Noah Cavestany
Prof. Brooks
CS386

Part 1

Algorithm	Time Complexity	Space Complexity	Complete?	Optimal?
BFS	$O(b^{d+1})$, b is the branching factor and d is the depth	$O(b^{d+1})$	Yes	Yes if all actions have uniform cost
UCS	$O(b^{c/e})$	$O(b^{c/e})$	Yes	Yes
DFS	$O(b^{d})$, d is the depth of the search tree	O(bd), only needs to store the current branch	Yes, on a finite graph. If the graph is infinite or we can't check for repeated states, no	No
DLS	$O(b^{l})$, l is the depth limit	O(bl), l is the depth limit	Yes, as long as the solution is not past the limit	No
IDS	$O(b^{d+1})$, same as BFS	O(bd), same as DFS	Yes	Yes if all actions have uniform cost
A*	$O(b^{d})$, d is the maximum depth of the search tree	$O(b^{d})$, n is the maximum depth of the search tree	Yes	Yes, if our heuristic is admissible (always underestimates the true cost to the goal)

Part 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 success of Deep Blue was a result of continuous upgrades to hardware and software, as well as building off of the data from previous models. One of the main challenges that the Deep Blue team faced was large searching capacity. The model had to be capable of searching further depths and in a reasonable amount of time in order to match up against Grandmasters. The search also had to be non-uniform, as any uniform search would not be able to meet the speed and aggressiveness of players like Kasparov. However, by utilizing custom-built hardware with over 500 processors, it was able to evaluate 2-2.5 million chess positions per second. This kind of approach could be applicable to any similarly computationally heave problem where constant, speedy evaluations are necessary. However, as Deep Blue is a machine made exclusively for chess, its particular software and optimizations are most likely not directly applicable to other problems. For example, Deep Blue was built with high-level chess algorithms and openers, allowing it to play chess optimally. While this might be useful for similar logistical problems, like finding optimal routes given preexisting data, it is obviously not directly applicable without substantial modifications beforehand.

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.

AlphaZero takes a much more human-like approach to search in comparison to Stockfish. It only searches 80 thousand positions (nodes) per second in chess, in comparison to Stockfish’s 70 million per second. AlphaZero is able to win because of the difference in their search methodologies. Stockfish, like Deep Blue, relies on high-performance alpha-beta pruning and handcrafted domain expertise to examine millions of chess positions per second. AlphaZero, on the other hand, uses a Monte Carlo Tree Search with a deep neural network. Rather than focusing on quantity of search nodes, it focuses on a smaller set of the most promising nodes and evaluates the game position and outcomes, therefore reducing the need for brute-force exploration. Through self-play, AlphaZero is able to evaluate chess positions and score them similar to how a human chess player might evaluate a position. It makes up for its lack of nodes evaluated by focusing on the most promising nodes overall and subsequently evaluating results, allowing it to consistently outperform models like Stockfish within a few hours of playing.