CMPUT 366 Reading-Writing Exercise 12

Name: Minghan Li
Student ID: 1561234
CCID: minghan4

Learning and searching are tightly connected to each other: we can form a bettering understanding by learning from the environment and use that understanding to search for a better or optimal solution. We have already learned a lot about reinforcement learning in this semester, and to move further we have to know about search. Heuristic search and Monte Carlo Tree search are one of the most commonly used search algorithms in combinatorial games, such as Chess, Go and etc; these two kinds of method have their own merits and disadvantages: Performance of heuristic search heavily depends on the quality of the heuristic function, and Monte Carlo Tree search avoids hand-crafting the heuritic function but requires tons of simulations to gurantee to have good results. After reading the materials, I think combining learning with them together really is the key to avoid those pitfalls and scale the algorithms to solve much bigger problem.

One of the most widely used heuristic search algorithms is A* search. A* search belongs to the family of greedy algorithm, which means that it solves problems by searching among all possible paths to the solution for the one that incurs the smallest cost, except the cost that A* tries to minimize is formulated as $f(n) = g(n) + h(n)$, where $g(n)$ is the actual cost from the starting point to one specific point n, and $h(n)$ is a heuristic that estimates the cost of the cheapest path from n to the goal. For the algorithm to find the actual shortest path, the heuristic function must be admissible, meaning that it never overestimates the actual cost to get to the nearest goal node. With good heuristic, A* algorithm would drastically reduce the computational time to find the solution and it gurantees to find the optimal solution if the heuristic is admissible. However, in games like Go anf Chess, designing a good heuristic function to evaluate massive numbers of board positions are very difficult. Fortunately, a reliable method called Monte Carlo Tree search which doesn't require any heurstic works pretty well on board games and very quickly becomes one of the dominant methods in combinatorial games. In earlier times, board games like Go requires expert knowledge to bias the search algorithms in order to make them at least "work", but Monte Carlo Tree Search doesn't require any expertise in the game and improve the performance of board game AI by a big margin.

Monte Carlo Tree search actually has two fundamental concepts: Tree search which means selecting and expanding nodes in a game and Monte Carlo means running many simulations of the game to estimate the values of moves. The algorithm is quite simple but effective. It requires iterating among four steps: selection, expansion, simulation and backpropagation. Ideas like upper confidence bound have been applied in the algorithm to obtain a better exploration-exploitation trade-off. But one disadvantage of Monte Carlo Tree seahc is that in order to get better approxiamtion, we need to run many simulations which requires massive computational resources for board game like Go. It is really tempting to combine it with heuristics so that we can decrease the computational time. Untill recently, AlphaGo Zero claims a success over Go without any human knowledge by combining Monte Carlo Tree Search and reinforcement learning. The value function can be viewed as heuristic that frees Monte Carlo Tree search from looking all the way to the end of the game; instead it just has to look several steps deeper and use value function as the estimate. AlphaGo's success is not just because of reinforcement learning and deep learning, the search algorithm in it also plays a big role and I believe that more and more applications will incorporate search and learning algorithms together to solve bigger problems in the real world.