Notes on an alternative approach to move choice in games such as Chess

Tom Holden

Motivation

Suppose at some point in a one-player game, a player has a choice between playing L and R. If they play L, then in the next period, they will have a choice between moves L1, L2, L3, L4 or L5, which their evaluation function gives scores of 0.499, 0.498, 0.497, 0.496 and 0.495, respectively. If they choose R, then in the next period they will have a choice between R1, R2, R3, R4 or R5, which their evaluation function gives scores of 0.5, -1, -1, -1 and -1, respectively. If L1, …, L5, R1, …, R5 were terminal nodes, then it is clear that playing R would be the correct choice, as 0.5 > 0.499. But if L1, …, L5, R1, …, R5 are not terminal nodes, then it seems clear that playing R would be crazy. Given the inevitable imperfections in the evaluation function, it seems highly likely that were we to evaluate the tree at greater depth, one of the L1, …, L5 options would obtain a higher score than 0.5. In general, we should value having a choice between many good options, when evaluations are noisy, as it is likely some will turn out to be even better once further searches are performed.

There are several possible approaches to dealing with this issue. One is just to perform those further searches until one arrives at evaluation scores that are more reliable (less noisy), and that show a clearer winner. This is part of the motivation behind quiescence searches. However, it is often hard to know how reliable the output of the evaluation function currently is, and quiescence at best provides a heuristic for when it is likely to highly unreliable, rather than one for when it is reliable. Another approach is to attempt to bound evaluation scores,[1] or to work with some kind of "fuzzy score".[2] Since these are still not treating uncertainty in a mathematically rigorous way, it is doubtful if they are worth the increase in search tree size. The approach I will present here will treat uncertainty in a rigorous way, but will still search a reasonably sparse tree.

The algorithm

Suppose that our game has three end states (win, draw, lose), with payoffs $v_W > v_D > v_L$, respectively. A perfect evaluation function would return whether any given state was a win, draw or loss. Therefore, to incorporate uncertainty into an evaluation function, we just require that given a game state, our function returns an estimate of the probability of each of these events. Let us suppose that we have such a function, and let us call the probabilities returned p_W, p_D and p_L respectively. This function is clearly similar to a standard evaluation function, but whereas a standard evaluation function just provides ordinal information (is this state better than another), here we require "cardinal" information: the levels themselves are meaningful, making our task harder in calibrating it.

My initial suggestion for calibrating the evaluation function would be to create this function via running a multinomial logistic regression on the states and outcomes of games between master level

[1] I believe this is the PB* approach mentioned in Junghanns (1994) (link: http://www.top-5000.nl/ps/Fuzzy%20Numbers%20in%20Search%20to%20Express%20Uncertainty.pdf), though I have been unable to find a copy of the book/thesis "Searching with Probabilities" by Andrew Palay, which sets out this algorithm. The B* approach from which PB* is derived is described in Berliner (1979) (link: http://www.sciencedirect.com/science/article/pii/0004370279900031#).

[2] See Junghanns (1994) (link as above).

players, using the traditional inputs to evaluation functions (possibly along with polynomial terms) as regressors. I.e.:

$$p_O = \frac{e^{X\beta_O}}{e^{X\beta_W} + e^{X\beta_D} + e^{X\beta_L}},$$

where $O \in \{W, D, L\}$, X is the matrix of regressors (one variable per column), and $\beta_W, \beta_D, \beta_L$ are the estimated coefficient vectors.

While multinomial logistic regression is a little limiting in the correlation structure it allows between outcomes,[3] it is a natural initial benchmark, and will be reasonably fast to implement and estimate, due to the speed of numerical exponentiation (though not, sadly, division).

There are two key elements to the algorithm. The first is how we aggregate leaf probabilities up to probabilities at the head; the latter is how we explore the tree. We begin by discussing the former.

Suppose that at some parent node the player has a choice between n child nodes with probabilities $p_{W,i}$, $p_{D,i}$, $p_{L,i}$ for $i \in \{1, \dots, n\}$. (So this is a "max" type node.) We wish to derive win, draw and loss probabilities at the parent node. To be able to win from the parent node, the player just needs to know that one of the child nodes is definitely a win. Let us imagine that by the time the player arrives at this parent node (which may already be deep in the game tree), she will be able to evaluate the child nodes accurately enough to classify each of them as a win, draw or loss. So then the probability that the parent node is a win, is just the probability that at least one of the child nodes is a win. I.e. it is one minus the probability that none of the child nodes are a win. Hence:

$$p_W = 1 - \prod_{i=1}^{n} (1 - p_{W,i}).$$

(The $\prod$ symbol denotes a product.) Similarly, the probability of a draw is the probability that none of them are a win, and at least one is a draw. By the laws of conditional probability, this is the probability that none of them are a win, times the probability that at least one is a draw, conditional on none of them being a win. Hence:

$$p_D = \left[\prod_{i=1}^{n} (1 - p_{W,i})\right]\left[1 - \prod_{i=1}^{n} \frac{p_{L,i}}{1 - p_{W,i}}\right].$$

Finally, the probability that the parent node is a loss is just the probability that every child node is a loss, i.e.:

$$p_L = \prod_{i=1}^{n} p_{L,i}.$$

As a confirmation of these updating probabilities, I verify that they do indeed add up to one.

$$p_W + p_D + p_L = 1 - \prod_{i=1}^{n} (1 - p_{W,i}) + \left[\prod_{i=1}^{n} (1 - p_{W,i})\right]\left[1 - \prod_{i=1}^{n} \frac{p_{L,i}}{1 - p_{W,i}}\right] + \prod_{i=1}^{n} p_{L,i}$$

[3] I would be happy to share thoughts on how a wider range of correlation structures could be allowed for later on.

$$= 1 - \left[\prod_{i=1}^{n}(1 - p_{W,i})\right]\left[\prod_{i=1}^{n}\frac{p_{L,i}}{1 - p_{W,i}}\right] + \prod_{i=1}^{n} p_{L,i} = 1.$$

For "min" nodes, all of the W and L labels in the above formulas are swapped.

Of course, this algorithm is not without simplifying assumptions of its own, not least the fact that I have implicitly assumed independence between outcomes at different child nodes,[4] and that I have ignored the "second order uncertainty" coming from not having perfect evaluation functions, even when the player arrives at the parent node. However, I would contend that these updating rules will still go a long way towards allowing the player to correctly account for the uncertainty in her evaluation function, producing interesting play, if nothing else.

The second key element of our algorithm is how we traverse the game tree. Given that the probabilities at a parent node are a function of the probabilities at every child node, one might be concerned that brute force evaluation of the entire game tree was the only possibility. Certainly, standard alpha-beta will not work. However, there is no reason why the probabilities combined at a particular parent node should not be a function of evaluations to vastly different depths, with some child probabilities coming directly from the evaluation function, and others coming from the combination of probabilities from huge sub-trees. The question then is in what order the game tree should be explored.

My suggestion, is to treat the exploration of the game tree as akin to a multi-arm bandit problem. Briefly, this is the problem of choosing the optimal exploration/exploitation trade-off when faced with many investment opportunities, each with unknown return distribution. One is most concerned with the probability of success down the path on which the game is most likely to flow, thus, one might be tempted just to traverse the "best" path (based on depth-0 evaluation) as deeply as possible. For example, if I told you that you were allowed to evaluate the child nodes of only two parent nodes, you would most likely choose the root, and the node that seems to be the best there (to make sure you are not making a mistake). This is the "exploitation" incentive in game-tree exploration. The "exploration" incentive on the other hand, comes from the desire to avoid missing out on nodes given low scores by depth-0 evaluation, but which are actually far stronger.

In general, optimal behaviour in multi-arm bandit problems is not analytically tractable; however, a number of heuristics rules have demonstrated good practical performance. One simple but well performing one is to play the "arms" with probability in proportion to the probability that they are (weakly) the best. For us, this means that starting at the rood node, we choose a child node with probability in proportion to the current probability the given child node weakly dominates all others. The selected child node becomes the new parent node. If we already have probabilities for the new parent node's children, then we repeat the procedure, choosing a child node with probability in proportion to the probability the given child node weakly dominates all others. If, on the other hand, we have never evaluated the probabilities of the new parent's children, then we evaluate these probabilities, and then reverse our steps back up the tree updating probabilities, as described previously.

[4] Potentially, this could be corrected for heuristically, by taking a re-normalised weighted average between the probabilities calculated in this manner, and the average outcome probabilities across all child nodes. However, given the max-min structure, this may not be necessary, as one might hope biases would roughly cancel out between moves.

We just need an expression for the probability that a given child node is weakly dominant over its siblings. Assuming we are at a "max" node, the probability that node i is weakly dominant is the probability that it is a win, plus the probability that it is a draw, and all its siblings are not wins, plus the probability that it is a loss, and all its siblings are losses. I.e.:

$$p_{W,i} + p_{D,i} \prod_{\substack{j=1 \\ j \neq i}}^{n} (1 - p_{W,j}) + p_{L,i} \prod_{\substack{j=1 \\ j \neq i}}^{n} p_{L,j}.$$

Evaluation thus consists of multiple trips down the game tree, each one terminating with the addition of new leaf nodes, and the calling of the depth-0 evaluation function on them, following by updating probabilities back up the tree. As a result, search can be trivially parallelised, which is certainly essential for acceptable performance.

At the beginning of a new turn, all of the old game tree starting from the move that was in fact made, may be preserved, along with the found probabilities. However, to preserve performance, it would probably be sensible to begin each new turn with a memory defragmentation of the old game-tree, to ensure cache efficiency. It would probably make sense to maintain a record at each game tree node of the memory size of the entire sub-tree starting from that node, as the size of the entire game tree ought to be capped to be equal to the available memory, minus the size of the largest root sub-tree, to enable the aforementioned memory defragmentation to take place. The need to preserve the entire tree in memory is certainly unfortunate, but it seems inescapable. Potentially, when memory was filled up, deeply evaluated sub-trees with dominated performance could be shut-off at their root, the memory of the sub-tree being freed up for re-use.

Evaluation will stop either when some time based heuristic cut-off is reached, or when all but one root node's probability of weak dominance fall below some very low threshold. We then just have to specify what should be played at the root node. There is clearly no point randomising at the root, so the player should just pick whichever move maximises their expected return, i.e. that for which $p_{W,i}v_W + p_{D,i}v_D + p_{L,i}v_L$ is maximised.

Discussion

It would be naïve of me to think that this had a chance of competing with current state-of-the-art chess algorithms. However, I would argue that this approach still has considerable value, and I would really urge someone to pursue its implementation. Firstly, while it may not beat a top chess algorithm, it could well beat a naïve implementation of alpha-beta, at least opening the possibility that with further work it may be competitive. It may also be less reliant on opening books in the early game. Secondly, it will play very differently to alpha-beta algorithms, and arguably in a more human-like fashion. For many mid-ranking players, there is considerable value in being exposed to different styles of play, beyond those that can be captured by tweaking pawn values in evaluation functions. So, I would expect the resulting program to be in considerable demand. Thirdly, this algorithm may be a promising contender for games such as Go for which alpha-beta appears to be dominated by undirected random-search, since it represents a kind of "middle-way" between the two approaches. Finally, it has some considerable curiosity value, I believe. Is taking into account the uncertainty in evaluation functions important? Can an algorithm not based on alpha-beta be competitive?

I truly hope one of you will take up the challenge of implementing this algorithm. I will of course help as much as I can, though my time is rather limited, hence not implementing this myself.