6 Computer Game-Playing

This chapter considers how the results so far established may be applied to the area of computer game-playing, as well as giving my personal views on the subject. Its tone is intended to be slightly less formal than previous chapters, since the problems tackled are bigger and more open-ended. We put this discussion in context by highlighting the tension between what is 'optimal' — in the dynamic stochastic control sense — and what is desirable in a practical sense. The remaining sections then outline, in varying degrees of detail, some new approaches to game tree searching and game-playing.

6.1 Optimality and Limited Rationality

The recent paper of Baum and Smith [6] summarises as follows the dilemma facing those who struggle for provably optimal policies in the field of computer game-playing:

> "One could . . . in principle, attempt a catalog of all leaf expansion strategies for any given game tree with a fixed number of leaves. If we could do vast computations offline, we could pinpoint one such strategy as optimal. But to be meaningful in practice, any "optimal" strategy must take into account its time cost (if we spend less time deciding which leaves to expand we can expand more leaves), and also the interaction of one leaf expansion with future expansion decisions. We know of no tractable approach to

> computing a provably “optimal” strategy. Since we don’t know how to compute efficiently the exact decision theoretic utility of expanding leaves, we search for a useful approximation.”

The result of Section 5.4 can be seen as a partial solution to the problems raised above, in that, by application of standard dynamic stochastic control methods, we have calculated the exact decision theoretic utility of expanding a leaf in a simple search game. However, this ignores the problem of limited rationality mentioned above; although the policy we have calculated is optimal — in the standard sense — we have no guarantee that it is useful in a practical sense. Since the dynamic stochastic control model takes no account of its deliberation time, there must always be a question mark over the practical feasibility of solutions adjudged to be ‘optimal’ on such a basis.

The way in which dynamic stochastic control has been used so far in this work is to deduce optimal policies for a range of simple games. These studies are intended to yield insight into the game-playing problem, and so lead on — directly or indirectly — to more complicated and powerful models.

Another way in which dynamic stochastic control might be applied to game-playing is to look for policies that are optimal within a certain class, such as the work done by Smith [79]. One might wish, for example, to choose a particular subclass of those tree search policies which are effectively implementable — for example, one which is $O(n)$ in the number of leaves searched. In this way one could ensure the feasibility of such optimal policies as were deduced. The success of such an approach would seem to rest entirely upon an insightful choice of subclass.

Such an approach again stops short of tackling the fundamental problem of limited rationality. More powerful and more challenging still would be a framework which provided the means to show a policy to be optimal *inclusive of its own deliberation costs*, presumably with reference to some standard computer architecture. This would represent a very significant extension of standard dynamic stochastic control theory in the direction of computer science.

6.2 Conspiracy Probabilities

In their recent paper on probabilistic B* search, Berliner and McConnell [16] report the failure of their attempts to combine B* with the conspiracy number search algorithm. They summarise their conclusions as follows (italics in the original):

> "[Berliner] quickly found out ... something very important about why conspiracy number search has not worked. While it is useful to think of conspiracies in the above way, and to plan to attack the conspiracy with the fewest conspirators, in practice this does not work. Conspiracies fall into buckets. There are buckets of conspiracies of magnitude 1, of magnitude 2, magnitude 3, etc. In Chess (and we would assume in almost all domains), there are lots of potential conspiracies, and the number of conspiracies of

magnitude 2 is usually quite large. Thus, when dealing with conspiracies of magnitude 2, one must examine them in some quasi-random order, and the chances of finding *the* conspiracy that is easiest to break is quite small. It is like a breadth-first search of conspiracies, with no other clue as to what might make a given conspiracy easy to break. *This is the reason for the failure of the conspiracy approach in game-playing.* There is no good method for deciding the weakest conspiracy of a given magnitude."

This is a significant objection to the conspiracy number search as described in Subsection 1.2.2; the implicit assumption in 'conspiracy number' that all nodes are equally likely to conspire is a serious limit to the efficiency of such a search. The successes of such simple methods as singular extensions and the null move are a testimony to how easily the most uncertain positions can be distinguished. It is therefore desirable to use such probabilistic information as is available. Subsection 3.4.2 shows how, in the two-valued case, this can be done optimally for conspiracies of size one by application of the stochastic OR-Tree model of Chapter 3. One step further down this road is to abandon altogether the notion of conspiracy 'number' in favour of the notion of conspiracy 'probability'. We now examine how this might be achieved, and reflect on some of the barriers to be overcome if such a scheme is to be developed into a practical selective search algorithm.

6.2.1 A Redefinition of Conspiracies

The child of root with the highest backed-up score we term the *provisionally best move*, and its score the *provisionally best score*. This is the move we make if no further search is carried out, so defines the utility of moving immediately from a position.

The original definition of a 'conspiracy' is described in Subsection 1.2.2. It refers to a minimal set of leaves which always has a chance of changing the provisionally best score. We note that not all such changes cause a change to the provisionally best move. This is a matter of importance to any algorithm which makes the Meta-greedy and Single-step Assumptions of Subsection 1.2.6, which are important ones on grounds of tractability. The reason is that the conjunction of these assumptions implies that unless a search has a chance of changing the provisionally best move, it has no value, and so the original definition of 'conspiracy' includes some sets of nodes which will never be of interest.

McAllester [50] originally detailed two categories of conspiracy, 'Prove Best' and 'Disprove Rest' strategies, as follows:

1. Sets of nodes that may decrement the score of the provisionally best move, so that it assumes a lower value than that of another move.
2. Sets of nodes that may increment the score of another move, so that it exceeds that of the provisionally best move.

We desire to include the following additional category to include those of the kind shown in the figure below, which were not originally included:

3. Sets of nodes which may decrease the provisionally best move's score and increase that of another move to exceed it.

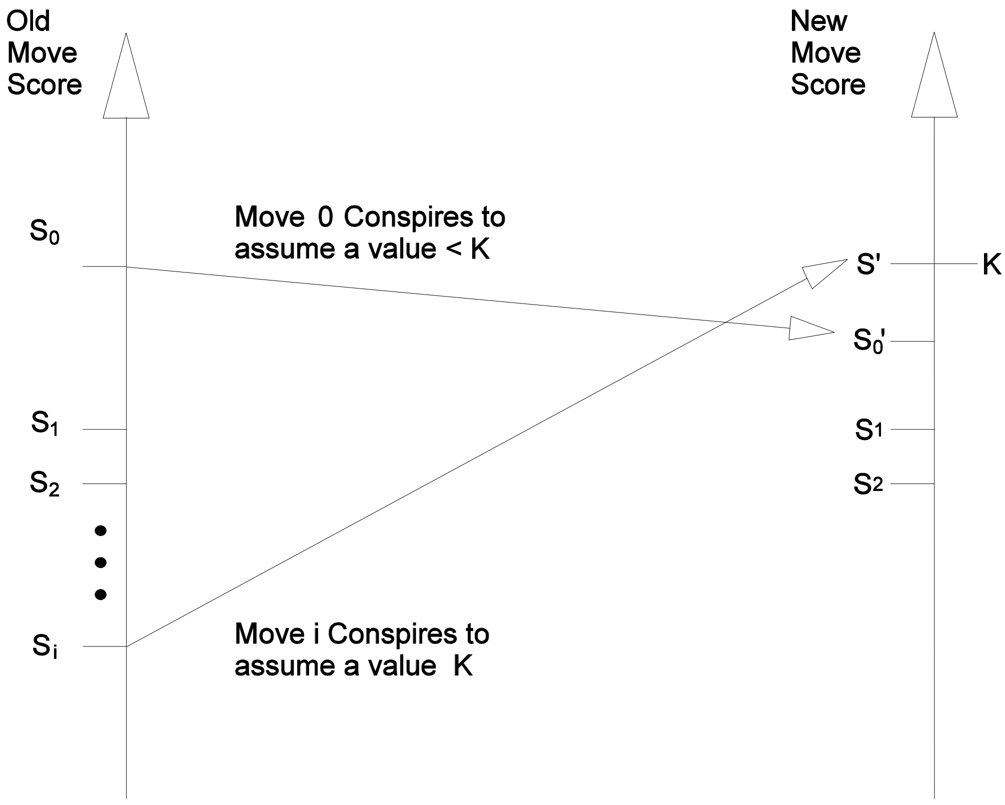
Old Move Score
New Move Score
S0
Move 0 Conspires to assume a value < K
S'
K
S0'
S1
S2
S1
S2
Si
Move i Conspires to assume a value K

Figure 35: Moves 0 and *i* Conspire to Change the Provisionally Best Move

We now redefine the term *conspiracy.* We have previously defined a tree's conspiracies to mean a set of nodes which, if they were to assume different scores (as a result of search) could together effect a change to the backed-up score of that tree's root node. We now introduce the idea of a *conspiracy to change the provisionally best move.* This is similar to the above definition except that, rather than changing root's score, the effect of the nodes' coordinated change is to change the tree's provisionally best move. This redefinition excludes those conspiracies which have no value under the single-step assumption, and includes the third class omitted by McAllester [50].

Assuming the children of root are ordered so that child 0 is the provisionally best move, and that move i has score s_i, a general form for such a conspiracy is (i, K), where i is the number of the move that conspires to assume a value $\geq K$ while move 0 conspires to assume a value of $< K$. Type 1 conspiracies may thus be expressed $(1, s_1)$, and type 2 conspiracies $(i, s_0 + \epsilon)$, where i is the move involved and ϵ the granularity of the evaluation function.

6.2.2 Ø-based Approximation Methods

The reward rate principle may be applied to the task of choosing a conspiracy in the following straightforward way. Each node of the search information is given a p and a t value with meanings analogous to the boxes case. For leaves, t, the expected time to expand each node of the conspiracy, is simply 1, while the probability that the conspiracy will change that node's value may be deduced directly from the prior distribution for expansion of that node.

For internal nodes of the search information with a negamax backed-up score of 1, all its daughters of value -1 must conspire, so the p and t values are calculated from their daughter nodes as follows:

$$\begin{aligned} p_{\text{root}} &= \prod_{i=1}^{n} p_i \\ t_{\text{root}} &= \sum_{i=1}^{n} t_i \end{aligned}$$

The case in which an internal node is scored with a negamax backed-up score of -1 requires only a single daughter to conspire, and so the efficiency of the search centres around how this choice is made. The naive Ø-based approximation chooses the daughter with the greatest Ø value:

$$\begin{aligned} p_{\text{root}} &= p_{i^*} \\ t_{\text{root}} &= t_{i^*} \end{aligned} \quad \text{where } i^* \in \{1 \ldots n\} \text{ maximises } \frac{p_i}{t_i}$$

We now consider how to percolate these p and t values back up the tree. In order to search efficiently for a conspiracy, we need to seek out those conspiracies with as large a p value and as small a t value as possible. We now extend this principle, in a straightforward fashion, to the task of choosing which daughter to search from a '-1' node. Defining $Ø_i = p_i/t_i$ as before, suppose that, from a '-1' node, we will select the daughter which maximises Ø. This approach does not lead necessarily to choosing the overall conspiracy which maximises Ø, as we shall see from the following example.

Suppose that we are investigating the tree shown overleaf in Figure 36. The values shown beneath the nodes Q, S, and T, are the p and t values of these nodes. Since $Ø_{\mathsf{S}} = (.1/1) > (.5/6) = Ø_{\mathsf{T}}$, S is deemed the most promising descendant of R to search for a conspiracy. Hence, R is scored

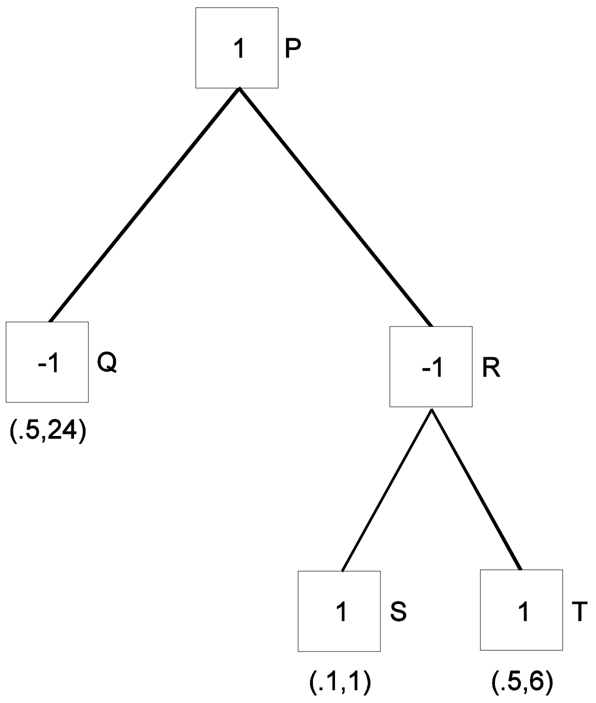

Figure 36: Inefficiency of Ø-based Approximation Method

(.1,1) and so P is scored (.05, 25). This leads to investigation of conspiracy {Q, S} which has a reward rate of 0.002. The optimal minimal conspiracy {Q, T}, has score (.15, 28), and so a reward rate of $0.008\dot{3}$. This example illustrates the problem that the overall reward $(1 - \Pi q)/\Sigma t$ cannot be maximised simply by maximising the individual p/t ratios of the subconspiracies. This problem is most severe for trees in which there is considerable variation between conspiracy t values.

A 'perfect' solution would require a catalogue of all the p and t values of the conspiracies possible below a node. This would require large overheads, both in terms of meta-calculation and in storage. Since the space required to store the tree would be super-linear in n, the number of nodes in the tree, we discard this approach as infeasible.

A simple compromise which reduces inefficiencies associated with a poor choice of conspiracy whilst increasing resource usage by only a constant factor would be not to store all the possible conspiracy details at a node, but to store up to a certain fixed number.

In fact, since there are different percolation formulae for '1' nodes and '-1' nodes, it may well be efficient to store the details of different numbers of conspiracies for odd and for even depths of the tree. The motivation for storing a set of p and t values is to enable a better choice to be made about which conspiracy we wish to search, and so it is also reasonable to store more conspiracies for nodes higher up the tree (particularly the daughters and granddaughters of root), since these nodes have the largest range of possible conspiracies from which to choose. One way of visualising the approximation which is going on is to consider Figure 37, overleaf, which shows (a continuous approximation of) the profile of available conspiracies available from a node.

A simple example of how this approach would work is to store at each node the p and t values of two conspiracies. A natural approach would be to chose the conspiracy with the largest p value and the one with the smallest t value, (p_1, t_1) and (p_3, t_3) in Figure 37. A choice between these two extremes could therefore be made at the top level '1' node in the tree. If the overall

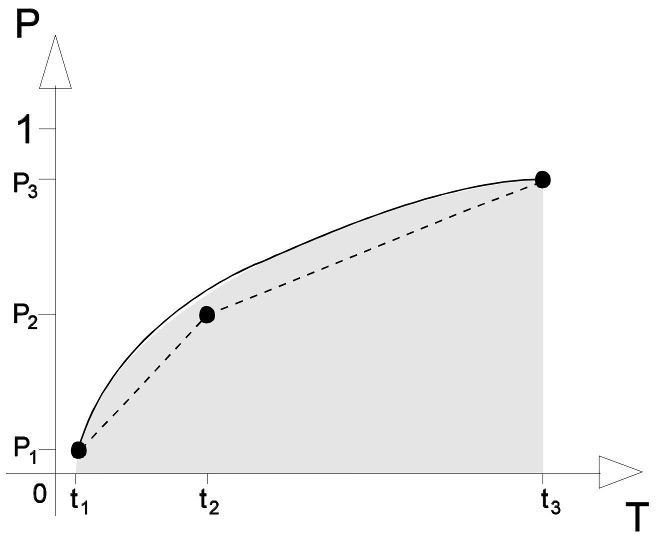

Figure 37: Range of Possible Conspiracies

reward rate of the conspiracy chosen in this manner was still too low, it could be improved by the storage of the details of more conspiracies. As the diagram shows, if details of a third conspiracy were stored at each node (say, that with the greatest Ø value) this would have an intermediate position, although it would not necessarily be the most probable conspiracy of its size.

6.2.3 PCN* Search

We consider a simpler selective tree search technique, PCN*, which is based upon conspiracy number search, but which searches conspiracies in order of probability, p, rather than reward rate, Ø. This is a less ambitious goal than attempting to select conspiracies based upon the p and t values. However, since the most likely conspiracy will tend — all other things being equal —

to be the one with the fewest nodes, and hence the cheapest to evaluate, this approximation may not be as crass as its simplicity would suggest.

We now explain briefly how it works, with the help of an example. The nodes of the tree each have a scalar score, amongst which the usual minimax relationship holds. We assume that the scores take integral values from 1 to m. In addition, every node has a *conspiracy probability vector* (c.p.v.), which details the probability with which this score can be changed by the investigation of a conspiracy amongst its children. Using C to stand for a single conspiracy, and $\mathbf{C}$ for the set of conspiracies, the c.p.v. is defined as follows for a node i with scalar score v_i:

$$\left(\begin{array}{rclcl}
p^i_1 & = & \max_{C\in\mathbf{C}}\{P[C \text{ will cause this node to have score} & = & 1]\} \\
p^i_2 & = & \max_{C\in\mathbf{C}}\{P[C \text{ will cause this node to have score} & \leq & 2]\} \\
\vdots & & & & \\
p^i_{v_i-1} & = & \max_{C\in\mathbf{C}}\{P[C \text{ will cause this node to have score} & \leq & v_i-1]\} \\
p^i_{v_i} & = & 1 & & \\
p^i_{v_i+1} & = & \max_{C\in\mathbf{C}}\{P[C \text{ will cause this node to have score} & \geq & v_i+1]\} \\
\vdots & & & & \\
p^i_{m-1} & = & \max_{C\in\mathbf{C}}\{P[C \text{ will cause this node to have score} & \geq & m-1]\} \\
p^i_m & = & \max_{C\in\mathbf{C}}\{P[C \text{ will cause this node to have score} & = & m]\}
\end{array}\right)$$

The definition of $p^i_{v_i}$ can be motivated by expanding the concept of conspiracy to include the 'null conspiracy', which effects no change to a node's score with probability 1. This has the useful consequence that a node's $p^i_{v_i}$ value can be interpreted in harmony with both $p^i_{v_{i-1}}$ and $p^i_{v_{i+1}}$.

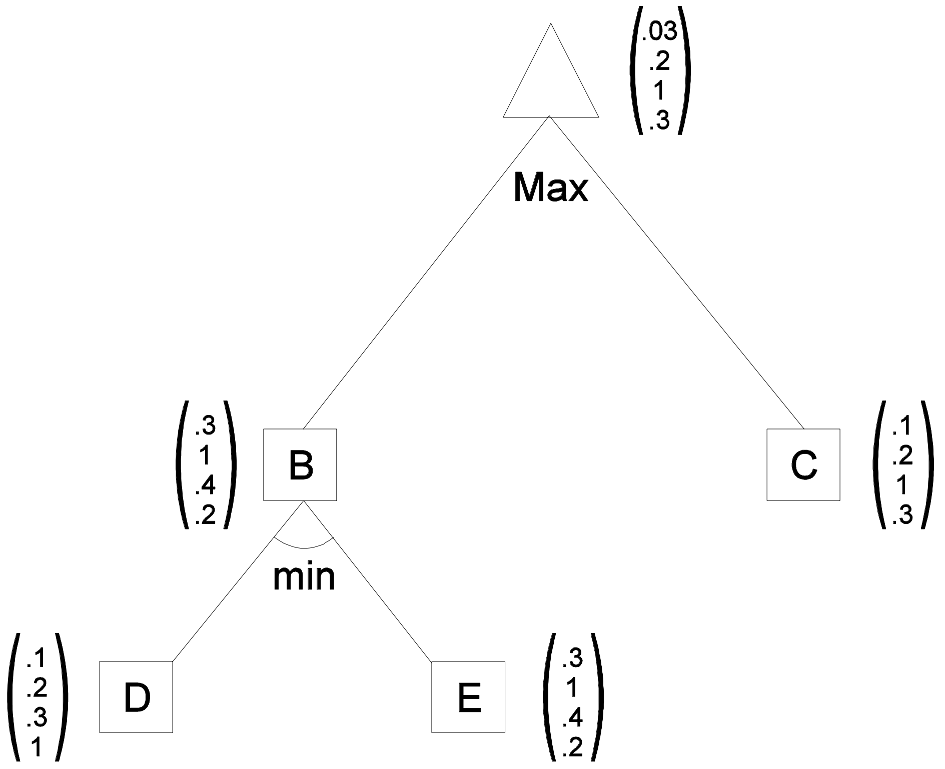

Figure 38: Example Conspiracy Probability Vectors

The above figure shows an example tree with conspiracy probability vectors, where m=4. Nodes C, D and E are leaves of the search information, and so their c.p.v.'s indicate the values that these leaves would take if expanded, which may be determined by the method of empirical evaluation described in Section 6.4. The c.p.v.'s of the internal nodes are percolated up from their daughters according to the formulae overleaf. These assume that a node has n daughters, which are ordered by their scalar score, so that

$v_1 \geq v_2 \geq \ldots \geq v_n$. For succinctness of expression, it will be useful to define two extra values, $v_0 = 1$ and $v_{n+1} = m$.

$$\text{For } \mathit{max} \text{ nodes :} \qquad\qquad \text{For } \mathit{min} \text{ nodes :}$$

$$p_j^{\text{parent}} = \begin{cases} \prod_{i=1}^{k} p_j^i & \Big| v_k \geq j \geq v_{k+1} \\ \max_i \{p_j^i\} & \Big| v_0 \geq j \geq v_1 \end{cases} \qquad p_j^{\text{parent}} = \begin{cases} \max_i \{p_j^i\} & \Big| v_0 \geq j \geq v_1 \\ \prod_{i=1}^{k} p_j^i & \Big| v_k \geq j \geq v_{k+1} \end{cases}$$

To see how this works in practice, suppose node D was expanded, and its c.p.v. was calculated (by percolating up from those of its daughters) to be $(.2, .4, 1, .2)$. Application of the formulae above yields a new c.p.v. for B of $(.3, 1, .4, .04)$. Note that the only element to have changed is p_4^{B}, the probability of a conspiracy causing the value of B to rise to 4. This being so, it is a trivial consequence of the percolation formulae that the only element of the c.p.v. of A which may change is p_4^{A}. In fact, a glance at the formulae show that p_4^{A} remains unchanged, since C was more likely to conspire to achieve a value of 4.

As demonstrated by the example above, the change of a leaf's c.p.v. does not necessarily percolate all the way back to root. For the purposes of complexity analysis, we nevertheless make this pessimistic assumption, and find that the cost of expanding a node is dominated by this percolation, which takes time proportional to the height of the tree. Expansion of a single node therefore has complexity $O(\log n)$, so PCN* search is $O(n \log n)$, comparable to BP and better than the MGSS* algorithm.

The implementation included in Appendix B searches one conspiracy at a time. If a greater degree of selectivity were desired, it could easily be modified to search one node at a time — picking from the most likely conspiracy the node with the greatest probability of conspiring. The price for this greater selectivity would be an increase in meta-calculation costs, although the algorithm would still be $O(n \log n)$. Another refinement over the included implementation would be to modify the criterion for selecting a conspiracy to investigate. Rather than just choosing the conspiracy with the greatest probability of making *a change* in the provisionally best move, account could be taken of the magnitude of the score difference. Conspiracies could be valued as follows:

$$V(C) = P_C(\text{Value of Best Move if } C \text{ works } - \text{ Value of current Best Move})$$

We make the point in Section 6.5 that for such a valuation to be sensible, the 'value' in the above expression should not be the simple scalar score, but should be a replacement that has been calibrated with respect to the final outcome of the game.

6.3 Choice of Search Step

The choice of search step size is inevitably a matter of compromise. On the one hand, the desire for selectivity motivates a small search step, on the other, the desire for minimal meta-calculation costs motivates a large search step. A further complication is the interaction between the choice of step size and the time-control policy. We saw in Subsection 3.6.2 how the weakness of a one step lookahead policy was exacerbated by choice of too small a step size. The MGSS* algorithm of Subsection 1.2.6 suffers from this problem; the meta-greedy search control policy terminates the search once the search information has a conspiracy number of two or more. This is because the search steps are single leaf expansions, which — under the single-step assumption — are scored individually, so no single step has positive value.

The BP algorithm of Subsection 1.2.7 avoids this problem by not attempting to be so selective. This is a pragmatic response to the problem. The parameter, 'gulp size' controls selectivity, providing a continuum of behaviours from highly selective to non-selective. Experiments by Smith, Baum, Garrett and Tudor [80] with BP in the game of Othello show how the performance of BP depends upon a suitable choice of gulp size. They conclude that the algorithms performs best with a value of around 0.04, so that each search step expands 4% of the leaves of the search information.

It would be interesting to test the suggestion of Section 1.4 that it is desirable for the degree of selectivity to increase during the course of search. This could be done by modifying BP so that the gulp size varied according to the number of nodes in the search information.

Another refinement to the 'gulps' method would be to vary the way in which a leaf is expanded[24]. Leaves of the search information with higher QSS

[24]Russell and Wefald [72] report a significant strengthening of the MGSS* algorithm

could be searched, in a single ‘expansion’, to greater depths, while those with a lower QSS could be only partially expanded (i.e. some of their children could be generated and appended to the search information). This offers the possibility of a more reasoned way of tackling the problem of variable selectivity than the previous suggestion. At the start of the search, its effect might be expected to be similar, since the low conspiracy number of leaves in a small tree would result in their high QSS.

Such an approach would benefit from the estimation of probability distributions for the expansion of nodes to varying depths, and so might be expected to increase the overhead involved in deciding whether or not to carry out a gulp of search. On the other hand, as well as increasing the selectivity of each gulp, it would increase their size, and so it need not necessarily entail an increase in the total burden of meta-calculations.

6.4 Estimation of Probability Distributions

The PSVB*, MGSS*, BP and PCN* algorithms make use of functions which evaluate a position with a probability distribution. The purpose of these distributions is to show how the evaluation is likely to change if further search is carried out. Whilst the PSVB* algorithm obtains this distribution via a rather obscure method involving the null move search, the more modern algorithms all use the same method, which might be described as empirical evaluation, or ‘training’.

when it was modified to partially expand nodes.

Human experts originally suggest a number of (usually quite simple) game-specific evaluation criteria known to be correlated with the winning chances of a position, and a means of combining them to deduce a single scalar score. Training proceeds by applying this evaluation function to a board position. This is searched to a certain depth, and the backed up scalar score of the position after the further search is noted. Once a large number of independently sampled positions have been examined in this manner, it is argued, the evaluation function will have a reliable probability distribution for each combination[25]. Russell and Wefald [70, 71, 72] used depth one searches to deduce their probabilities, while Baum and Smith [6] recommend empirical choice of the training depth.

One potential problem with this approach, in theory at least, is what we call the *representative positions problem*. A set of positions taken from beginners' games will be very different, in general, from a set of positions from expert play, and so the probability distributions may also be expected to differ. Baum and Smith appear to have discovered this problem empirically. Their solution is to use a two stage procedure. Firstly, the probabilistic element of their algorithm is disabled, and games are played using only a static evaluation function. Positions from these games are then used to calculate distributions which are, in turn, used to gather another set of positions. The

[25]The number of training games required may be adjusted by dividing the space of game positions up into a number of bins, according to the number and granularity of these criteria.

final probability distributions are then calculated with reference to the second set of positions.

The fact that their description of this was relegated to a footnote suggests that they did not find this to be a serious problem. They do not mention why they stopped after two steps of this procedure — whether convergence had been observed, or whether they were concerned about the possibility of degeneration. Position evaluation is relatively easy for the games treated by Baum and Smith (Kalah, Warri and Othello). For a game such as Go, for which position evaluation is notoriously difficult, the representative positions problem may prove a serious difficulty. Certainly, the theoretical problem of how to 'bootstrap' a set of positions that are representative of how the algorithm *will* play — when it has used the evaluation function derived from that set — remains unaddressed.

6.5 Utility

The reason why the game tree is searched is that by application of the evaluation function, it is possible to distinguish the good from the bad branches, and so guide both play and search.

For the purposes of guiding search, a static score is too simple a description of what search may reveal; a simple ordering of positions is not sufficient to measure the importance of searching a node because of the complicated effect of interactions with the scores of other nodes. We therefore require a richer representation — i.e. a probability distribution — to represent belief about the possible consequences of search.

For the purposes of guiding play, a reasoned choice of move requires that an ordering of positions exist. If the positions are scored with probability distributions, the natural choice is its expectation. The only result of ultimate interest is the outcome of the game, and so if the expectation of the probability distribution is to be used in this way, it is essential that the scores be calibrated to the expected reward from playing the game. The MGSS* and BP algorithms carry out this calibration as explained in the previous section. The failure of the PSVB* and Probabilistic B* algorithms to carry out such a procedure is an omission no less serious just because they do not use expectation to choose between moves.

It is not as simple as is generally believed to calibrate position evaluation to something that might appropriately be referred to as 'utility' (i.e. probability of winning[26]), because of the complications raised in Chapter 2. Since the whole notion of attaching a win probability to a particular position is a simplification, problems are bound to occur if an attempt is made to exactly define it in practice.

The notion of 'utility' is central to the MGSS* and BP algorithms. They both make the simplifying assumptions that time cost is separable from the position to which it is applied, and from the opponent against which they play. Using **L** to stand for search information, they assume the following:

[26]The rest of this section makes the simplifying assumption that there are two only outcomes, with rewards 0 and 1. This does not restrict the generality of the discussion.

$$U(\mathbf{L},\ \tau_{\text{Us}},\ \text{Opponent},\ \tau_{\text{Opponent}},\ \mathbf{L}_{\text{Opponent}}) = V(\mathbf{L}) + V(\tau_{\text{Us}})$$

The fact that the opponent's search information, $\mathbf{L}_{\text{Opponent}}$, is unknown is clearly a strong practical argument for ignoring it in the calibration. Similarly, the case for ignoring the opponent in the calibration process is supported by the impracticality of having a separate training set for each opponent, or the difficulty and degree of approximation required to determine statistics able to summarise the range of possible opponents. Eventually, programs will benefit from calibration according to both the opponent and inference about the opponent's search information. In the meantime, the practical consequence from not taking into account the opponent's strength when calibrating the utility function is that the program will play inappropriately against an opponent of greatly different ability[27]. This is not a cause for great concern, since the primary aim of current research into computer game-playing is to devise programs that play as strongly as possible in even games.

The easiest approximation to address in the calibration process is the complete neglect of the opponent's time. Since this is readily available, to completely ignore it may seem rather slack but is in fact so standard as to

[27]For example, suppose a strong Chess program offered a beginner a queen's odds. If the utility function did not take the opponent into account, it would suggest that a loss was almost certain. This would cause a problem similar to the horizon effect; the program would play increasingly desperate and risky moves, in the hope that it had underassessed its chances. By contrast, a human expert would expect the beginner to make simple errors, and play accordingly.

pass without comment in the discussion of computer game-playing[28]. Nevertheless, it is of use in determining human play, a fact which will be agreed by any games player who has experience of playing under serious time controls. The most immediate use of this information would seem to be its (cautious) application to the area of time control, where it could be added in as another factor to the rather ad hoc algorithms that are currently the norm. It seems reasonable to vary the value of time in sympathy with the amount of time the opponent has. The justification for this is that if the opponent has a lot of time, this gives him the potential to think deeply about the position and so cause complications (by playing trick moves etc.) which will cause us to run short of time. Conversely, if the opponent has very little time, then either he is in time trouble — which suggests we should look for such complications — or else he has correctly judged that he will not need much more time, in which case we have probably been too frugal up to now if we have a lot of time left.

A more robust use of the opponents' time would be to take it into account when calibrating the evaluation functions. Attempting to equate a board position with a win probability ignores the aspect of time control altogether, limiting the effectiveness of time control. A more thorough approach would use a population of games to derive a function $U(\mathbf{L}, \tau_{\text{Us}}, \tau_{\text{Opponent}})$, which might then be used for time control as well as move selection. The modelling process can be aided by certain theoretical knowledge of the properties of

[28]I am not aware of any game-playing programs which use this information.

$U()$. We require a smooth function which is increasing in τ_{Us}, decreasing in τ_{Opponent} and — drawing upon the thoughts presented in Section 5.2 — that is subharmonic in τ_{Us} and superharmonic in τ_{Opponent}. From a theoretical point of view such a reasoned approach to time control would be greatly preferable to the current crude approximations; from a practical point of view, it would require a considerable increase in the computing power needed to calibrate the utility function, would lead to a more complicated program and might decrease the speed slightly, so computer game-playing may take some years to reach the point where it becomes a priority.

Of the game-playing algorithms devised so far, BP has come closest to defining a satisfactory model of utility. Inevitably, some sacrifices have been made to expediency. One of these is the oversimplified time control mechanism highlighted above, which, by the authors' own admission has 'an annoying sickness' [80].

Another shortcoming of BP is the strategy for leaf expansion. The 'gulp' idea is an important way of cutting down meta-calculation costs. However, the notion of 'gulp size' is a rather imperfect implementation of it. No justification is presented for why the gulp should be a fixed proportion of the information, much less a constant one. If QSS really is a good approximation to 'utility', a desirable goal would be to expand those leaves with the largest QSS values, *throughout the entire game.* This would suggest that an appropriate mechanism of leaf selection would be to expand in one gulp all those leaves with a QSS value above a certain threshold. This threshold should be dynamic, changing in response to the findings of search. If time ran short, if the game looked like taking longer than expected, or if the position suggested that there would be many leaves with above average QSS ahead, the threshold should increase. Conversely, it would decrease in the opposite circumstances — the guiding principle should at every stage be the aim of maximising expected 'utility/unit of search'.

6.6 Trends in Computer Game-Playing

At the risk of stating the obvious, I hope that the reader is at this point completely convinced of the importance of quantifying the uncertainty around a position's static evaluation. This remark is by no means as trite as it may appear to the reader with a statistical background, since this realisation has taken the artificial intelligence community a very long time[29]. Almost all the top game-playing programs have no such element, being basically alpha-beta search engines with a selection of refinements such as those reviewed in Section 1.1.

The explanation for this is simple. Research into game-playing has been largely empirically-driven, and algorithms have not been developed *in vitro*. The meta-calculation costs associated with the processing of probability distributions are very considerable as compared with point estimates, while

[29]If the research published on tree search and game-playing is any indication, it is only just becoming widespread, almost 50 years after the notion of computer game-playing was first mooted by Shannon [76].

considerable computing power is required to realise the benefits of greater selectivity. This alone, apart from their greater intricacy, is sufficient to explain why probabilistic selective search techniques have not, in the past been in the mainstream of research. If they *had* been developed twenty years ago, they would not have held their own against the sheer speed of non-selective search algorithms.

One notable exception to the early concentration on 'brute force' methods was the work of the statistician Good. His 1968 "Five Year Plan for Automatic Chess" [30] would more realistically have been entitled a "Twenty Year Plan"; even in the late 1980's, when Russell and Wefald introduced the MGSS* algorithm, they found it only to be comparable with fixed-depth alpha-beta search[30]. The recent results of the BP program seem very encouraging, and so it seems as if, another 10 years on, selective searching has finally come of age. Since the advantage of being selective increases with the amount of computing power available, it will surely not be long before even the most powerful alpha-beta program, running on specialist hardware, will succumb to the greater intelligence of game-playing algorithms based on selective search. If the reader will indulge me in a little prediction, I suggest that by 2020, all computer game-playing programs will be Bayesians.

[30]Baum and Smith [6] have suggested that even this may have been an overoptimistic assessment of its performance.

6.7 Conclusion

This work has attacked the problem of computer game-playing from both ends. At one extreme are the models introduced, which have a provably optimal solution, at least in some circumstances. At the other, I have included some musings about the problem which inspired their development, that of how a statistically-based game-playing program might work. As we have seen, this problem does not have an optimal solution — in the classical sense — so while I may be disappointed with the size of the gap between the two extremes of study, I make no apology for its existence. I do hope that my work has gone some way towards enlightening the reader about what may be achieved by wider application of dynamic stochastic control methods to the tasks of tree search and game-playing.

It seems appropriate to conclude with the remark that, both as a games player and as a researcher, I find it a reassuring thought that it is impossible to define an 'optimal' strategy for game-playing, and so this problem is one which will remain permanently open.