpha-beta search, which can be less efficient because it looks at all possible options. AlphaZero, with MCTS, focuses on important nodes and ignores those that are less significant. This distinction is made possible by an effective evaluation function and an artificial neural network.