Deep Blue Paper

The paper reviews Deep Blue's improvements on its large search ability, complicated evaluation function and a special designed search algorithm.

Deep Blue's search ability is enhanced by many factors. The paper highlights following points:

1. hardware support on search, evaluation and move generation: there are special chess chip designed to generator moves in a best-first order (as much as possible), to evaluate the board and to search the game tree. They combine to allow Deep Blue to search a full-width depth of 12.2 on average in a 3-min search.

2. parallel search: Deep Blue can run search in parallel on its 30-node computer and 480 chess chip in some situation. Although no hard evidence, it's interpolated that the efficiency of parallel search is about 8% (I don't know what does that mean in the paper but that's the result they provide)

3. Opening book and endgame data base. Deep Blue has hardware storage of opening book and endgame database to refer to when doing the search. They are not often used. For endgame data base, only game 4 VS Kasparov used it.

Deep Blue's evaluation function is a some of scores of patterns. Its chess chip can recognize about 8000 patterns. The score for each pattern can be fixed or dynamic according to the search. Those patterns range from simple ones to complicated ones that encodes human knowledge as well. In total, about 8150 features are used in the evaluation function.

Deep Blue also has its own improved version of alpha-beta negamax formulation, called "dual credit with delayed extension" (DC). The feature of DC is based on observations of some principles on forced pairs of moves, and how to avoid search exploding. It's un-uniformed, meaning can choose some game tree branch to explore more. It does this by keeping a concept of "credit", which roughly means how good one's moves so far is. The really credit that influences the choice on how deep to go in a search tree is relative, depending on the relative value of opponents and players. This algorithm allows Deep Blue to do deep search on promising branches while avoiding search exploding.

Other techniques Deep Blues faces include time control, avoiding oscillating searches and keeping search envelop in search, load balancing and sharing knowledge between nodes in parallel and keep extension ability in hardware design, and many other points.

Game Tree Searching by Min-Max Approximation

This paper presents an extension of iterative searching algorithm. It aims at improving the choice of child node to explore during searching. The technique it uses is based on an approximation of the min-max function. This extension is implemented in the game called connect-four and tested against minimax algorithms with alpha-beta pruning and iterative deepening. The result shows that when CPU circles are the limiting factor, this extension is inferior than the minimax but if the number of calls to "move" routine is the limiting factor, this method outperforms minimax.

The key idea in this extension to make a better choice of node-to-explore is based on derivatives. The algorithm tries to find the tip node ("leaf" nodes in a partial game tree that are not terminal nodes so that the nodes can be expanded for next step.) with the largest impact on root's nodes score. To achieve this, the min/max function normally used to choose a tip node need to be replaced by an approximated function that has continuous derivatives (a function that look "smooth"). The authors choose generalized p-mean function with larger than 10 p value.

The algorithms need to get the impact of a tip node relative to the root node so that the chain rule of derivatives is used to calculate the derivatives of the score of a tip node to the scores of roots. To reduce parameters and same memory, the algorithm breakdown he derivatives by edges and stores parameters in each parent and child nodes. Those parameters can be used to find the any derivatives along a path from the root to any tip node or between two nodes with ancestor-descendent relationship.

Other details of the algorithm include but not limits to use the min/max function instead of generalized p-mean function when calculating the actual value of derivatives and choose some constant parameters for the calculation.

The tests are conducted on the same game and same hardware for all possible starting configurations so that the experiments are comprehensive. Besides confirming the effectiveness (in the move-bound situation) of this algorithm, the experiments aloo shows future discussions such as how to improve the efficiency when there is only one valid choice for a tip node, to spend less time traveling up-and-down on the same paths and choice of better parameters in either derivative calculation and the evaluation function.

AlphaGo Paper

In this paper, value network (v, evaluate the current board) and policy network (p, estimate the probability of each move under current board) and the integration of v and p in Monte Carlo Tree Search (MCTS) are explained. The result is that AlphaGo being able to dominate all current strongest Ai players and defeat human masters the first in history.

DeepMind applies deep neural networks (DNN) in building value network and policy network. They first train a DNN to embody the policy network, p1, using data from game records from human experts. DNN enable it to reach a prediction accuracy never achieved before. But at the same time, they develop another policy network, p2, using simpler algorithm than DNN but faster than DNN to produce a prediction. Both p1 and p2 will be used in MCTS, but in different stage.

DeepMind applies DNN also on value network. Once they have p1, they improved it and twig it towards winning game rather than predicting human experts' move by applying reinforcement learning of a DNN. The main idea is to let the p1 and variation of p1 play against each other and learn by winning. The result is p3. The two p3 DNN play with each other and produced a large dataset that then be used to train a value network. Of course, this value network, v, is also embodied by a DNN.

In gaming, MCTS works by running many simulations. In each simulation, in steading of expanding a tree node to all its children, only on child is selected to expand and then only one of this child's child is expanded...until the game terminates. Then MCTS averages the simulations and pick a child at the beginning.

The variation MCTS that DeepMind uses combines both v, p1 and p2. In one simulation, the first stage of expansion is based on choosing the child with the largest score receptivity for L levels. This score contains the probability value calculated by p1. Then after reaching L level, v is used to calculate a score of the board at L and p2, the faster but less accurate policy, is used to simulate the game until termination and returns a score of board at L. Both scores from v and p2 are combined to give a final score of the board at level L. This score is backup to the parent node, where all backup values are averaged to get a score the L-1 level. This backing-up happens towards the beginning of this simulation. Then starts another simulation...the scores are shared between simulations so each simulation brings additional info. Finally, after running n simulations, the root chooses one path that's being choose most times in simulations.

AlphaGo, even with single-machine hardware, beats all Go Ai players with high winning rates or high give-ups (free moves for opponent). The distributed version then beats single-machine version with

another large margin. And finally, it becomes the first Ai Go player that beats human masters in the history.