On Optimal Game-Tree Search using Rational Meta-Reasoning

Stuart Russell and Eric Wefald*
Computer Science Division
University of California
Berkeley, CA 94720

Abstract

In this paper we outline a general approach to the study of problem-solving, in which search steps are considered decisions in the same sense as actions in the world. Unlike other metrics in the literature, the value of a search step is defined as a real utility rather than as a quasi-utility, and can therefore be computed directly from a model of the base-level problem-solver. We develop a formula for the expected value of a search step in a game-playing context using the *single-step* assumption, namely that a computation step can be evaluated as it was the last to be taken. We prove some meta-level theorems that enable the development of a low-overhead algorithm, MGSS*, that chooses search steps in order of highest estimated utility. Although we show that the single-step assumption is untenable in general, a program implemented for the game of Othello soundly beats an alpha-beta search while expanding significantly fewer nodes, even though both programs use the same evaluation function.

1 Introduction

The RALPH (Rational Agent with Limited-Performance Hardware) project is a long-term research effort aimed at understanding the effects of finite computational resources on the performance of, and the design of algorithms and architectures for, artificially intelligent systems. Rather than viewing AI as the production of human-like or commercially valuable programs, or as the design of systems containing only true beliefs, we see the AI problem as that of designing systems capable of achieving optimal behaviour (in a decision-theoretic sense) under computational constraints in complex task environments. Such a constrained optimization problem may have very different solutions from the computationally unconstrained case, for which a pure logical (or, more generally, decision-theoretic) implementation may be appropriate. We therefore view the problem of finite resources for real-time behaviour as *central* to AI, rather than something that necessitates undesirable approximations to a 'perfectly rational' agent.

*This research was carried out with support from the AT&T Foundation and the Computer Science Division of the University of California, Berkeley. The second author is supported by a Shell Foundation Doctoral Fellowship.

In this paper, we outline the application of decision theory to the problem of reasoning about which computations to perform. The theory can be used to analyze the rationality of bounded agents, or to design optimal algorithms for real-time situations — situations in which the utility of an agent's actions depends significantly on the time at which they are carried out, rendering perfect deliberative rationality inoperable. We apply the theory to design a game-playing algorithm that selects nodes to expand according to the expected utility of the expansion, and can beat alpha-beta with the same evaluation function while searching fewer nodes.

2 Optimal allocation of computational resources

We are studying the extent to which situation-dependent control of reasoning can benefit an agent operating in real time. In such an agent, choices concerning which computation to carry out, if any, must be made by *meta-level* computations. This, of course, leads to the potential for an infinite regress of meta-levels. This regress can be avoided by, among other methods, the use of approximate decision-making (the outcome of which is not guaranteed to be optimal) at some point in the hierarchy, or by carrying out an unbounded computation at design time. For the tasks faced in AI the variety of situations and time pressures suggests that a combined approach is appropriate.

The basic proposal here is quite simple: choose computations in the same way that any other actions are chosen. Decision theory is the standard normative theory for actions, so let's apply it to computations too. This idea, although similar to the concept of *information value* in decision science [RaifTa and Schlaifer, 19(51, Howard, 1966, Good, 1968], is relatively new in AT. In addition, getting the model right and putting it into practice are not such easy tasks.

In this section, we give a general formula for the *expected value* of a computation. The following section discusses ways in which the formula must be modified to make it usable by a limited rational agent that has only *estimates* of utilities and their expectations. Subsequent sections develop the analysis needed, and the simplifying assumptions employed, to create a game-playing program along these lines.

Notation: The agent's goal is to maximize its *utility function* $U(w)$ on states of the world w. We denote the outcome of an action A performed in a situation W_j by $[A, W_j]$, or simply $[A]$ if performed in the "current" situation. At any given time the agent has a default base-level action, α, which is the action that currently appears best to the agent. At the same time it has available to it a set of possible computational actions S_j, with which it can continue its deliberations. Computation *Sj* might lead the agent to revise its choice of default action; the new default action will be called αs_j The meta-level decision problem for the agent is thus to decide among the immediate options a, $S_1, \ldots, S_k$.

Computational actions are distinguished from base-level actions in that they directly affect only the agent's internal state. We define the *net value* of a computational action to be the resulting increase in utility, compared to the utility of taking the default action instead:

$$V(S_j) = U([S_j]) - U([\alpha]) \tag{1}$$

If *Sj* is a "complete computation", i.e., is not followed by any further deliberation, then the utility of state *[Sj]* is just the utility of the action αs_j chosen as a result of the computation,[1], given that it is to be performed *after* the computation is complete, i.e., in the state *[Sj]:*

$$V(S_j) = U([\alpha_{S_j}, [S_j]]) - U([\alpha]) \tag{2}$$

In the general case, the computation will bring about changes in the internal state of the agent that will affect the value of possible further computational actions. We call such a computation a "partial computation". In this case, we want to assess the utility of the internal state in terms of its effect on the agent's ultimate choice of action. Hence the utility of the internal state is the *expected* utility of the base-level action which the agent will ultimately take, given that it is in that internal state. This expectation is defined by summing over all possible ways of completing the deliberation from the given internal state. We denote a computation sequence consisting of computation S_1 followed by computation S_2 by the expression $S_1.S_2$. Thus, letting T range over all possible complete computation sequences following Sj, and α_T represent the action chosen by computation sequence T, we have

$$U([S_j]) = \sum_{\mathbf{T}} P(\mathbf{T}) U([\alpha_{\mathbf{T}}, [S_j.\mathbf{T}]]) \tag{3}$$

where P(T) is the probability that the agent will perform the computation sequence T.

3 Estimating the Value of Computation

How should a *limited* rational agent use these equations to evaluate its own computational actions? The agent cannot simply apply equation 3 directly, because it does not know the exact utilities of its base-level actions. (If it did, it would not need to deliberate at all.) We will assume the agent makes a numerical estimate U([A]) of the utility of its available actions, and that deliberation proceeds by revising these estimated utilities. At the end of its deliberations it picks the action with highest estimated utility. This model applies to, or can be adapted to, most search algorithms in AI. Since it is important to distinguish the different utility estimates as computation proceeds, we will use S to denote the base-level computation sequence carried out up to the current state. Then Q^S represents an estimate of a quantity *Q* calculated by computation S. Thus the *estimated utility* of a computation *Sj* is given by:

$$\hat{U}^{\mathbf{S}.s_j}([S_j]) = \sum_{\overline{\mathrm{T}}} \hat{P}^{\mathbf{S}.s_j}(\mathbf{T}) \max_i \hat{U}^{\mathbf{S}.s_j.\mathbf{T}}([A_i, [S_j.\mathbf{T}]])$$

Here A_i ranges over all of the possible base level actions in the current state; hence α_T is just the action which maximizes US.sj.T in the state [Sj.T].

Finally, the estimated net value of computation *Sj* is given by

$$\hat{V}(S_j) = \hat{U}^{\mathbf{S}.s_j}([S_j]) - \hat{U}^{\mathbf{S}.s_j}([\alpha]) \tag{5}$$

Of course, before the computation *Sj* is performed, *V(Sj)* is a random variable. Although the agent can't know ahead of time what the exact value of *V(Sj)* will be, given sufficient statistical knowledge of the distribution of *V* for similar actions in past situations, the agent can take its *expectation,*

$$E[\hat{V}(S_j)] = E[\hat{U}^{\mathbf{S}.s_j}([S_j]) - \hat{U}^{\mathbf{S}.s_j}([\alpha])] \tag{6}$$

Such an agent can thus employ *meta-level rationality* in its control decisions. The principle of rationality—choose the action with highest estimated expected utility—applied to the meta-level decision problem is equivalent to:

1. Estimate the expected value *E[V(Sj)]* of each computational action *Sj.*
2. If any computational action has positive expected value, take the one with highest expected value. Otherwise, cease computing and take the current base-level action, *α,* with highest expected utility.

Thus far we have captured the real-time nature of the environment by explicitly including the situation in which an action is taken in the argument to the utility function. Such a comprehensive function of the total state of affairs captures all constraints and trade-offs; in particular, any form of time constraint can be expressed in this way. However, the inclusion of this dependence on the overall state significantly complicates the analysis. Under certain assumptions, it is possible to capture the dependence of utility on time in a separate notion of the *cost of time,* so that the consideration of the quality of an action can be separated from considerations of time pressure.

In applications, typically we are given some value estimator that is independent of time and takes just the action outcome as argument. We will call such a function the estimated *intrinsic utility,* denoted by *Uj.* We can[1] then define a function *TC* presses the difference between total and intrinsic utility, assuming this difference depends only on the temporal duration |Sj| of the computation:

$$\hat{U}([A_i, [S_j]]) = \hat{U}_I([A_i]) - TC(|S_j|) \tag{7}$$

Consider again the case of a complete computational action *Sj,* where *Sj* results in taking action αSj, where

[1]We ignore learning as a means of generating utility for future situations.

α was the default action prior to S_j. In this case, the estimated net value of S_j is given by

$$\hat{V}([S_j]) = \hat{U}_I^{\mathbf{S}.S_j}([\alpha_{S_j}]) - \hat{U}_I^{\mathbf{S}.S_j}([\alpha]) - TC(|S_j|) \quad (8)$$

Note that we use the later estimate $\hat{U}_I^{\mathbf{S}.S_j}$ to evaluate both α_{S_j} and α, the new and the old best moves. The reason for this is most obvious if we consider the case where $\alpha_{S_j} = \alpha$, but $\hat{U}_I^{\mathbf{S}.S_j}([\alpha_{S_j}]) > \hat{U}_I^{\mathbf{S}}([\alpha])$; i.e., S_j simply revises upward the estimated intrinsic utility of α but does not alter the choice of move. Then the net value of S_j should depend only on the passage of time spent in deliberation, since the real intrinsic utility is of course constant. An important corollary of this is that if $\alpha = \alpha_{S_j}$ then $V(S_j)$ is guaranteed to be negative, that is *a search action that doesn't change the system's preferred move will have had no value.*

This points to deep issues which arise from the fact that the agents we are dealing with have only limited rationality. For instance, in [Russell and Wefald, 1989], we discuss differences between our formula and R. Howard's [1966], in which Howard analyzes the expected value of obtaining complete information about certain variables in a bidding problem. Briefly, Howard assumes that the *expected* value of the error term $\hat{U}_I^{\mathbf{S}.S_j}([A]) - \hat{U}_I^{\mathbf{S}}([A])$ for a given action A is strictly 0. This is true for a perfectly rational agent, but cannot be assumed always to hold exactly for a limited rational agent, for instance one who uses a polynomial evaluation function to estimate expected move utilities.

4 Specific base-level problem-solvers

Up to this point, we have been working at a very general level, making no assumptions about the nature of the base-level decision-making mechanism. The above equations are applicable to a brain or a pocket calculator, in principle. Naturally, there are some attributes of certain mechanisms that make them amenable to meta-level control. The overall computation should be modular, in the sense that it can be divided into 'steps' that can be chosen between; the steps must be able to be carried out in varying orders. The steps can of course have any grain size.

We now look at some particular systems in more detail.[2] We will focus on the use of the complete-computation formula, equation 8. This equation gives the net benefit of the computation as a random variable. In order to compute its expectation, we need information about the distribution of possible effects of a computation on the system's value estimates for its available actions.

In a search program, computation typically proceeds by expanding 'frontier nodes' of the partially-grown search tree. The value estimates for actions are calculated by backing up values from the new leaf nodes through their parents. The effect on the value estimate for an action therefore is composed of two aspects: 1) the effect on the value of the leaf nodes that are expanded, and 2) the transmission of this effect back through the tree.

[2]Space limitations prevent more than a cursory justification of the results. For more detailed analysis, see [Russell and Wefald, 1988b].

The first component depends on the nature of the node being expanded, and the nature of the expansion computation. Statistical information on the probability distribution can therefore be acquired by induction on a large sample of similar states using the same type of expansion computation. The second component is an analytic function of the state of the tree (in particular, the current value estimates for the nodes on the path to the root, and their children) that depends on the backing-up method used by the search program.

In this paper, we consider systems that make decisions using simple search algorithms. In order to make calculation of the expected value of computation tractable, we will introduce explicit simplifying assumptions. These are related to what Pearl [1988] has called a "myopic policy". The notation used will be as above, with the addition of B1,B2, etc., to denote current second-best action, current third-best, and so on.

Meta-greedy algorithms

Explicit consideration of all possible complete sequences of search steps is clearly intractable. The obvious simplification is to consider *single* computation steps (remember that a single step can represent an arbitrary amount of computation, but without interleaved control), and to estimate their ultimate effect; we then choose the step appearing to have the highest benefit. Such *meta-greedy algorithms* effectively have a fixed meta-meta-level policy of a depth-limit of 1 on the meta-level decision problem.

Single-step assumption

The expression for the estimated utility of a computation (equation 4) can be used directly with a meta-greedy algorithm, but it is often somewhat hard to evaluate, particularly when the future availability of computational resources is hard to estimate. The computation is greatly simplified if we assume that it is reasonable to act as if we will take at most one more search step. In this case we can use the complete-computation formula 8. We call this the *single-step assumption.* The assumption sometimes fails. Recall that a complete computation has no value unless it changes the choice of move; thus the single-step assumption will predict that certain search steps can have no value, whereas often those steps enable other steps to become valuable. We will see that, although it makes the expected-value computation tractable, this assumption also places certain limitations on the depth of search in some domains, including game-playing.

It is worth emphasizing here that if either the meta-greedy or single-step assumptions were completely relaxed, the other would become completely true. That is, if it were possible to consider all possible sequences of computations, then it would be no restriction to consider those sequences as complete computations. On the other hand, if we could accurately compute the full expected value of a single computation step considered as a possibly partial computation, we would thereby be implicitly evaluating all possible continuations of that computation step, and hence we would not need to consider explicitly all possible computation sequences. Thus, the simplification lies in employing the two assumptions jointly; neither alone would be a restriction.

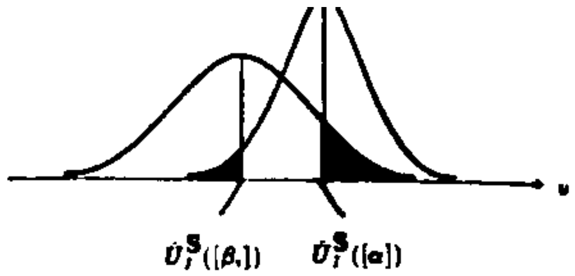
$\hat{U}_I^{\mathbf{S}}([\beta_1])$
$\hat{U}_I^{\mathbf{S}}([\alpha])$
u

Figure 1: Effects of search actions

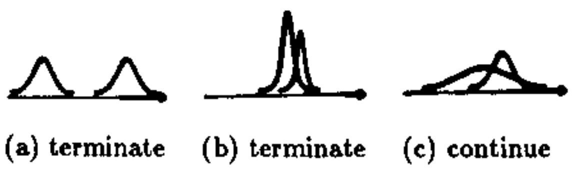

Figure 2: Three basic situations

5 Error: The variation of estimated utility with search

According to equation 4, calculating the value of a computation S_j requires knowledge of how the estimated utility of the top-level moves A_i will change as a result of S_j. We will use the term *error* for the difference $\hat{U}_I^{\mathbf{S}.S_j}([A_i]) - \hat{U}_I^{\mathbf{S}}([A_i])$.

Let $p_{ij}(u)$ be the probability density function in the current state for the future value of the utility estimate $\hat{U}_I^{\mathbf{S}.S_j}([A_i])$ after the search action S_j has been carried out. We assume that S_j is the expansion of a single leaf node, and that the search graph is a tree, so that the backed-up value of at most one top-level move will be affected by S_j. Hence $\hat{U}_I^{\mathbf{S}.S_j}([A_i]) = \hat{U}_I^{\mathbf{S}}([A_i])$ if S_j does not expand a node in the subtree under move A_i. A search action can have positive value by changing the preferred move in only two ways: by raising the value of some move β_i above the current value of α, or reducing the value of α below the current value of β_1 (see figure 1).

If we are considering a search action S_j which affects the estimated utility of move β_i, this will only change our move choice if $\hat{U}_I^{\mathbf{S}.S_j}([\beta_i]) > \hat{U}_I^{\mathbf{S}}([\alpha])$. In that case, we expect to be better off by an amount $\hat{U}_I^{\mathbf{S}.S_j}([\beta_i]) - \hat{U}_I^{\mathbf{S}}([\alpha])$. Thus

$$E[V(S_j)] = \int_{\hat{U}_I^{\mathbf{S}}([\alpha])}^{\infty} p_{ij}(x)(x - \hat{U}_I^{\mathbf{S}}([\alpha]))dx - TC(|S_j|) \tag{9}$$

In the other case, if S_k affects the move α, following similar reasoning we have:

$$E[V(S_k)] = \int_{-\infty}^{\hat{U}_I^{\mathbf{S}}([\beta_1])} p_{\alpha k}(x)(\hat{U}_I^{\mathbf{S}}([\beta_1]) - x)dx - TC(|S_j|) \tag{10}$$

These formulæ allow the straightforward calculation of the optimal search action under the meta-greedy and single-step assumptions, and of the appropriate stopping point, given the distributions p_{ij} and expected utilities $\hat{U}(A_i)$, and a cost estimation function $TC(|S_j|)$. Below, we show how these quantities can be calculated for a game-playing system in terms of easily obtainable, statistical quantities.

We can describe the *qualitative* behaviour of any algorithm based on our approach. There are two cases where further search will have very low expected value: when there is little overlap between the distributions of the top two moves, and when the moves overlap a great deal but the distributions are narrow. (This latter case has received scant attention in the literature.) We illustrate the three major situations graphically in Figure 2.

6 Game-playing: The MGSS* algorithm

In this section, we describe how the utility of a computation can be estimated for two-player games, in which the basic decision-making algorithm is some kind of lookahead search using minimax backup. The choice of search steps is then made so as to maximize the system's overall utility, as determined by the move chosen and the time at which it is made. A system with low-overhead, normative control of search would be guaranteed to win, on average, against any other program with equal computational resources and domain-specific knowledge. We believe such a system can be achieved. In addition, we obtain qualitative insight into the nature of efficient search.

We assume a standard evaluation function that is independent of time factors, and we therefore use the time cost function described above. The evaluation function is viewed as an intrinsic expected utility function, not as an *estimate of the true value* of the position. The true value for a game such as chess is 0 or ±1 (on a scale in which 1 is a win); the estimated utility is a computationally-bounded, probability-weighted value p_{win} — *Ploss* [Good, 1968].

In our analysis [Russell and Wefald, 1988b], we use both the meta-greedy and single-step assumptions, with individual node expansions as the chosen computational steps. We have developed an efficiently computable formula for evaluating the integrals in equations 9 and 10. The principal tool is the re-expression of the probability distributions p_{iJ} in terms of the error distribution *pjj* at the leaf node affected by the search action Sj,. Since *pjj* is a function of only the board situation, and not of tne game tree, it can be estimated from empirical data on the results of previous tree searches. A computable expression for the integral is then obtained by analyzing how the change in value of the leaf node is propagated to the top level.

The proof of correctness of the following equations, corresponding to equations 9 and 10, depends on a theorem proved in [Russell and Wefald, 1988b] that gives a simple recursive definition for the nodes that are *Relevant* to a top-level move, that is, nodes whose expansion can affect the top-level move's value (cf. singleton conspiracies in [McAllester, 1988]). The value of expanding any other node is strictly 0, and our algorithm derives considerable efficiency from restricting its attention to only relevant nodes.

The expected value of expanding a leaf node *j* in the subtree of a move β_i, previously given by equation 9, is

$$E(\Delta(S_j)) = \int_{\delta}^{\infty} p_{jj}(x)(\delta - \hat{U}_I^{\mathbf{S}}([\alpha]))\,dx + \int_{\hat{U}_I^{\mathbf{S}}([\alpha])}^{\delta} p_{jj}(x)(x - \hat{U}_I^{\mathbf{S}}([\alpha]))\,dx \quad (11)$$

where p_{jj} is the density function of the random variable $\hat{U}_I^{\mathbf{S}.S_j}(j)$, and δ is the lowest value of the second-best children of the min nodes between j and the root. Similarly, the expected value of expanding a leaf node k under α is given by

$$E(\Delta(S_k)) = \int_{-\infty}^{\delta} p_{kk}(x)(\hat{U}_I^{\mathbf{S}}([\beta_1]) - \delta)\,dx - \int_{\delta}^{\hat{U}_I^{\mathbf{S}}([\beta_1])} p_{kk}(x)(\hat{U}_I^{\mathbf{S}}([\beta_1]) - x)\,dx \quad (12)$$

where p_{kk} is the density function of the random variable $\hat{U}_I^{\mathbf{S}.S_k}(k)$, and δ is the highest value of the second-best children of the max nodes between k and the root.

7 Implementation of MGSS*

We have used the game of Othello for our experiments in this domain. In a direct implementation of the analysis of the previous section, the entire search tree is kept in memory, and the tree is grown one step at a time by choosing, at each step, a tip node to expand, and adding its successors to the tree.

The following information is maintained for each node:

1. a link to the top-level move, if any, to which the node is *Relevant;*
2. the *SearchValue* of the node, i.e., the expected gain in utility from expanding the node, for leaf nodes;
3. the *Game Value* of the node, i.e., its *StaticValue* if it is a leaf node, or its current backed-up value otherwise;
4. the *6* value for the node, as defined above;
5. a link pointing to the *Parent* of the node; and
6. if the node has already been expanded, a pointer to a list containing the node's *Children.* The list is maintained in order of *GameValue.*

Relevant leaf nodes with positive *Search Value* are maintained in a *Queue,* in decreasing order of *SearchValue.*

Algorithm MGSS*

```
1. Generate the successors of the Root. Place the Relevant ones in the Queue ordered by
   SearchValue. For each successor, set its GameValue equal to its StaticValue. Place the
   successors in the Children list of the Root ordered by GameValue.
2. Remove the first element j of Queue. Compute E(A(Sj)) using equation 11 or 12. Estimate the
   time-cost TC of expanding node j. If $E[\Delta(S_j)$ ~ TC] < 0 then return the first element
   in the Children of the Root as the best move.
3. Otherwise
   (a) Carry out the computation Sj;. For each resulting leaf node, set GameValue = StaticValue.
   (b) Place the new leaf nodes, ordered by GameValue, in the Children list of j.
   (c) Back up the GameValues of the successors to j's GameValue. If this changes, re-insert j
       in its parent's Children list and continue backing up recursively, stopping at the Root.
       Whenever a Children list is re-ordered, or the best move in such a list increases its
       value, or the second-best move in such a list decreases its value, recompute the
       appropriate S values and Relevant node pointers. The latter step may involve updating
       Queue membership.
   (d) Add j's Relevant successors, ordered by SearchValue, to the Queue.
4. Go to 2.
```

All this means that the overhead for controlling the search is small, provided that the integral expression in equations 11 and 12 can be evaluated quickly and exactly. We now show how this is done.

The leaf-node error distributions (the functions *pjj* in the integral) are generated by prior statistical sampling. For search steps consisting of a single node expansion, such as are used in MGSS*, the error is simply defined as the difference between the leaf's static value and its backed-up value from a depth one search. The data points are gathered into buckets according to selected features of the board situation. 35,000 data points were gathered into roughly 1000 buckets. A different error density function is created for each bucket. The functions appear to be normal curves to a reasonable degree of approximation, and can thus be represented by two parameters, the mean u and standard deviation *a.*

Given normal curves *Nua* for the density functions *pjj* and *pkk,* we can simplify equations 11 and 12 considerably, after some mathematical effort. The new equations

$$\begin{aligned}\Delta(S_j) &= \sigma\left(N_{0,1}\left(\frac{\hat{U}_I^{\mathbf{S}}([\alpha]) - \mu}{\sigma}\right) - N_{0,1}\left(\frac{\delta - \mu}{\sigma}\right)\right)\\ &+ (\hat{U}_I^{\mathbf{S}}([\alpha]) - \mu)\Phi\left(\frac{\hat{U}_I^{\mathbf{S}}([\alpha]) - \mu}{\sigma}\right)\\ &- (\delta - \mu)\Phi\left(\frac{\delta - \mu}{\sigma}\right) + (\delta - \hat{U}_I^{\mathbf{S}}([\alpha])) \qquad (13)\end{aligned}$$

corresponding to equation (11), and

$$\begin{aligned}\Delta(S_k) &= \sigma\left(N_{0,1}\left(\frac{\hat{U}_I^{\mathbf{S}}([\beta_1]) - \mu}{\sigma}\right) - N_{0,1}\left(\frac{\delta - \mu}{\sigma}\right)\right)\\ &+ (\hat{U}_I^{\mathbf{S}}([\beta_1]) - \mu)\Phi\left(\frac{\hat{U}_I^{\mathbf{S}}([\beta_1]) - \mu}{\sigma}\right)\\ &- (\delta - \mu)\Phi\left(\frac{\delta - \mu}{\sigma}\right) \qquad (14)\end{aligned}$$

algorithm	wins	nodes	time(sec.)
MGSS*	16	108K	2163
α-β[2]	15	79K	761
MGSS*	22	190K	3847
α-β[3]	10	280K	2488
MGSS*	24	368K	10,037
α-β[4]	8	1,436K	13,264

Table 1: Summary of results for Othello

corresponding to equation (12). Here Φ is a tabulated integral of the normal curve *No,1* Armed with these formulae, computing search values is a good deal quicker than computing the static evaluation function, enabling MGSS* to perform with very low overhead.

The time cost estimation function can also in principle be determined empirically. Assume that the average time needed to perform a search step is, say, a millisecond. Then we want to determine empirically the effect of wasting a millisecond, in a given game situation, on the probability of ultimately winning the game. We then have a common utility scale for moves and time costs. In practice, it is sufficient to have *C* be an appropriate function of the number of seconds per move remaining, such that a loss on time is impossible. The appropriate sort of time-cost function will depend heavily on the particular rules under which the game is played, such as whether there is a per-move or only a per-game time limit, and so on. For the purpose of testing the implementation, we set a per-game time limit in terms of numbers of nodes expanded, and set the time cost to be an appropriately parameterized inverse function of the average time remaining per move.

8 Performance of MGSS"

The qualitative behaviour of the MGSS* algorithm is much like that described for the general decision-theoretic case, with two distinct classes of search termination points (see figure 2). The search is highly selective, with some branches reaching depths of 20.

Our experiments indicate that, with moderately small time allocations, MGSS* is significantly better than an alpha-beta search *using the same evaluation function, even though* MGSS* *generates significantly fewer nodes of search.* Our results vs. alpha-beta search to depths 2, 3, and 4 are summarized in table 1. For each search depth, the time cost function of MGSS* was adjusted to allow the algorithm to generate about as many nodes on average as alpha-beta; beyond depth 2, however, MGSS* chose to generate far fewer nodes. Each tournament consisted of 32 games played from 16 different starting positions, with the two algorithms alternately playing black. (One game against depth-2 a-/? was tied.)

Note that MGSS* is roughly even with depth-2 a-/3, and searches slightly more nodes; evidently, at this depth a-/? is wasting very little of its search effort. However, at greater effort limits, MGSS* plays significantly better, while doing much less search, in terms of nodes generated. In terms of time used, the performance of MGSS* is less impressive, using more time at depths 2 and 3, and slightly less at depth 4. It is important to emphasize that very little effort was made to optimize the implementation of MGSS* with respect to execution time, and we are confident that opportunities exist for considerable savings by improving the data structures used, replacing sorted lists with hash tables and binary search trees, and so on.

In terms of nodes generated, we expect further improvement with more work on the features used to classify the error density functions. The number of nodes searched can probably be further reduced by reducing the unit of computation to be the generation and evaluation of a single successor node, as in alpha-beta, rather than the expansion of a node. Our algorithm will also make better use of a more reliable evaluation function — as the errors tend to zero, the value of search will decrease, and in the limit the algorithm will only search to depth 1. An interesting possibility is that the evaluation function of BKG [Berliner, 1980] has effectively reached this level already: because of the dice rolls, the branching factor in backgammon at the root is 20 but at all other levels is close to 400, so that no current program uses a significant amount of search. Jeff Conroy [Conroy, forthcoming] has extended MGSS* to handle probabilistic outcomes, and initial results for backgammon indicate that additional search still pays significant dividends.

Although MGSS* seems extremely effective for small time allocations, the single-step assumption eventually begins to bar almost all nodes from being expanded as the tree grows larger. Hence comparisons against a much deeper-searching alpha-beta are unfavourable. Preliminary results indicate that MGSS* plays a slightly better than even game against depth-5 alpha-beta, while generating about 1/6 as many nodes of search. Against depth-6 alpha-beta, MGSS* appears to be incapable of generating large enough search trees to play effectively. Extension of the algorithm to consider sets of search steps may overcome this problem. We are also investigating the effect of applying MGSS* to *selectively extend* search beyond the depth limit of an alpha-beta search. A simpler form of this method appears in the work on *singular extensions* by Campbell [Campbell, 1988], and has proved surprisingly effective in the HITECH and Deep Thought chess machines.

9 Related work

A detailed comparison with related work on selective search appears in [Russell and Wefald, 1988b]. Here we can only point the reader to the pertinent literature.

Many sophisticated programs have been constructed for game-playing, where the infeasibility of search to termination has long been accepted as a fact of life. In some excellent early work, statistician I. J. Good [1968, 1977] proposed a decision-theoretic analysis both of move choice and search step choice, but the proposal was never implemented.

Berliner [1979] pointed out that alpha-beta undertakes needless search when only one legal move is available, or when one move can be seen to be better than all others even without search. His B* algorithm attempts to find the shortest 'proof, based on heuristic value bounds, that one move is best. Palay [1982, 1985] has given heuristics for choosing how to expand the tree to produce a 'proof quickly. In our terms, B* is using the wrong utility function at the meta-level, as illustrated by the case of symmetrical moves, where one should toss a coin rather than attempt to prove one better.

Rivest's Min/Max Approximation algorithm [Rivest, 1988] expands nodes that have the highest effect on the root value. This only weakly relates to the expansion's actual utility. McAllester's "Conspiracy Numbers" approach [McAllester, 1988] approach is more sophisticated, considering the set of nodes that must change their values to increase or decrease the root value beyond given bounds.

10 Discussion

The principal contributions of the paper are the following:

- The rational choice of search steps was defined and operationalized. Both meta- and base-levels should use the same utility function. This leads to improved choice of search steps compared to several other algorithms, and unifies pruning and termination in a natural framework.
- A class of meta-greedy algorithms was defined, using the single-step assumption, and a general formula for the value of search was developed in this context. The formula depends on a notion of error in evaluation that is both coherent and directly available from empirical data.
- Theorems were proved that allow for efficient implementation and for computation of search values using only simple, parameterized distributions.
- Good performance of the MGSS* algorithm, despite its restrictive assumptions, suggests that extensions of the approach to cover sequences of search steps may yield very high-quality algorithms that are also theoretically well-founded.

We also hope this work will help to reduce the gap between research on game-playing and that on more general types of decision-making.

As usual in computer science, justification is the prelude to synthesis, and we expect more new algorithms to appear as researchers prove various meta-level theorems concerning search action sequences which provide the basis for efficient implementations. Application to single-agent search [Wefald and Russell, 1989a] has yielded an algorithm with excellent real-time search behaviour, and a proof that best-first search is in fact optimal in the sense of performing the most valuable node expansions given the information available.

Several problems and research tasks remain. The implications of our approach for learning techniques are being investigated [Wefald and Russell, 1989b], as are its implications for the choice of backing-up procedure. We would like to apply the theory to provide optimal control of general decision-theoretic calculations, for example in an influence diagram system [Pearl, 1988]. However, the current analysis relies on the unit of computation only affecting the value of one top-level action, whereas refining a probability value, for example, might affect the value of several actions. Other extensions are needed to handle the case where time cost cannot be treated as independent of move chosen. The most serious theoretical problem is the need for the single-step assumption. The theory of'conspiracy numbers' developed in [McAllester, 1988] may be of some help here.

References

[Berliner, 1979] Berliner, H. J. (1979) The B* Tree Search Algorithm: A Best-First Proof Procedure. Artificial Intelligence 12.

[Berliner, 1980] Berliner, H. (1980). Backgammon computer program beats world champion. Artificial Intelligence, 14, 205-220.

[Campbell, 1988] Campbell, M. (1988) Singular Extensions: Adding Selectivity to Brute Force Searching. In Proceedings of the AAAI Symposium on Computer Game-Playing, Stanford, CA, 8-13.

[Conroy, forthcoming] Conroy, J. M. (forthcoming) Decision-theoretic control of search in probabilistic domains. MS Report, Computer Science Division, University of California, Berkeley, CA.

[Good, 1968] Good, I. J. (1968) A five year plan for automatic chess. Machine Intelligence, 2.

[Good, 1977] Good, I. J. (1977) Dynamic probability, computer chess and the measurement of knowledge. Machine Intelligence, 8.

[Howard, 1966] Howard, R. A. (1966) Information value theory. IEEE Transactions on Systems Science and Cybernetics, SSC-2(1), 22-26.

[McAllester, 1988] McAllester, D.A. (1988) Conspiracy Numbers for Min-Max Search. Artificial Intelligence, 35, 287-310.

[Palay, 1982] Palay, A. J. (1982) The B* Tree Search Algorithm—New Results. Artificial Intelligence 19.

[Palay, 1985] Palay, A.J. (1985) Searching with Probabilities. Marshfield, MA: Pitman Publishing Inc.

[Pearl, 1988] Pearl, J. (1988) Probabilistic Reasoning in Intelligent Systems: Networks of Probable Inference San Mateo, CA: Morgan Kaufmann.

[Raiffa and Schlaifer, 1961]
RaifFa, H, and Schlaifer, R. (1961) Applied Statistical Decision Theory. Cambridge, MA: Harvard University Press.

[Rivest, 1988] Rivest, R.L. (1988) Game Tree Searching by Min/Max Approximation. Artificial Intelligence 34.

[Russell and Wefald, 1988a] Russell, S. J., and Wefald, E. H. (1988a) Multi-Level Decision-Theoretic Search. Proceedings of the AAAI Symposium on Computer Game-Playing, Stanford, CA, 3-7.

[Russell and Wefald, 1988b] Russell, S. J., and Wefald, E. H. (1988b) Decision-theoretic control of search: General theory and an application to game-playing. Technical report UCB/CSD 88/435, Computer Science Division, University of California, Berkeley, CA.

[Russell and Wefald, 1989] Russell, S. J., and Wefald, E. H. (1989) Principles of Metareasoning. In Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Ontario: Morgan Kaufmann.

[Wefald and Russell, 1989a] Wefald, E. H., and Russell, S. J. (1989a) Estimating the Value of Computation: the Case of Real-time Search. AAAI Spring Symposium on AI and Limited Rationality Working Notes Stanford, CA, March, 1989.

[Wefald and Russell, 1989b] Wefald, E. H., and Russell, S. J. (1989b) Adaptive Learning of Decision-Theoretic Search Control Knowledge. Sixth International Workshop on Machine Learning, Ithaca, NY, June, 1989.