Decision-Theoretic Control of Reasoning: General Theory and an Application to Game-Playing [*]

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

October 11, 1988

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 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 appears to rival an alpha-beta search with equal node allocations or time allocations. Pruning and search termination subsume or improve on many other algorithms. Single-agent search, as in the A* algorithm, yields a simpler analysis, and we are currently investigating applications of the algorithm developed for this case.

[*]This research has been supported by an equipment grant from the AT&T Foundation. Wefald has been supported by a GE Foundation Fellowship and more recently by a Shell Doctoral Fellowship. We gratefully acknowledge this assistance.

Contents

Blot out vain pomp; check impulse; quench appetite; keep reason under its own control.

Marcus Aurelius Antoninus

1 Introduction

This report describes research in progress on the RALPH (Rational Agents with Limited Performance Hardware) project. The principal goal of this project is the design of robust software architectures for goal-driven intelligent systems operating under resource constraints, particularly time bounds, with the capability for self-adaptation to improve performance in a given task environment. A major tool of this research is a normative meta-level theory for the value of computations, since this allows an agent to allocate scarce computational resources optimally. Progress on developing such a theory forms the main subject matter of this report; the following remarks may help to put this work in context.

The study of resource-bounded intelligent systems promises to be a major area of research in AI in the near future, with the potential for drastic revision of our understanding of learning, inference and representation. The *logic-based* approach to AI emphasizes the ability to reach correct conclusions from correct premises. The *rational agent* approach, derived from philosophical and economic notions of rational behaviour, emphasizes maximal achievement of goals via decisions to act. The main criterion driving the design of such systems is the effectiveness of their behaviour; having true beliefs is a secondary goal. When resource bounds come into play, we have the case of a *limited* rational agent (LRA), for which infallible, open-ended inference is an unaffordable luxury. The solution to such a *constrained*-optimization design problem may look very different from that provided by the deductive model for the unconstrained problem. Our approach will be to address the design problem with the finitude of resources as a *starting point*, rather than trying to lop corners off the deductive model.

The problem of resource-bounded reasoning is usually studied under the heading of *real-time AI*, but has more general applicability than the class of problems usually studied in that field. The formal characterization of the problem situation is the following: the utility of a given action performed by the system exhibits a significant dependence on the time at which it is carried out. The dependence is usually such that complete solutions to the decision problem are infeasible. Typically, the utility of an action will be a decreasing function of time. Since the time at which an action can be carried out depends on the amount of deliberation required to choose the action to be performed, there will usually be a tradeoff between the *intrinsic utility* of the action chosen and the *time cost* of the deliberation. Standard algorithms either maximize intrinsic utility with little regard for the time cost, or minimize the time cost for achieving some fixed level of intrinsic utility. An assumption underlying this work is that considerably greater flexibility in the control of reasoning is required in order for systems to maximize their overall performance.

We envisage resource-bounded problem-solving as occurring in an *adaptive, decision-theoretic, meta-level architecture*. With this architecture, the system can perform explicit reasoning about the costs and benefits of the various computational actions available for solving a problem with varying degrees of solution quality and certainty. Below, we develop the foundations of a theory for estimating the value of any computation within the context of a given base-level decision-making system. The architecture is adaptive in the sense that the (sometimes expensive) meta-level reasoning can be compiled into efficient control policies for use in future problem solving, thereby amortizing the cost of control reasoning and allowing the system to converge towards an optimal configuration for its environment and goals. For this reason, the topics of meta-level reasoning and compilation need to be pursued together in a unified framework. However, these subjects are not discussed in this report.

2 Related work in resource-bounded problem-solving

This section discusses related work in several subfields. We first cover research on traditional "real-time AI" systems. We then discuss work involving explicit reasoning about the costs and benefits of computations, and work on architectures incorporating some kind of meta-level capabilities. A special subfield is that of selective search algorithms for game-playing and puzzle-solving. Lastly, we discuss the problem of computation cost from the point of view of theoretical computer science.

2.0.1 Real-time AI

Work in real-time AI has traditionally focused on delivering AI capabilities in applications demanding high performance and negligible response times. As a result, research has emphasized maximally efficient processing of inputs. Designers typically choose a fixed level of desired output quality, and then perform the necessary precompilation and optimization to achieve that level within a deadline. In a recent survey [54], Laffey defines real-time systems as follows: "The feature that defines a real-time system is the system's ability to guarantee a response after a fixed time has elapsed, where the fixed time is provided as part of the problem statement." In the large majority of the systems surveyed in their article, the required response time is relatively constant; given this view of real-time, the authors are led to the conclusion that "research which focuses on speeding up a version of the algorithm that can guarantee a response time should be given high priority." As Dean has pointed out [22], this approach, combined with algorithms that do not degrade gracefully as time bounds are reduced, leads to a research strategy based on compile-time optimization and variants of job scheduling. It seems unlikely that such an approach will generalize, particularly to cases of widely varying time pressures and problem complexities, or cases in which the need to act may arise on short notice. In addition, the 'deadline' model of time pressures is overly restrictive, since in reality there is almost always a continuous increase in the cost of time. A large number of application programs have been developed for real-time problems [3,5,6,16,17,18,19,30,31,33,40,42,50,58,62,67,68,76,79,81,87,89,93,94,98,101]. Many of these are discussed in detail in Laffey et al. [54]. The authors of the latter survey note, somewhat despairingly, that "Currently, *ad hoc* techniques are used for making a system produce a response within a specified time interval."

2.0.2 Limited rationality

It is a commonplace of artificial intelligence that perfect rationality, in the sense prescribed by decision theory, is unlikely to be computationally attainable by systems that explicitly solve the decision problem at each juncture. Simon made clear the distinction between systems that compute the rational thing to do (procedural rationality), and systems that simply do the rational thing (substantive rationality). Several articles on the subject of rationality appear in [90]. Substantive rationality, however, does not come for free. Although it means that an agent can be perfectly rational despite limited computational resources, it can only arise in one of three ways:

1. By design, where the *designer* possesses the computational and informational resources required to find optimal solutions.

2. By simple adaptation, that is, direct adjustment of behaviour in response to feedback from the environment.

3. By deliberative self-design, where the agent itself carries out the required computations, (perhaps incrementally) compiling them to ensure substantive rationality in future situations.

In non-trivial environments, particularly those with significant variation, it will be the case that exact solutions to the decision problem are intractable. Moreover, simple adaptation will be inadequate as a means of learning. With appropriate allocation of resources, however, it may be possible to approximate rationality to some degree. A premise of this research, therefore, is that flexible, autonomous systems in complex environments require the ability to reason explicitly about the appropriate resources to allocate to computation at any point, and about which computations will be most effective. This premise is shared by several other researchers in AI. Below we briefly describe the major current research projects in this area.

- Jon Doyle has been carrying out a long-term theoretical, perhaps even philosophical, investigation into the nature of rationality in real systems, under the heading of 'rational psychology' [26,27]. Using the principle that computations, or internal state changes, are actions to be chosen like any other actions, he has investigated the general notions of belief, intention and learning. In mainstream philosophy, Harman [39] has evinced a similar position concerning reasoning.

- Rational choice of computation has emerged as a topic of study in the field of medical decision-making for two reasons: firstly, because of earlier work on the value of performing tests to obtain information (there being an obvious analogy between performing tests and carrying out computations); and secondly, because of the need for high-stakes decision-making in real time, for example in intensive care units. Most work in this area derives from early work on the value of information by Howard [48] and Good [37]. Eric Horvitz [44,45] discusses maximizing the *comprehensive value* of a computation, referring to the utility of the action chosen by a computation taking into account the cost of the computation itself, and studies various tradeoffs and default policies in this light. He has also applied similar ideas to the problem of sorting under resource constraints [46]. Heckerman and Jimison [43] show how the amount of detail in the decision-theoretic formulation of a therapy problem can be varied according to the cost of complicating the model, the criticality of the decision and the degree of dominance of one course of action over another.

- Alice Agogino has been investigating the use of decision-theoretic modeling in the control of mechanical systems such as high-speed lathes and pumps. Using IDES (Influence Diagram Expert System) [2] she is able to carry out meta-level analysis automatically to decide, for example, whether to run a diagnosis routine, with associated downtime costs, or to continue running the machine under potentially suboptimal conditions.

- Tom Dean has been investigating several aspects of temporal reasoning [23] and real-time planning [21,22]. In the latter work he assumes a known variation of the intrinsic utility (see below) of the results of a computation method with the amount of resources allocated to that method, and uses this *utility profile* to allocate resources optimally in various resource-bounded scenarios. His principal contention is that real-time problem-solving requires the use of what he calls *anytime algorithms*, that is, algorithms whose quality of performance degrades gracefully as their resource allocation decreases.

- Mike Fehling and Jack Breese [32], like Agogino and Horvitz, have adopted a decision-theoretic approach to the control of reasoning and information-gathering, and describe a simple robot scenario in which a robot is faced with a choice between heading for one of two bridges without knowing which is usable, and taking a reconnaissance detour. They show how the optimal choice depends on the cost of time and on the probabilities of the various outcomes from reconnaissance.

- Bratman, Israel and Pollack, members of the Rational Agency (RatAg) working group at Stanford's Center for the Study of Language and Information, have proposed a system design that combines elements of decision theory with AI planning techniques [11]. They claim that the role of plans in a decision-theoretic framework is to limit the options to be considered at each juncture, thus keeping the decision problem tractable. Essentially, computation consists mainly of extending and instantiating a partial plan, or executing the plan's immediate recommendation subject to interrupts and plan failure monitoring.

- Vic Lesser and his group at the University of Massachusetts have studied real-time problem solving in some depth, particularly in the context of a distributed vehicle-tracking application [29,49,61,62]. Although the approach taken is primarily pragmatic, the principles adopted are very similar to those given below, particularly the definition of 'real-time' as denoting situations in which the utility of actions varies with time, rather than just simple deadline situations. Plans are viewed as expressions of prior control decisions, as in the RatAg work, and approximation is the main method for achieving flexible processing.

The research program proposed herein shares some of the ideas in the above efforts. Although detailed discussion is left to the following section, it is worth pointing out that this research differs in the following respects: it aims to integrate control reasoning with compilation to produce efficient execution architectures; it uses decision-theoretic evaluation of computations, in terms of their expected outcomes and utilities, rather than using meta-level condition-action rules; it computes the utility of computation directly in terms of the resulting change in the base-level action selected for execution, thereby embodying a *formal justification* for control policies; it emphasizes fine-grain control of reasoning, rather than a big-switch approach; it defines and integrates multiple forms of execution architectures using compiled knowledge. More globally, we take a system's internal utility function to have an empirical status, rather than to be a given, and we emphasize the use of defaults combined with continual adaptation for performance improvement as a route to avoiding the standard problem with reasoning about control, namely infinite regress.

2.0.3 Selective search

Selective search algorithms, typically employed in game-playing, use a simple but well-developed form of meta-level control. In these algorithms, the search tree is grown not using a situation-independent procedure such as breadth-first search, but rather by choosing, at each cycle, to expand the node that is 'most promising' according to some measure. This measure embodies the system's implicit meta-level preferences - the criteria according to which one computation is considered to be more valuable than another. A second facet of such algorithms is their *termination criterion*, which effectively embodies the system's meta-level preferences between time costs and computational benefits.

The simplest such algorithms are the best-first and branch-and bound procedures (including A*), which assess the desirability of *expanding* a node according to the desirability of a solution path that includes the node. We propose that this confuses two quantities: the intrinsic cost or value of the solution found, and the cost of finding it. In some cases, this confusion is harmless, but particularly in game-playing it can lead to irrational search policies. Part of the difficulty with using these algorithms in real-time situations is that they were developed for finding *optimal* solutions to problems before taking any action - essentially, the cost of search was considered to be a second-order concern. In realistic settings (and in the majority of games) only a limited amount of search can be done before committing to an action, usually well before the goal state is reached in the search. Korf [51,52] has been investigating the properties of search algorithms which commit to actions before reaching the goal state, and has found that acceptable performance can be obtained (in terms of the length of solution path found) with only a limited look-ahead horizon.

More sophisticated programs have been constructed for game-playing, where the infeasibility of search to termination has long been accepted as a fact of life. Some early work, that unfortunately does not seem to have been picked up by the AI community, was done by the philosopher/statistician I. J. Good [37,38]. Good proposed a decision-theoretic analysis both of move choice and search step choice, along with a comprehensive set of ideas for building high-performance chess programs. His suggestion for choosing search steps was for the system to estimate its expected utility before and after expanding a node, and to prune those nodes that did not appear worth expanding. The utility estimates were to be derived from the backed-up value of the moves viewed as best before and after the node expansion. As we discuss in more detail below, his suggestion contained a technical flaw but was certainly ahead of its time, and has reappeared in simpler forms in more recent work by McAllester and Rivest (see below). The mainstream of game-playing research and programming has been dominated by fixed-depth minimax search using alpha-beta pruning [53]. The alpha-beta heuristic is in effect a meta-level 'theorem' stating that a certain class of search actions have no value, on the assumption that the current leaf-node evaluations will not be changed as a result of further search (hence the utility of this heuristic in conjunction with fixed-depth search).

The rigidity of these simple meta-level policies was realized by Berliner [10], who pointed out that they result in 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.

Berliner's B* algorithm [10] represents a significant departure from minimax and alpha-beta. In B*, the single-valued static evaluation function is replaced by a pair of values, representing an upper and lower bound on the presumed "true" value of the node. The algorithm finds a proof that one first-level node is better than all others, by expanding the tree and backing up new upper and lower bounds, until the lower bound of one first-level node dominates the upper bounds of all the other first-level nodes. The algorithm can accomplish this either by attempting to raise the lower bound on the best node, called the "prove-best" strategy, or by lowering the upper bounds on all other nodes, called the "disprove-rest" strategy. Palay [77,78] has given heuristics for deciding between the two strategies. The choice can easily be formalized and computed within the decision-theoretic framework described below.

Although the line of research on B* algorithms reflects a number of useful insights into the search process, it does not incorporate explicit reasoning about the value versus the cost of search, in terms of its ultimate effect on move quality. Instead, it assumes that progress toward the goal of proving that one move dominates all others is always worthwhile, and seeks to maximize the rate at which such progress is made. 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.

Two other recent approaches are Rivest's Min/Max Approximation method [82] and McAllester's "Conspiracy Numbers" approach [69]. Both are iterative, selective-search methods. Rivest's method is based on the principle that the best node to expand at any given point in the search is that node whose expansion would provide the greatest amount of information. He suggests that the node which should be expanded under this principle is that node such that a small change in its value would cause the greatest change in the value at the root. He shows that the backed-up minimax value of a node can be approximated by a differentiable function, whose derivative can be maximized, giving the node on whose value the root value most highly depends, in the sense that the ratio between a given change in the node's value and the resulting change in the root's value is greatest. In general this node will simply be the frontier node reached by following the line of best play, other things being equal. However, the influence of any node at the frontier on the value at the root diminishes with depth, so that as the line of best play is expanded to greater and greater depth, other nodes eventually will come to be expanded. Thus, Rivest's method could perhaps be approximated by using a control-level evaluation function which estimates the worth of expanding a node to be an amount which increases with the closeness of the node to the line of best play, and decreases with the depth of the node. Such a function could be seen as a refinement of the implicit two-valued estimator used by alpha-beta. Below, we show that Rivest's algorithm can be led into fruitless searches because of the lack of correspondence between his principle of maximum information and the true value of a computation.

McAllester's approach is based on the notion of a "conspiracy". A conspiracy is a set of nodes that must change their values to increase or decrease the root value beyond given bounds. If we assume that a conspiracy of a certain size is unlikely, then we can put upper and lower bounds on the range of likely values that can be taken at the root. The tree is then searched in a way that shrinks this range of likely values as much as possible. In choosing which move to make, McAllester follows the dominance strategy of B*, terminating the search when the lower bound of one possible move exceeds the upper bounds of all other moves. Note that this approach depends on making an *a priori* assumption about what size of conspiracy is "unlikely". More work is needed to analyze the basis for such a choice.

2.0.4 Theoretical computer science

Since the recognition of the intractability of finding exact solutions for many interesting classes of problems, theorists have spent a good deal of time studying algorithms with one of two properties:

1. A guarantee that the solution returned will be within some ϵ (either relative or absolute) of the optimal solution. This is called *approximation*.

2. A probability of at least $1 - \delta$ that the algorithm will return the correct or optimal solution. These are often called *probably correct* algorithms.

Some researchers, notably Valiant and others in the field of inductive learning [97,41] have studied 'probably approximately correct' algorithms, combining the above properties in the obvious way. In many cases complexity bounds are expressible in terms of the error parameters δ and ϵ; this provides us with exactly the information needed to carry out effective resource allocation among a discrete set of competing processes (see [22]).

However, as Horvitz [46] has pointed out, what is needed is a theory of algorithms that maximize the *comprehensive value* of computation. In other words, since the utility of a computation and resulting action is a function $U(S,t)$ of both the quality S of the resulting solution and the time t taken to choose it, we would like methods for designing algorithms that maximize this combined utility. Current theory studies only minimization of t for fixed output behaviour, or in some cases minimization of error for fixed t. We believe that a combined approach may be possible using the techniques described below, and is essential for a system faced with varying resource constraints.

3 Optimal allocation of computational resources

In order for the concept of resource allocation to make sense, the system in question must have a choice of computations available to it, each of which can return a decision or can affect the ultimate decision made. The computations may vary along dimensions such as the amount of time used, the quality of the solution returned, the certainty that an adequate solution will be returned, and the usefulness of a partial computation if the process is interrupted. For our purposes, the amount of time used and the quality of solution returned will be of paramount importance, since limited resources of time are the major focus of this research.

Choices concerning which computation to carry out 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, and by carrying out an unbounded computation at design time. The latter method has been preferred in computer science generally, being the subject of research in algorithms, but for the tasks faced in AI the variety of situations and time pressures is too great, and we must imbue our systems with an autonomous capability to control computation in a situation-dependent way.

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 reasonably well-established in decision science [37], is relatively new in AI. In addition, getting the model right and putting it into practice are not such easy tasks. Work in decision science has concentrated on the relatively simple case of deciding whether to act now or to carry out a single computation and then act on the results. We discuss this situation, and other, more realistic cases, in the following paragraphs. We first give a formal definition of the expected value of a computational action in the general case, and discuss why it is a difficult quantity to estimate. We then discuss various methods for computing approximations to this quantity. Finally, we describe an implementation of one of these methods, and discuss its strengths and weaknesses.

Let us first review the basic aspects of any implementation of decision theory [88,99]. We have

1. A utility function $U(W)$[1] on world situations W, whose range must form a partial ordering and be closed under multiplication by the reals. (Standard real-valued utility functions satisfy these requirements.)

2. A set of possible actions A_i

3. Some knowledge of the current world situation, expressed by X's subjective probabilities $P(W_j)$ of the possible situations W_j.

4. Some knowledge of the outcome O_{ij} of action A_i in situation W_j.

The optimal action is that which maximizes the agent's expected utility, computed according to the formula[2]

$$E[U([A_i])] = \sum_j P(W_j)U(O_{ij}). \tag{1}$$

In many situations, such as betting on dice, the application of decision theory is quite straightforward. After all, probability was invented partly out of a desire to understand gambling. A good introduction to decision theory and its uses in artificial intelligence is given in [47].

In the choice of computations, however, we have a different story. A computation directly affects the system's internal state, and only indirectly the external world (except by consuming time), whereas a utility function usually refers only to aspects of the total situation that are external to the agent (such as winning a game of chess) or to aspects of internal state such as happiness that are, fortunately, not generally accessible to direct computational manipulation. So how are we to judge the effects of computations, in order to choose properly between them?

The answer lies in *knowledge of the system's own decision-making process.* It is essential to know how an *internal* computation will affect the system's ultimate choice of an *external* action. Thus, there is value in computation only inasmuch as it affects the system's ultimate choice of action.

There are also costs to computation. As stated in the introduction, real-time AI is properly defined as the study of decision-making under conditions in which the utility of a given action varies significantly with the time at which it is carried out. Any class of time constraints can be represented by describing this variation. The 'cost of computation' arises from the fact that in many cases the utility of an action decreases with time (or the agent's overall utility is decreasing in the absence of action), so that the time delay caused by computation will tend to reduce an agent's expected utility, all other things

[1]In the standard formulation the utility function is indexed by the agent and by the time at which the agent's preferences are calculated. The calculation is assumed to be instantaneous. We omit the agent index for simplicity, and assume that the agent's preferences *on world states* remain constant, although the world state can certainly include a clock.

[2]We will refer to the result of taking action A in the 'current state' as $[A]$. This notation, though informal, is adequate for our current purposes. When we need to be more specific about the state in which the action is taken, we will use the notation $[A, S]$.

being equal.[3] Later, we define the conditions under which the cost of a computation can be defined and separated from its 'intrinsic benefit'.

Note that we do not assume that the agent knows the expected utilities of his actions; otherwise, no computation would be necessary, and the agent could simply take the action with the greatest expected utility. In general, it may be impossible for the agent to determine with certainty all of the probabilities and utilities he would have to know in order to compute the expected utility by applying equation 1. However, often the agent does have available an *estimate* of the expected utility $U([A_i])$ of his actions. In the AI literature, such estimation functions are often called evaluation functions. We will use the symbol $\hat{U}$ to denote the agent's current estimated utility function. In fact, it is often the case that *computation proceeds by revising the estimated utility function*. This is the general picture of deliberation about how to act on which we will most closely focus our attention. For instance, in a game-playing program, a computation might involve a further ply of search, followed by back-up of new minimax values for the top-level move choices. These new backed-up values would give the agent's new estimated-utility function, based on its latest computation. Note, however, that we make no general assumptions about how the estimated utility function is arrived at. For instance, the estimate might or might not be based on individual estimates of the component probabilities and outcome utilities.[4]

We assume that at any given time the agent has available to it a default action, which is the action, α, that appears best to it given its reasoning to date. That is, $\alpha = \underset{A}{\text{argmax}}\{\hat{U}([A])\}$. Initially, before any deliberation has been done, $\hat{U}$ will presumably be a constant. For definiteness, we will assume that ties are broken somehow, perhaps by picking randomly from the actions with maximum estimated utility, so α is always a determinate action.

In general, the agent has a choice between acting upon its reasoning to date, that is, carrying out its *current best action* α, and engaging in further deliberation. Deliberation consists of *computation steps*, which represent the basic units about which control reasoning is done. The utility of a computation step S_j is defined, just as in ordinary decision theory, in terms of the resulting state $[S_j]$.

Before sketching how this utility can be estimated by a program, it is worth a few words to discuss its use. It is tempting to view a formula for $U([S])$ as a general intermediate utility expression for non-committed states, such that the utility increment of an individual computation step S_j can be viewed as the difference between the formula applied to $[S_j]$ and the same expression applied to the current state. This would be a mistake. The agent does not have a choice between carrying out the computation and *staying in the current state*: time passes inexorably, and given that the expected intrinsic utility of

[3]An additional source of computation cost may be that computing itself consumes energy, as is the case in biological systems. For the purposes of AI, however, we will assume that all instruction executions, including no-ops, have the same cost per unit time.

[4]It is not necessary, of course, that the agent ever actually make a numerical estimate of the utilities of the available actions and then choose the action with maximum estimate. It might instead simply have an *estimated preference ordering*, and choose the action which comes first in the ordering. We will have little directly to say about this more general case.

computation is always non-negative, the agent is obliged either to act or to compute. In other words, any completion of the computation beginning with the null step (no-op) has lower expected utility than the best completion beginning with a useful step. Thus to choose a computation step S_j, we compare the utility of $[S_j]$ with the utility of $[S_k]$ for $k \neq j$, and also to the utility of the state in which the agent has committed to the current best action α.

We therefore define the *net value* $V(S)$ of a computation S as the *increase* in utility gained by carrying out that computation rather than ending deliberation and taking the current best action immediately.

$$V(S) = U([S]) - U([\alpha]) \tag{2}$$

Similarly, we define the *estimated net value* $\hat{V}(S)$ as the increase in estimated utility (= the estimated increase in utility) due to the computation.

$$\hat{V}(S) = \hat{U}([S]) - \hat{U}([\alpha]) \tag{3}$$

The major distinction that must be made in calculating $U([S])$ is between *isolated computations*, those in which no further computations occur after S, and *partial computations*, those that may be followed by additional deliberation.

Case 1: Isolated computations.
Suppose we know that the agent will have to act after the computation step S_j in question (if it does not act immediately). In other words, suppose the agent is only choosing between complete, or *isolated* computations. Then the utility of S_j will be equal to the utility of the action α' believed by the agent to be optimal after S_j has been carried out, given that the action is carried out *after* S_j is completed. That is,

$$E(U([S_j])) = E(U([\alpha', T_j])) \tag{4}$$

where T_j is the time at which computation S_j will be completed if begun at time T. (Note that from the agent's point of view, both α' and $U([\alpha', T_j])$, and perhaps T_j as well, are random variables in the current state. That is, their actual values are not yet determinate, though the *expectations* will be well-defined.)

In this case, therefore, the expected net value of S_j is given by

$$E[V(S_j)] = E[U([\alpha', T_j]) - U([\alpha])] \tag{5}$$

That is, the net value is what the agent gained (or lost) by doing the calculation and thus delaying action.

In [48], R.A. Howard, using a somewhat similar formula to the above, analyzed in detail the expected value of obtaining complete information about various unknown quantities in a competitive bidding situation. For our purposes, a more directly usable formula is the following, which gives the expected *estimated* net value of computation S_j

$$E[\hat{V}(T, S_j)] = E[\hat{U}([\alpha', T_j]) - \hat{U}([\alpha])] \tag{6}$$

An important point about this formula is that in many domains, it is possible to arrive fairly cheaply at a useful estimate of the expected estimated net value of a particular computation *before* it is carried out, given statistical data concerning the value derived from *past* computations. We will describe below a game-playing program we have implemented which makes use of such estimates. Briefly, suppose we can characterize S_j as belonging to a given *class* of computations, i.e. suppose S_j is a particular *type* of computation, such as an additional ply of search in an iterative-deepening algorithm[5], being considered in a particular type of situation, where the situation is characterized by some pre-determined set of features. Then we can characterize the distribution of the random variable $\hat{V}(S_j)$ to almost any desired standard of approximation by computing *post hoc* the net increase $\hat{U}([\alpha', T_j]) - \hat{U}([\alpha])$ for a large sample of computations in similar situations drawn from the same class.[6] Of course, computing the actual increase for a large sample of similar situations will be much more expensive than simply carrying out the computation S_j. However, if a large amount of "leisure time" is available, in which such statistical sampling can be done without time pressure[7], with the resulting distributions stored for later use, presumably in a parametrized form, then the cost of applying formula 6 to estimate the expected net value of computation S_j can be orders of magnitude cheaper than carrying out S_j itself. In that case, the expected value calculation will be well worth doing, since in many cases it can tell us ahead of time that S_j is expected to be of little value, and hence should not be carried out. Or, if more than one computation is under consideration, we can choose the one with highest expected net value.

Case 2: Partial computations.
If we could always assume that the agent would necessarily take the action α' after performing search action S_j, then calculating the value of computation would always be as simple as in the previous paragraph. However, in general this is not the case, since, as long as the agent does not arrive at complete certainty about its utility function—which we assume our agents almost never attain[8]— in state $[S_j]$ the agent will still have a choice between taking action α', and continuing to deliberate. Thus the value of the computation S_j will be the value of having this choice. There are at least two ways to model the utility of being in the state of having a choice between actions.

A. If we assume that the agent will be able, when faced with such a choice, to choose

[5]In fact, in our actual experiments, we computed statistics for individual node expansions in the game of Othello. This case is slightly more complicated to describe; we discuss it in detail below.

[6]In fact, we can simplify the *post hoc* computation of the net increase in the sample situations by decomposing the estimated utility of each move into an *intrinsic* component, independent of time, and a time-cost component, which depends only the amount of time required to perform a given computation. We discuss this decomposition in more detail below.

[7]For instance, this is clearly true of game-playing, where, at least in terms of time, quite high development costs can easily be afforded, even though the time pressure during actual play is high.

[8]There are some exceptions to this; for instance, during end-game play it is sometimes possible to determine the exact minimax value of a position. In such cases further computation would obviously be a waste of time.

the best option, then the utility of having a choice between two actions is just the maximum of the utilities of the two actions. This is a standard approach taken in decision analysis, for instance by Howard in [48]. Thus let S_k be the action of continuing to compute in state $[S_j]$, and let T_j be the time at which S_j is completed. Then on this model, the utility of computational action S_j in the current state is given by

$$E[U([S_j])] = E[\max\{U([\alpha', T_j]), U([S_k, T_j])\}] \tag{7}$$

One difficulty with this formula is that it appears to give little help in figuring out how in general to *calculate*, or even estimate, the expected value of a computation, since it defines the value of a present computation in terms of the value of a future computation. In the next section, we will describe a possible line of research which we hope will overcome this difficulty.

Moreover, it is probably unreasonable for the agent to assume that it will necessarily choose the action with maximum utility in state $[S_j]$. For even though he is *committed* to choosing the action which then *appears* best to him, it doesn't follow that the best action will in fact appear best.

On the other hand, if we replace U by $\hat{U}$ in equation 7, we obtain the following:

$$E[\hat{U}([S_j])] = E[\max\{\hat{U}([\alpha', T_j]), \hat{U}([S_k, T_j])\}] \tag{8}$$

It might be argued that equation 8 is indeed a principle of rationality, in the sense that it imposes a constraint on any *extension* of $\hat{U}$ from base-level actions to computational actions. For it is true almost by definition that the agent will pick the action with maximum $\hat{U}$, and hence in a sense equation 8 says that the agent should value an action the same as he values the computation that told him to take the action. But against this it might be argued that if the agent *knows* that his estimated-utility function is error-prone, he needn't think that the right-hand side of equation 8 gives the true expected utility of the computational action S_j.

We have found it very difficult to settle the sorts of philosophical questions raised by such considerations, and will not attempt to do so here. But we hope the dilemma of the previous paragraph might convince the reader, as it has us, that the questions raised here are very deep.

In any case, equation 8 suffers from the same difficulty as equation 7, that it cannot define a *unique* extension of $\hat{U}$ to computational actions because it is "ungrounded"; i.e. it does not specify how the recursion bottoms out. Later in this section we will discuss a possible way of overcoming the non-uniqueness, by specifying a *conservative extension* of $\hat{U}$ from base-level to computational actions.

B. More generally, if we do not wish to assume that the agent will be able to choose infallibly the best action in state $[S_j]$, we may assume instead that the agent has a certain probability of taking any given action. This probability may be directly related to the utility of the action; in an extreme case, if we assume that the probability is 1 for the action with highest utility, and 0 for other actions, we arrive at the model of equation 7. The better

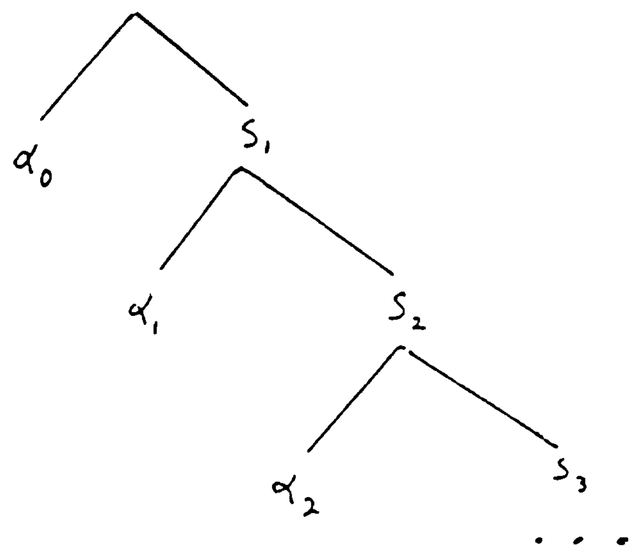

Figure 1: A decision tree for possible computations and resulting actions

the agent's utility estimator $\hat{U}$ as an approximation to U, the closer will these probabilities come to this extreme case, but as long as $\hat{U}$ remains error-prone, the probabilities will lie in the open interval $(0, 1)$.

Let $p_j(\alpha'), p_j(S_k)$ be respectively the probabilities that in state $[S_j]$ the agent will immediately take the new best action α', and that it will continue deliberation with computational action S_k. Then on this model the expected utility of the computational action S_j is given by

$$E[U([S_j])] = E[p_j(\alpha')U([\alpha', S_j]) + p_j(S_k)U([S_k, T_j])] \tag{9}$$

Again, this formula gives the value of a current computational action in terms of the value of a future one. However, if we expand the action S_k recursively in a similar way to the expansion of S_j, and so on, we develop the familiar *decision tree* characteristic of such situations, shown schematically in figure 3.

Moreover, if we expand all computational actions in this way, the right-hand side of equation 9 becomes the expectation of a sum of terms of the form $p_j q_j(\alpha_0) + p_k q_k(\alpha_1) + \ldots$, where the α's are selected from the base-level actions A_i, and q_k represents the probability of reaching the kth node, that is, $\prod_{i=j}^{k-1}(1 - p_i)$. This expressions thus averages over all possible complete computations (see equation 10).

If we then combine terms corresponding to situations in which the same action is chosen, we get an expression of the form $p_1(A_1) + \cdots + p_n(A_n)$, where now p_i represents the *probability that action A_i is eventually chosen* after completing the computation. Thus, after transforming the equation in this way, we can then drop the expectation sign on the right-hand side (see equation 14 below).

We will explore these ideas, and discuss possible implementations based on them, in more detail in the next section.

Whichever model we choose for the utility of a possible computation, the idealized algorithm that results in optimal overall behaviour is the following:

1. Keep performing the most valuable of the available computations until none have positive value V.

2. Commit to the action recommended by the last computation in 1.

This intuitive definition of the value of a computation, and the algorithm accompanying it, were proposed by Good [37,38]. This algorithm is to be distinguished from a *best-first search*, in which search steps are ordered by the utilities of the nodes to which they apply. Although 'best-first' sounds like a good idea, there is no compelling reason why there should be a correspondence between node utility and node search value. We do show below that in some cases this correspondence exists, but in others it does not.

Obviously, the computation of the values V cannot be instantaneous; in fact, as we describe below, it can be arbitrarily hard. However, in many cases calculation is sufficiently fast to result in benefits over other control methods. In other cases, the formulation of a formally correct estimation of the value of computation results in a tractable meta-level after compilation, or forms the basis for well-motivated approximations.

4 Calculating the value of computation

In order to develop the appropriate formulae for calculating the value of a computation, we must first introduce some notation. As we have seen, the notion of the agent's *estimate* of its utility is central to a discussion of the agent's deliberative actions, since deliberation in general proceeds by revising the estimated utilities. Let us use $\mathbf{S}$ to denote the base-level computation sequence carried out up to the current state. In a game-playing system, for example, $\mathbf{S}$ might be the sequence of node expansions to date. If we are now considering some computation step S_j, then the computation sequence after S_j has been carried out will be $\mathbf{S}.S_j$. We will use the notation $\hat{Q}^{\mathbf{S}}$ to represent an estimate of a quantity Q calculated in the state resulting from computation $\mathbf{S}$.[9]

In this section, we will try to accomplish three things. We want to spell out in detail the transformation of equation 9 alluded to at the end of the last section. Also, we want to replace the function U in that equation by the function $\hat{U}$, since we are assuming that the former is unknowable, and the agent is able to calculate only in terms of the latter. This will require some care, and perhaps further transformations of the equation. Finally, we will try to show how, and under what conditions, the net value of a computation can be decomposed into two terms, the net or *intrinsic benefit* of the computation, due to the improved choice of action, and the *cost* of the computation due to the expenditure of temporal resources. This is an important issue because AI applications usually employ evaluation functions for actions that are independent of time.

As mentioned above, the probability distribution among actions corresponds to, and can be derived from, a probability distribution among *complete computations*, since a completed computation defines the action the system will take, assuming its decision-making is

[9]In discussing an estimate of some quantity of interest, such as the agent's estimate of its actions' utility, it is important to keep in mind the computational state on which the estimate is predicated, since we will be considering estimated utilities and probabilities at different stages in the decision-making process; in particular, the current state and the projected state after the computation step we are considering.

deterministic. By a complete computation we mean any *sequence* of computational actions that ends with the agent performing a particular base-level action.

We will return to the problem of the system's estimated utility if it acts immediately. Now, let us consider the situation resulting from taking computational action S_k. In this state, we can complete the computation by taking any sequence $\mathbf{T}$ of computational steps, followed by an actual action in the world. $\mathbf{T}$ could be the null action, in which case the system will take the current best action after S_k. Since we are not now *choosing* the subsequent course of computation, it is reasonable to impose a probability distribution over the sequences $\mathbf{T}$. Then the utility of $[S_k]$ can be estimated from averaging the utilities of the states resulting from all possible complete computations beginning with $\mathbf{S}.S_k$:

$$\hat{U}^{\mathbf{S}.S_k}([S_k]) = \sum_{\mathbf{T}} \hat{Pr}^{\mathbf{S}.S_k}(\mathbf{T}) \max_i \hat{U}^{\mathbf{S}.S_k.\mathbf{T}}([A_i, t(\mathbf{S}.S_k.\mathbf{T})]) \tag{10}$$

Note that the estimated utility of an action A_i is calculated in the computational state in which it is taken. For example, in a chess program, the estimated utility of a move will be the evaluation backed up from the search tree that has been grown to date. But also, the estimated utility is a function of the external situation in which the move is taken, this situation being denoted by the function t.[10] In simple cases such as chess, the function t denotes the time that has passed during a computation, since the board situation does not change. In more complex (and realistic) worlds, the dependence extends to include anything that changes as the computation proceeds and affects the nature and utility of the action's outcome.

The utility estimator that takes as arguments both the 'time factor' and the 'local' state resulting from the action is a well-defined function: the optimal resource-bounded probability estimate for winning the game from the class of states described by the arguments. But typically we are given some estimator that is independent of time and takes just the action outcome as argument. We will call such a function the *intrinsic utility*, or *intrinsic benefit*. We can then define a function TC, for time cost, that simply expresses the difference between total and intrinsic utility:

$$\hat{U}([A_i, t]) = \hat{U}_I([A_i]) - TC([A_i, t]) \tag{11}$$

Rewriting equation 10, we obtain

$$\hat{U}_I^{\mathbf{S}.S_k}([S_k]) = \sum_{\mathbf{T}} \hat{Pr}^{\mathbf{S}.S_k}(\mathbf{T}) \max_i [\hat{U}_I^{\mathbf{S}.S_k.\mathbf{T}}([A_i]) - TC(A_i, t(\mathbf{S}.S_k.\mathbf{T}))] \tag{12}$$

Now if we assume that the time cost term TC is independent of the state being evaluated (that is, of the action being taken), then it will just be a function of t and can be factored out of the max expression:

$$\hat{U}^{\mathbf{S}.S_k}([S_k]) = \sum_{\mathbf{T}} \hat{Pr}^{\mathbf{S}.S_k}(\mathbf{T}) \max_i [\hat{U}_I^{\mathbf{S}.S_k.\mathbf{T}}([A_i]) - \sum_{\mathbf{T}} \hat{Pr}^{\mathbf{S}.S_k}(\mathbf{T}) TC(t(\mathbf{S}.S_k.\mathbf{T}))] \tag{13}$$

[10]It is of course possible to make t be implicit in the value of $[A_i]$, but it seems more perspicuous to separate the two.

Hence, under certain assumptions, the notion of a general time cost for computation can be made precise and used to simplify the meta-level decision process.

Obviously, for the system to make an effective choice between computations, it cannot possibly examine all possible completions $\mathbf{T}$ in order to implement equation 13. In the following paragraphs we examine some possible methods for arriving at reasonable estimates of the value of a computation; each uses a different way of averaging over the possible computations, and each therefore requires different statistical knowledge of the domain under consideration. By making still further approximations, we can arrive at some interesting domain-independent heuristics.

The probabilistic strategy approach

The above formulation (equation 10) takes the subsequent course of computation $\mathbf{T}$ as the principal random variable in estimating the utility of the situation resulting from computation step S_j. The action ultimately chosen is treated as a dependent variable. An alternative approach is to focus on the actions as random variables, estimating the probability that each will become the action actually carried out. In this approach, the situation $[S_j]$ after the computation step is evaluated as if the agent had a *probabilistic strategy*, somewhat like that proposed by von Neumann [99] as a solution to certain games. Thus in $[S_j]$ the agent's expected utility is estimated by calculating the probability that each action A_i will be chosen eventually, and multiplying by the estimated utility of each action *on the assumption that it is chosen.*

$$\hat{U}^{\mathbf{S}.S_j}([S_j]) = \sum_i \hat{\Pr}^{\mathbf{S}}(A_i)\hat{U}^{\mathbf{S}}([A_i]|A_i \; chosen) \tag{14}$$

where we use the conditional notation to denote informally that the fact of an action being chosen influences our estimate of its utility, and the fact that the action's utlity will depend on the time at which it is taken, and hence on the computation that ends in its being recommended.

Now, in order to estimate the utility of $[S_j]$ using the above equation, we need to evaluate three quantities:

1. The probability distribution for the values of the various actions after the step S_j. This is the same problem as appears in the isolated computation case.

2. The probability that a given action will be chosen given the action values in $[S_j]$ and the expected variation in the action values with further search. In principle, this involves a probability distribution over the possible completions, as in equation 10. In practice, we can get approximate values from prior statistical experiments.

3. For each action, the value it is expected to have *given that it is chosen.* This requires the same information as the previous item.

We cannot go into detail here on the computation of these items. The advantage this approach possesses over the single-step-assumption approach described below is that it allows the agent to judge the value of a computation step that by itself *cannot* alter the action to be taken, but because of its altering the action value estimates can set up subsequent revisions of the current best action. We can describe a tree that is very unlikely to yield a subsequent alteration in the choice of move as having high *inertia* or *stability*; a computation step that renders a tree *unstable* can thus be valuable.

Adaptive feedback estimation

Another approach to estimating the value of partial computations is based on equation 8; it involves the idea of a *conservative extension* of the agent's utility-estimation function $\hat{U}$ from base-level actions to computational actions. As we noted above, an apparent difficulty in attempting to use equation 8 is its "ungroundedness"—it defines the value of a present computation in terms of the value of a possible future computation. On the other hand, an advantage of this equation is that it does not require an estimate of the agent's probabilities of taking the various actions; all that is required is knowledge of the distributions of the values of the actions, *including computational actions.* We have already mentioned that the distribution of the value of α' can be statistically inferred from sample outcomes of the various computational actions. We believe that the same technique might well work in learning the distribution of the value of S_k as well. Two things are required to implement this approach. First, in order to evaluate the outcomes of our sample computations, we would need to have a method for arriving at a reasonable *post hoc* evaluation of a given computational action, S_k. Second, we would need features of situations, including the agent's internal computational state, that discriminate between situations on the basis of the distribution of the value of further computation.

Regarding the first requirement, it is to be noted that at the time we are collecting our sample data, we can have available to us the complete outcome of any given decision-making incident. This makes the problem of evaluating a computational action *post hoc* quite different in nature from the problem of estimating the expected value of a future computational action. If we are willing to assume, for the purpose of collecting our sample data, that our current agent's decisions are, to a certain extent, correct, then we can arrive at justifiable values for sample computational actions. Consider for instance the decision history shown in figure 4. In this case, the agent's first computational action S_0 at time t_0 resulted in a state where its best base level action was α_0 and its best further computational action was S_1. It chose to continue deliberating with S_1, which brought it to a state with best base level action α_1 and best computational action S_2. It again chose to continue deliberating with S_2, which resulted in best action α_2. At this point, it chose to stop deliberating and take action α_2, at time t_3.

Consider now the value of computational action S_1. On the isolated-computation model, i.e., ignoring the effect of further computation, our *post hoc* estimate of the utility of this action would be[11],following equation 4,

$$\hat{U}([S_1, t_1]) = \hat{U}([\alpha_1, t_2]) \tag{15}$$

On the current model, however, we would instead employ equation 8, and determine

[11]Note that here the estimate $\hat{U}$ is being made after all computation has ended. Hence we dispense with the superscript notation, and employ an explicit temporal argument.

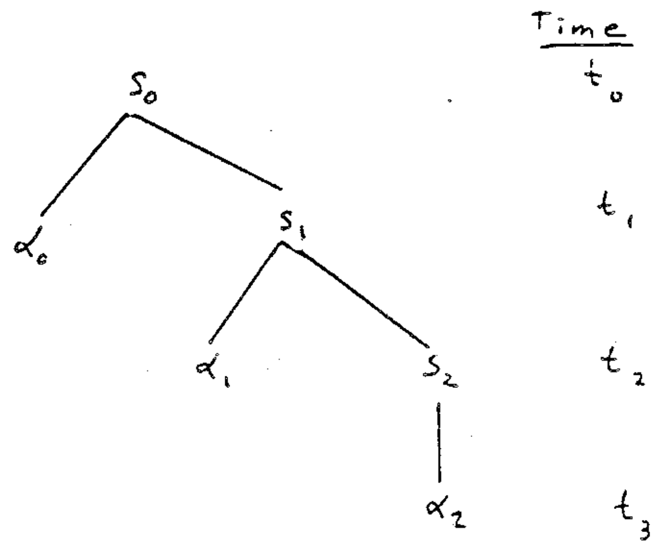

Figure 2: Evaluating a computational action *post hoc*

the value of S_1 by "backing up" values from the bottom of the tree. That is, if we assume that the agent was correct in stopping deliberation when it did, it must follow that α_2 was better than any possible further computation it could have made at that point. Hence it follows that the estimated utility of S_2 was simply

$$\hat{U}([S_2, t_2]) = \hat{U}([\alpha_2, t_3]) \tag{16}$$

Applying equation 8, the estimated utility of S_1 is given by

$$\begin{aligned}\hat{U}([S_1, t_1]) &= \max\{\hat{U}([\alpha_1, t_2]), \hat{U}([S_2, t_2])\} \\ &= \max\{\hat{U}([\alpha_1, t_2]), \hat{U}([\alpha_2, t_3])\}\end{aligned} \tag{17}$$

Similarly, continuing in this way we find that

$$\hat{U}([S_0, t_0]) = \max\{\hat{U}([\alpha_0, t_1]), \hat{U}([\alpha_1, t_2]), \hat{U}([\alpha_2, t_3])\} \tag{18}$$

and so on.

The last equation may look more interesting if we assume that $\hat{U}$ is given by an intrinsic utility estimate $\hat{U}_I$ and a time-cost estimator TC. In that case we find that

$$\hat{U}([S_2, t_2]) = \hat{U}_I(\alpha_2) - TC(S_2) \tag{19}$$

and therefore that

$$\hat{U}([S_1, t_1]) = \max\{\hat{U}_I(\alpha_1), \hat{U}_I(\alpha_2) - TC(S_2)\} - TC(S_1) \tag{20}$$

and so on.

Finally, we can define the *estimated net value* of S_1, along the lines of equation 2, by

$$\hat{V}(S_1) = \max\{\hat{U}_I(\alpha_1), \hat{U}_I(\alpha_2) - TC(S_2)\} - TC(S_1) - \hat{U}_I(\alpha_0) \tag{21}$$

This value of $\hat{V}$ then becomes a data point for that type of search action in that particular type of situation. Given a large number of such data points, it is possible to know a great deal about the distribution of V in various types of situations. Such knowledge can then be used to make decisions about when to search in future situstions.

The reason the extension of $\hat{U}$ to computational actions derived in this way is called "conservative", is that it preserves the agent's judgments about the *relative* value of computational and base-level actions. That is, one might well ask why we should in hindsight accept the agent's decision to stop deliberating at time t_3. After all, we could if we wanted simply continue the computation where the agent left off; we might ultimately come up with an evaluation of the search actions S_1 and S_2 drastically different from the above. Such a radical re-evaluation of the agent's deliberative process *might* turn out to be worthwhile; but we suspect it would ultimately have to result in a re-working of the agent's evaluation of *base level* actions as well. Part of the reason for preferring a conservative extension of $\hat{U}$ to computational actions over a more radical one is that we wish to observe the intuitive distinction between deliberation about the base level and about the meta-level. The point of the current research program is to figure out how, *given* a limited rational agent with its own methods for estimating the utility of its actions and its own procedures for revising those estimates by deliberation, such an agent's decision-making procedures can be *optimized* by rationalizing its choice of which computational action to take at a given time.[12]

Data derived in the above way will of course be error-prone, since the whole point of doing the analysis is that we think that the agent often makes incorrect judgments concerning when to stop and when to continue computing. Certain of these errors will be caught by the above method of evaluating sample computations, if the agent makes computations which turn out *post hoc* to have had negative value. For instance, in the above example, if α_2 turns out to be the same move as α_1, we would estimate the net value of computation S_2 as simply $-TC(S_2)$. However, in cases where the agent cuts off search prematurely, our *post hoc* evaluation will underestimate the value of the actual search actions taken, since we are tied to the agent's own underestimate of the value of search actions *not* taken. In this case, our estimates will be lower bounds on the true value of the search actions taken.

Note, however, that this sort of underestimate may be precisely what we want. Recall that one objection made against equation 7 was that it assumes the agent *will* choose the most valuable action. Since this is not always the case, the distribution we want to measure is that of the value of a given computational action *to the agent*. There is no point in arguing that the action had a higher value than we are assigning to it, if the agent is incapable of recognizing this and acting on it.

On the other hand, once we have done our sampling and equipped the agent with decision-theoretic search control knowledge, it will in a sense no longer be the same agent, since it will make different choices in the same sorts of situations. We expect that its

[12]In fact, this may seem a rather superficial reason for preferring a "conservative" extension of the utility estimator from base level to computational actions. The deeper reason is that we strongly suspect on intuitive grounds that such an extension will be more *reliable* than a more radical one, though we haven't proved this or even fully spelled out the notion of reliability that we have in mind.

behavior will be improved, i.e. that having knowledge about the expected value of its computational actions should enable it to make better decisions. The problem is that as soon as that knowledge is incorporated into the agent's decision-making procedures, it will be out of date, since it is knowledge about the value of computational actions performed by the agent *as previously constituted.* The solution to this is simply to repeat the process of gathering data about the agent's computations, and then incorporate the new knowledge into the agent's procedures, and so on. In fact, once an initial set of distributions is obtained, it should be a simple matter for the agent to revise those distributions incrementally as it gains more decision-making experience. In this way, the agent might adaptively converge on a state in which the agent would possess accurate knowledge of the value of its own computational procedures.

A number of technical problems need to be solved before the ideas of this section can be implemented. The most important one is that we will need to find easily-computable features of the search trees which will discriminate situations according to the distribution of the value of further search. We believe that in game-playing domains, one important such feature will almost certainly be the *conspiracy number of the search tree*, where this is defined here, by a slight alteration of McAllester's definition, as the minimal number of leaf nodes whose values must change in order to change the current best move choice. It is clear that as this number increases, the cost of achieving a given increase in the utility of the chosen move increases as well. In this way, we hope to make McAllester's valuable insights a part of a rigorous decision-theoretic approach to game-tree search control. In fact, we believe that similar properties are relevant to single-agent search trees, as we discuss in section 5.2 below.

5 Specific base-level problem-solvers

Up to this point, we have been working at a very general level, making absolutely 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, ranging from a single machine cycle to a multi-week economic simulation; and we can apply the above analysis to a single step in isolation, as for example when deciding whether to pay for the aforementioned economic simulation or to use the proverbial seat of the trousers. In the case of the isolated computation step, the analysis becomes much simpler, as we discuss below.

We now look at some particular systems in more detail. These systems are amenable to analysis because of the close match between the base-level decision-making mechanism and the standard decision-theoretic model. This means that such things as utility estimates are already available as an integral part of the computation. The analyses, although simplified, will serve as models for explorations into more complex, real-world systems.

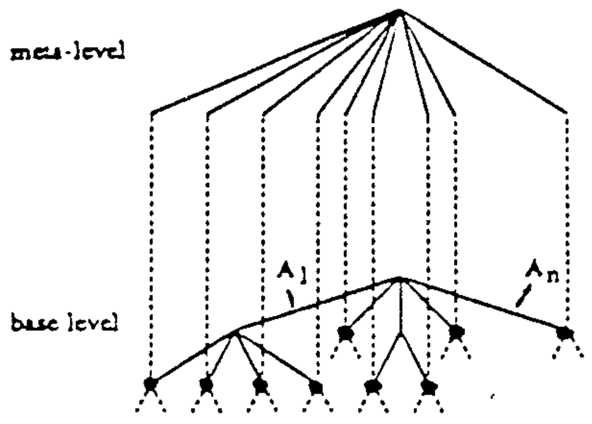

Figure 3: Schematic two-level search

5.1 Simple decision-making in real time

In this section we examine the case of a base-level system that is a simple, decision-theoretic algorithm with lookahead. It essentially follows the algorithm sketched on page 17. In order to make calculation of the expected value of computation tractable, we will make certain simplifying assumptions, which we will introduce explicitly and discuss in detail as we go along.

We will continue our practice of using A_i to denote a generic base-level action available to the agent. These actions are called "top-level moves", to distinguish them from possible future actions represented lower down in a search tree. As before, we use α to denote the current preferred move, i.e. the action with highest current estimated utility. We use β_1, β_2, etc., for the current second-best move, current third-best, and so on.

5.1.1 Meta-greedy algorithms

As mentioned above, explicit consideration of complete sequences of search steps is clearly intractable. The obvious simplification is to consider *single* search steps and to estimate their ultimate effect: we then choose the step appearing to have the highest immediate benefit. We call such algorithms *meta-greedy algorithms*. We get different variants depending on how we estimate the ultimate effect of the search step. A meta-greedy algorithm, then, is one that effectively has a fixed meta-meta-level policy of a depth-limit of 1 on the meta-level decision problem. The decision-making situation is summarized in figure 5.1.1. At the meta-level we have a choice of search actions, each corresponding to the expansion, possibly by more than one level, of a node at the base level.

5.1.2 Single-step assumption

The above expressions for the estimated expected utility of a computation (equations 10 and 14) 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 had time for at most one more search step.

Assumption 1 (Single-step):

> The value of a search step can be estimated by evaluating its expected effect as if it was the only remaining search action to be taken; that is, as if it was an isolated computation.

Note that this assumption is true if we are considering search *sequences* instead of steps, but as it stands it is strictly false. It sometimes predicts that single search steps have no value, whereas often those steps enable other steps to become valuable. The simplification comes from the fact that, given a minimax base-level, it is relatively easy to predict how a search step will affect the choice of move, since the system will always choose the move with the highest backed-up value. In other words, given the single-step assumption, the probabilities $\Pr^{\mathbf{S}.S_k}(A_i)$ in equation 14 will be 1 for the highest-valued move and 0 for the other moves.

Our implementations to date have employed Assumption 1 as a simplifying assumption. We are currently working on implementations which will employ the strategies outlined in Section 4, which we hope will obviate the need for such an assumption. In this section we describe the problems which had to be solved in order to implement actual algorithms employing the single-step assumption, and discuss their performance, strengths and weaknesses.

Let α be the move we would have made *before* performing the search action S_j being considered; that is, the current preferred move. Its current expected utility is $\hat{U}^{\mathbf{S}}(\alpha)$. Let α' be the move that is preferred *after* performing S_j.

Hence the *intrinsic benefit* of a search action S_j is given by the following:

$$\Delta(S_j) = \hat{U}_I^{\mathbf{S}.S_j}(\alpha') - \hat{U}_I^{\mathbf{S}.S_j}(\alpha) \tag{22}$$

Recall that, prior to performing S_j, $\Delta(S_j)$ is a random variable since we don't yet know how the search step will affect our opinions of the moves.

Writing $TC(S_j)$ for the time cost of the search action, the *expected net value* $E[\hat{V}(S_j)]$ is defined using the expectation of the benefit of S_j:

$$E[\hat{V}(S_j)] = E[\Delta(S_j) - TC(S_j)] \tag{23}$$

An important corollary of equations 22 and 23 is that if $\alpha = \alpha'$ then $V(S_j)$ is guaranteed to be negative. Since the *true* move utility is unchanged, *a search action that doesn't change the system's preferred move will have had no value.* It is important to recall that this result is conditioned on the single-step assumption. We will see that, while it makes the expected-value computation tractable, this assumption also places certain limitations on the depth of search in some domains, including game-playing.

5.1.3 Variation of estimated utility with search

Looking at equation 10, we see that computing the expectation of $\Delta(S_j)$ requires some knowledge as to how the estimated utility of the top-level moves A_i will change as a result

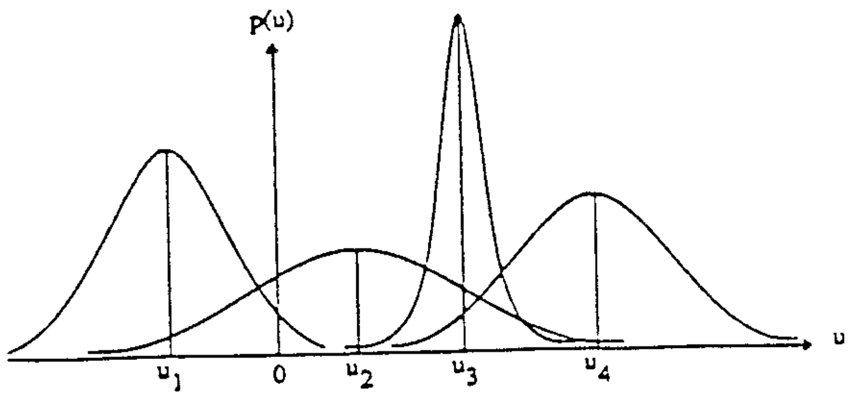

Figure 4: Move utilities with error distributions

of S_j. We will use the term *error* to describe the variation of $\hat{U}^{\mathbf{S}.S_j}(A_i)$ from $\hat{U}^{\mathbf{S}}(A_i)$. This notion of error has a clear, operational semantics and is empirically available (the statistical prediction of errors in game-playing is described in section 4). It also provides exactly what we need to choose search actions intelligently.

Let $p_{ij}(u)$ be the normalized prior probability density function for the future value of the utility estimate $\hat{U}^{\mathbf{S}.S_j}(A_i)$ after the search action S_j has been carried out. We make the following assumption:

Assumption 2 (Subtree independence):

> Search steps affecting the subtree from move A_i affect only the expected utility of the top-level move resulting from A_i, and not that of the other top-level moves.[13]

Hence we can immediately conclude that $\hat{U}^{\mathbf{S}.S_j}(A_i) = \hat{U}^{\mathbf{S}}(A_i)$ if S_j does not affect the subtree from move A_i.

A typical situation appears in figure 5.1.3, where we show the error curves corresponding to one search action for each top-level move.

There are only two cases in which search actions can have value, by changing the preferred move. Either further search on some non-preferred move β_i causes it to replace α, or search on α causes β_1, the current second-best move, to replace α (see figure 5.1.3).

Suppose we are considering the search action S_j, which affects the expected utility of move β_i. The search action will only change our preferred move if $\hat{U}^{\mathbf{S}.S_j}(\beta_i) > \hat{U}^{\mathbf{S}}(\alpha)$ (the shaded region to the right of $\hat{U}^{\mathbf{S}}(\alpha)$). If this happens, we expect to be better off by an amount $\hat{U}^{\mathbf{S}.S_j}(\beta_i) - \hat{U}^{\mathbf{S}}(\alpha)$; otherwise, the improvement is 0 since our move preference

[13]In many domains, such as standard game-playing programs using minimax as a back-up method, this assumption will be straightforwardly true, if we consider individual node expansions as single computational steps. However, if the search space is treated as a graph rather than a tree, as in some chess and go programs, then the analysis becomes slightly more complicated. In certain problem-solving systems a different approach must be taken. For example, if a computation involves refining a probability estimate used in an influence diagram, the new value may affect the utility of all the top-level actions.

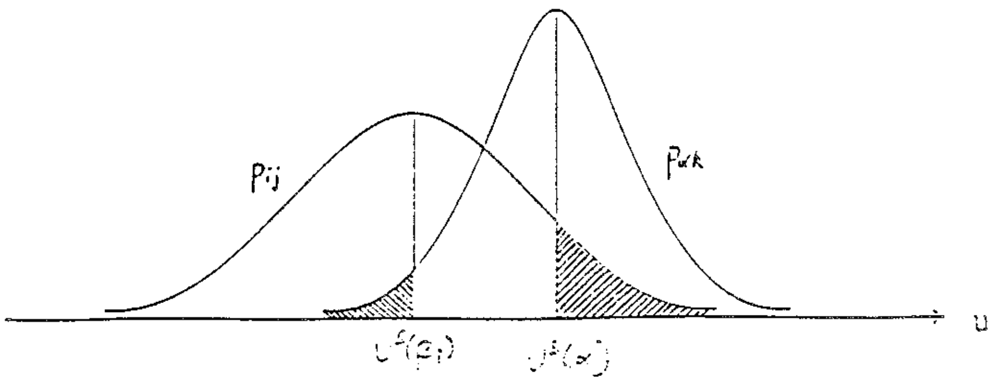

Figure 5: Effects of search actions

remains unchanged. Thus

$$E[V(S_j)] = \int_{\hat{U}^{\mathbf{S}}(\alpha)}^{\infty} p_{ij}(x)(x - \hat{U}^{\mathbf{S}}(\alpha))dx - C(S_j) \tag{24}$$

Similarly, if we perform a search action S_k on the subtree of the current best move, our move preference is changed only if $\hat{U}^{\mathbf{S}.S_k}(\alpha)$, the new expected value of our current preferred move, is less than $\hat{U}^{\mathbf{S}}(\beta_i)$ (where in this case β_i is β_1, the current second-best move). Although the new estimated utility of the new preferred move would be less than the current estimated utility of the current preferred move, we would still be *better off than we are*, since the search action will have revealed that α is a blunder. Hence

$$E[V(S_k)] = \int_{-\infty}^{\hat{U}^{\mathbf{S}}(\beta_1)} p_{\alpha k}(x)(\hat{U}^{\mathbf{S}}(\beta_1) - x)dx - TC(S_k) \tag{25}$$

As we would hope, the expected benefit of any search action is always non-negative, although this does not of course mean that we will be better off for doing it. Even disregarding the search cost, our expectations, as expressed in the static evaluation function, can be wrong.

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 $\hat{U}(A_i)$.

Before looking at specific applications, we can describe the *qualitative* behaviour of any algorithm based on our approach. Clearly, an agent will tend to forego further consideration of an action whenever its current estimated value and that of the best candidate are too far apart: in this case, it is unlikely that search will provide useful information, since the probability of changing action preference with any reasonable amount of extra computation is negligible. But search may also be pointless if the current values of the two nodes are too close together; in that case, it may be unlikely that search will reveal a *significant* difference between them. In an extreme case, the two moves may actually be symmetric, or nearly so, so that no amount of search will differentiate significantly between them. This case has received scant attention in the literature. B* in particular may be vulnerable on this account, since its sole goal is to prove one move better than all others. Lastly, if there is considerable uncertainty as to the values of the available actions, and considerable overlap, further computation is recommended. To shorten this discussion, we illustrate the three major situations graphically in Figure 5.1.3.

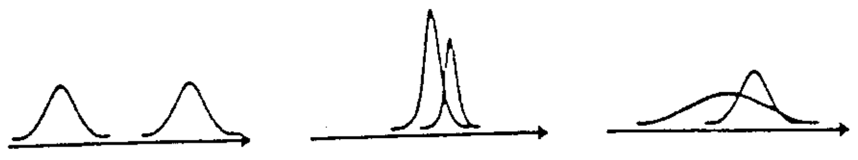

Figure 6: Three basic situations

5.2 Decision-Theoretic Real-time A*

In this section we examine a special case of the previous analysis, involving the application of the above ideas to real-time single-agent search, or, as Korf has dubbed it in [51,52], Real-time A*, in which the utility of a state is the minimum cost of a path to the 'goal state' constrained to go through that state. This is the standard situation in branch-and-bound search. In particular, we examine both the case of an admissible[14] heuristic function, and the more general case of a non-admissible heurstic function. If we assume a meta-greedy approach to search control, this case turns out to be fairly simple, but instructive. The main results are

1. If we make the single-step assumption, and consider only individual node expansions as single computation steps, then there is at most one frontier node that is worth expanding at any given time.
2. However, on this model it is possible, in fact highly likely, that the search tree will quickly reach a point where the expected value of all further search actions is calculated to be 0 or negative, even if the current estimates of move utilities is still highly uncertain: this phenomenon is called the "single-step barrier".[15]
3. But if we use a slightly more general model, in which the decision-theoretic calculation is applied to sequences of node expansions, all affecting the same top-level move, considered as an isolated computation, then it is always easy to pick out the minimal *set* $\mathcal{M}$ of nodes which must be expanded in order for further search to have positive expected value, and moreover the expected value of expanding all nodes in $\mathcal{M}$, or in any superset of $\mathcal{M}$ consisting of leaves descended from the current best move, can be computed by evaluating a single integral.

[14]The term *admissible* was originally defined as in [75] to mean any search algorithm that is guaranteed to find an optimal path to a goal, if one exists. In the context of the A* algorithm, a well-known sufficient condition for admissibility in this sense is that the cost-estimation function never over-estimate the true cost of reaching a goal from the current node. Following fairly common usage, we use the term "admissible" here only to refer to this sufficient condition.

[15]This fact is of particular interest because a similar barrier will be seen to occur in the case of game-playing programs.

4. Furthermore, the single-step assumption seems to be almost completely justified in the latter case.

5.2.1 Real-Time A*

Korf's Real-Time A*, or RTA*, algorithm is appropriate in situations where complete search to a goal is infeasible. It is basically just the familiar A* branch-and-bound procedure with a truncated search horizon, just as game-playing programs apply classical minimax with a truncated search horizon. The result of the search is not a complete path to a goal, but rather an educated guess as to the best first or next step on such a path. As with A*, the cost of getting to a solution via a given node n is estimated as $f(n) = g(n) + h(n)$, where $g(n)$ is an exact measure of the cost of getting to node n from the current position, and $h(n)$ is a heuristic estimate of the cost of getting from node n to the nearest goal. In Korf's formulation, RTA* involves searching forward to a fixed search depth, or alternatively a fixed $g(n)$ cost, and then backing up the costs of the frontier nodes. In this case, back-up is a trivial operation, since the backed-up cost of any top-level action is just the minimum cost[16] of any leaf node descended from that action.

Korf shows that given admissibility of the cost-estimation function $h(n)$, it is possible to avoid searching the entire tree down to the search horizon by following a branch-and-bound procedure, which he calls alpha-pruning[17]. Basically, alpha-pruning involves searching the tree in a depth-first manner, and setting the bound α equal to the lowest cost of any leaf node seen so far. Whenever an internal node is reached whose estimated cost exceeds α, we can prune the subtree under that node, since we know that the estimated cost is a lower bound on the actual cost.

We will call our algorithm, which involves the addition of meta-greedy decision-theoretic search control to RTA*, Decision-Theoretic A*, or DTA*.

5.2.2 DTA* with an Admissible Heuristic

A. Throughout this section we will be employing the single-step assumption, in which possible computational actions are evaluated as if they were isolated computations; i.e., as

[16] We deliberately use the term "cost" rather than "value" here, because we will later use the term "value" in the sense of the negative of cost.

[17] In fact, Korf shows that monotonicity, or equivalently consistency or obeying the triangle inequality (see Pearl, [80], p. 82) is sufficient in order for alpha-pruning to be justified. But Korf points out that, given a non-monotonic admissible function, "a monotonic function can trivially be constructed by taking its maximum value along the path". In fact, it should be obvious that one will always want to do this, given an admissible cost estimation function. Since the estimate at any point is a lower bound on the true cost, to use a lower estimate at a later point on the same path would be to throw away information. Conversely, any monotonic cost function must be admissible, provided the algorithm is infallibly capable of recognizing a goal node, so that $f(n) = g(n)$ when n is a goal. For a non-admissible function would have to over-estimate the cost of getting to a goal at some point, and therefore it would have to decrease between that point and the goal; hence it would be non-monotonic. Thus monotonicity is essentially equivalent to admissibility in practice.

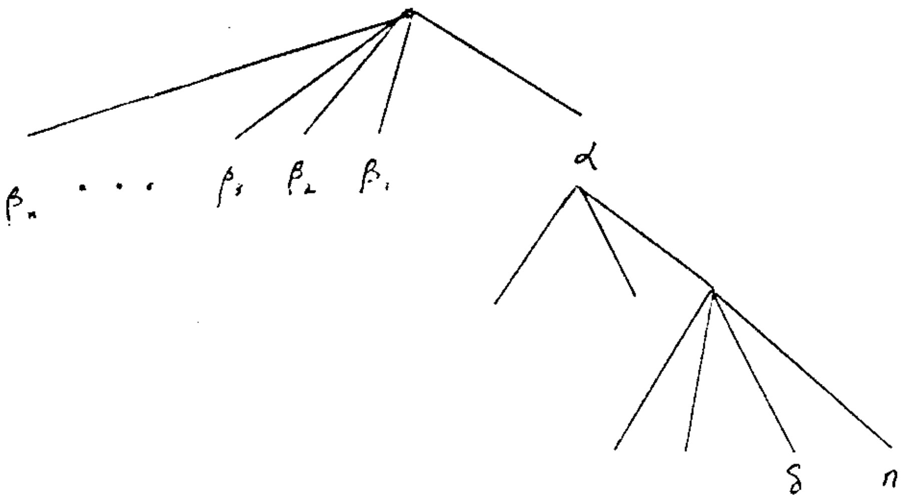

Figure 7: The value of a node expansion.

if they would necessarily be immediately followed by a commitment to act. We will first consider the most specific application of this assumption, in which only individual node expansions are considered single computation steps. In this case, deciding which node, if any, to expand is trivial. Only the frontier node with the minimum estimated cost is worth expanding.

Let α be the current best (lowest-cost) top-level action, and $\beta_1, \beta_2, \ldots$ the current second-best, etc. Recall that search is worthwhile on the single-step assumption only if it causes us to change our choice of top-level action. The only ways to do this in a single node expansion are either to increase the cost estimate of α, the current best move, or decrease the cost estimate of one of the alternative moves β_i. But we cannot do this by decreasing the estimated cost of β_i, since the current estimate $f(\beta_i)$ is a lower bound on the true cost of β_i. Hence it is only worthwhile to search nodes in the sub-tree under α, and only if doing so can increase the estimated cost of α. But since the backed-up cost of α is always equal to the minimum cost of any leaf node under α, and search can only increase the estimated cost of any leaf node, only the leaf node under α with minimum cost estimate will be worth expanding. Thus, there will be at most one node with positive expected search value. *This node is precisely the one which would be expanded next in a "best-first" search.*[16]

Let n be the unique leaf node with minimum cost; hence currently $f(\alpha) = f(n)$. Suppose we expand n, and let $f'(n)$ be the new backed-up cost of node n after expansion. (See figure 7.) It does not automatically follow that $f'(\alpha) = f'(n)$. For instance, suppose n has a sibling m such that $f(n) < f(m) < f'(n)$. Then the new backed-up cost of α will be at most $f(m)$, rather than $f'(n)$. Thus, it is clear that there will be an upper bound, δ, on how far the backed-up cost estimate of action α can be increased by increasing the cost estimate of node n. Clearly, δ is simply the cost of the leaf node under α with second-lowest cost estimate, prior to the expansion of n. If there is only one leaf

[16]In case of ties there could be more than one node with minimum cost estimate; in that case search is only worthwhile if *all* of the nodes with minimum cost estimate are searched. This case will be covered in the next section.

node, then $\delta = \infty$.[19]

Clearly, if $\delta < f(\beta_1)$, then no single node expansion will have positive expected search value on the single-step assumption since in that case it will be impossible to raise the cost of α above that of β_1 by a single node expansion. The point at which this situation first obtains we call the "single-step barrier". Another way of describing this point is that it is the point at which the *conspiracy number* of the tree increases from 1 to 2. From that point, isolated single node expansion steps have no value.

Otherwise, if $\delta > f(\beta_1)$, it follows that the expected value of expanding frontier node n is

$$E[\hat{V}(S_n)] = \int_{f(\beta_1)}^{\delta} p_n(x)(x - f(\beta_1))dx + \int_{\delta}^{\infty} p_n(x)(\delta - f(\beta_1))dx \tag{26}$$

where p_n is the prior probability density function for $f'(n)$.

B. Here we will consider sequences of node-expansions as isolated computations on the single-step assumption. However, to simplify the choice of sequence, we will only consider sequences of expansions of leaf nodes, all of which are descendents of the same top-level move. We will argue that this restriction is justifiable in the case of an admissible heuristic, because given the admissibility assumption, it still follows that only frontier nodes under α are worth expanding at any given point. Of course it may turn out to be worthwhile to expand, say, a node under α *and* a node under β_1, making β_2 the new choice of move. But if so, it cannot hurt to expand the node under α first, and then, *after* β_1 has become the new best move, expand the node under β_1. Moreover, if the expansion of the node under α fails to increase the estimated cost of α above that of β_1, then it will not turn out to be worthwhile expanding the node under β_1 after all. Thus even in this case it seems clear that at any given time it is only worth considering expanding nodes in the subtree under the current lowest-cost move.

Furthermore, it is fairly easy to characterize the *minimal set*, $\mathcal{M}$ of nodes under α which must be expanded in order to alter the top-level move choice. Let $\mathcal{L}_\alpha$ be the set of leaf nodes in the subtree under α. Then $\mathcal{M} = \{n_i | n_i \in \mathcal{L}_\alpha \& f(n_i) < f(\beta_1)\}$. That is, *all* leaf nodes under α whose current cost estimate is less than $f(\beta_1)$ must have their cost estimates raised above $f(\beta_1)$ in order to change the choice of best move from α to β_1. Let $f'_{\min} = \min\{f'(n_i) | n_i \in \mathcal{M}\}$ where for each n_i $f'(n_i)$ is the new cost estimate for node n_i after expansion. Clearly, $f'(\alpha) \leq f'_{\min}$, and $f'(\alpha) = f'_{\min}$ as long as $f'_{\min}$ is less than the current cost estimate of any node in $\mathcal{L}_\alpha - \mathcal{M}$. Let $p_{\min}$ be the prior probability density function for $f'_{\min}$. Then the expected value of expanding all of the nodes in $\mathcal{M}$ is:

[19]Equivalently, let $\delta = \min\{\delta_1, \delta_2, \ldots \delta_k\}$, where $\{\delta_1, \delta_2, \ldots \delta_k\}$ are the estimated costs of all of the second-best children of nodes along the path from α to n ($\delta_i = +\infty$ if a node has only one child). Then if $f'(n) < \delta$, we will have $f'(A) = f'(n)$, and otherwise $f'(A) = \delta$. This formulation will be recalled later on, when we are discussing game-playing algorithms.

$$E[\hat{V}(S_{\mathcal{M}})] = \int_{f(\beta_1)}^{\delta} p_{\min}(x)(x - f(\beta_1))dx + \int_{\delta}^{\infty} p_{\min}(x)(\delta - f(\beta_1))dx. \qquad (27)$$

In this case, δ is just the lowest cost of any leaf node in $\mathcal{L}_\alpha - \mathcal{M}$, or $+\infty$ if $\mathcal{M} = \mathcal{L}_\alpha$.

Further, let $\mathcal{M}'$ be any superset of $\mathcal{M}$ such that $\mathcal{M} \subset \mathcal{M}' \subset \mathcal{L}_\alpha$. Then the same formula gives the expected value of searching all of the nodes in $\mathcal{M}'$, where the definitions of $p_{\min}$ and δ are revised accordingly.

In this case, there is no "single-step barrier". That is, there always exists some subset $\mathcal{M} \subset \mathcal{L}_\alpha$—perhaps $\mathcal{L}_\alpha$ itself—such that searching all of the nodes in $\mathcal{M}$ can conceivably result in a change in the choice of top-level action. Moreover, it seems intuitively clear that, other things being equal, the leaf node with lowest current cost estimate should be expanded first, since if its cost estimate is not raised by expansion, expanding the others in the set $\mathcal{M}$ will not be worthwhile. Thus, again it appears that the "best-first" criterion of A* is justifiable on decision-theoretic grounds.

Note that even if we do not make the single-step assumption, or do not even bother to try to estimate numerically the expected value of a contemplated sequence of node expansions, we can still make the following assertion unqualifiedly, which so far as the authors are aware has not before been noticed by researchers on heuristic search: *Unless sufficient computational resources are available to expand all of the leaf nodes in the set $\mathcal{M}$ as defined above, search should cease immediately, since it will not be possible to alter the current move choice with the resources remaining.*

C. *Estimating Error Distributions and the Cost of Search*

We will call the difference $f'(n) - f(n)$ between the backed-up cost estimate of node n after one ply of search, and its static cost estimate, the *1-ply error* for node n. Before the above ideas can be incorporated in an algorithm, we need some way of estimating the distribution of the 1-ply error in the heuristic cost-estimation function, as well as the cost of search.

Our approach to the task of learning the error distributions is actually quite simple. We randomly sample the domain, and at each point apply the cost-estimation function without search, then estimate the cost of the same node at various depths of search. A relatively small sample of such points will be sufficient to characterize the distribution of the error to a high degree of accuracy. In some cases, these distributions can be parametrized, making computation of node search values a relatively straightforward matter.

Estimating the cost of search involves computing the trade-off between further search on the current move, and search on future moves. The nature of this trade-off is highly domain-specific, and there is little of a completely general nature that we have to say about it. In section 5.3 we will describe a time-cost estimation procedure that we found to work fairly well for the game of Othello.

Finally, note that, as we have pointed out previously, a lot of information on the cost of search can be derived just from the size of $\mathcal{M}$. From the above formula for the expected value of $V(S_{\mathcal{M}})$, it seems clear that on average, when $\mathcal{M}$ is twice as large, it will require about twice as much search to achieve about the same expected value. Compare the use of "conspiracy numbers" as a stopping criterion in McAllester's work.

5.2.3 DTA* With a Non-Admissible Heuristic

To a certain extent, this case is fairly similar to the case of an admissible heuristic, though more complex. We will discuss it briefly here.

A. Again, we start by considering single node expansions as isolated computations. Let $\mathcal{L}_B$ be the set of leaf nodes under any of $\beta_1, \beta_2, \ldots$ With an admissible heuristic, none of the nodes in $\mathcal{L}_B$ are relevant; i.e. changing the value of any of these can have no immediate effect on the choice of current best move. Here the situation is exactly the opposite. Since the cost estimate of any node can decrease by an arbitrary amount, the estimated cost of any leaf node in $\mathcal{L}_B$ could conceivably decrease below $f(\alpha)$, causing its top-level ancestor to become the new default move choice. Thus all of the nodes in $\mathcal{L}_B$ are relevant. Moreover, in this case there is no limit, δ, on the amount by which the estimated cost of the default move choice can change. Thus the expected value of expanding a node n in $\mathcal{L}_B$ descended from β_i will be:

$$E[\hat{V}(S_n)] = \int_{-\infty}^{f(\alpha)} p_n(x)(f(\alpha) - x)dx \tag{28}$$

where p_n is the prior probability density function for $f'(n)$.

For a node in $\mathcal{L}_\alpha$, the situation is basically as it was for the admissible case. Since searching a node in $\mathcal{L}_\alpha$ is worthwhile only insofar as it can cause the backed-up cost of α to increase above $f(\beta_1)$, only the unique frontier node n of minimum cost in $\mathcal{L}_\alpha$ such that $f(n) = f(\alpha)$ is relevant to α. As before, the amount by which $f(\alpha)$ can be raised is limited by δ, the cost of the second-lowest-cost node in $\mathcal{L}_\alpha$. Thus, there is still a "single-step barrier" on this side of the search tree, but not for the tree as a whole. The expected value of expanding a node n in $\mathcal{L}_\alpha$ is still given by equation 26

B. We will now consider sequences of node expansions in the same subtree as isolated computations. We can extend the results for the single-node model in an analogous way to what we did for the admissible case. However, in this case it is much less clear that the single-step assumption is justifiable even for such sequences. For it is possible that first raising the cost of α, and subsequently lowering the cost of β_1, could cause β_1 to unseat α as the default move choice, even if the initial increase in the cost of α does not bring it above the current cost of β_1. Thus, expanding a set of nodes in $\mathcal{L}_\alpha \cup \mathcal{L}_B$ could have positive expected value even when no node or set of nodes in $\mathcal{L}_\alpha$ has positive expected search value.

It is clear that equation 27 still holds for the set $\mathcal{M}$ as previously defined (and also its extension to any superset $\mathcal{M}'$ such that $\mathcal{M} \subset \mathcal{M}' \subset \mathcal{L}_\alpha$). An analogous extension of equation 28 holds as well, for any subset of the nodes in $\mathcal{L}_B$. That is, let $\mathcal{N} \subset \mathcal{L}_B$ be any subset of the leaf nodes in $\mathcal{L}_B$, and let $f'_{\min} = \min\{f'(n_i) | n_i \in \mathcal{N}\}$. Let $p_{\min}$ be the prior probability density function for $f'_{\min}$. It is clear that as long as $f'_{\min} < f(\alpha)$, the corresponding β_i will become the new default move choice, with $f(\beta_i) = f'_{\min}$. Thus the expected value of expanding all of the nodes in $\mathcal{N}$ is given by

$$E[\hat{V}(S_{\mathcal{N}})] = \int_{-\infty}^{f(\alpha)} p_{\min}(x)(f(\alpha) - x)dx \tag{29}$$

5.3 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. We describe the implementation of the resulting control regime in a new[20] algorithm, MGSS*.

In designing a game-playing algorithm, as with any other machine, the choice of computation, in this case the choice of search steps, should be made so as to maximize the system's overall utility. Overall utility is determined by the move chosen and the time at which it is made. As we discuss below, the main stream of game-playing literature has either used heuristic quasi-utilities [10,77,78,82,69,95] or fixed policies (such as alpha-beta search with a depth limit) to select search steps. A system with low-overhead, optimal control of search would be guaranteed to win, on average, against any other program with equal computational resources and domain-specific knowledge.

Our objectives in this domain are to examine the degree to which a reasonable approximation to optimal control of search can be implemented with minimal overhead; and to study the qualitative properties of the resulting search behaviour. As in the discussion of single-agent search, we make a number of assumptions that appear appropriate for this domain, in order to arrive at a formula for the expected value of a search step that can be calculated quickly.

Our first assumption is that the utility function is approximately separable into temporal and intrinsic components, as described above. In the game-playing domain, where there is typically a minimum number of moves that have to be made within some fixed period, this assumption means that the time cost TC of a computation will just be a function of the length of the computation, the remaining time allowed, the number of moves that have to be made and possibly some features of the current board situation; it will ignore the effect of, for example, quickly choosing a move under time pressure that complicates the game rather than simplifying it. This can be partially compensated for by including some time factors in the board evaluation function. The implementation section discusses in greater depth a method for estimating TC. Here we concentrate on how search steps can affect $\hat{U}$.

Intrinsic move utilities can be measured on a real-valued scale, corresponding to the

[20]An early version of some of the ideas and implementation described in this section were discussed in our paper [86]. In that paper, this algorithm was called simply MG*, because in the earlier formulation we did not carefully distinguish the meta-greedy assumption from the single-step assumption. There are other, less important differences from the earlier version which we will not discuss in detail.

values returned by a normal, static evaluation function. It is important to note that the static evaluation function is not to be viewed as an *estimate of the true value* of the position, but as an *expected utility*. The true value for a game such as chess is 0 or ± 1 (on a scale in which 1 is a win); the expected utility is a computationally-bounded, probability-weighted value $p_{win} - p_{loss}$ [37].

In the following analysis, we use both the meta-greedy and single-step assumptions, with individual node expansions as the chosen computational unit of analysis. We develop an efficiently computable formula for evaluating the integrals in equations 24 and 25. The principal tool is the re-expression of the probability distributions p_{ij} in terms of the error distribution p_{jj} at the leaf node affected by the search action S_j. Since p_{jj} is a function of only the board situation, and not of the game tree, it can be estimated from empirical data on the results of previous tree searches.

5.3.1 Backing up error distributions

In order to apply formulae 24 and 25, we must be able to compute the expected gain in utility of a node *at the top level*, resulting from the expansion of a particular *leaf* node. Although the relation between the new leaf node value and the new top-level move value is more complicated here than in the single-agent case, it turns out that computation of search values is not a very difficult task, and can be accomplished using only information available at the leaf node itself at the time the expected utility computation is made.

Recall that α is the current best top level move, with backed-up value $\hat{U}^{\mathbf{S}}(\alpha)$, and let β_1, β_2, etc. be the second-, third-, etc. best moves, with current utilities $\hat{U}^{\mathbf{S}}(\beta_1)$, and so on. The estimated values of the top-level moves will be calculated by minimax backup from the leaf node values in the tree generated by computation $\mathbf{S}$. The search action S_k is defined as an expansion of leaf node k (to some fixed depth or node limit) followed by calculation and propagation of its new minimax value.

We now compute the set of nodes that can have non-zero search value:

Definition 1 (Relevance):

> We will say that a given node n is *"Relevant"* to move α, if and only if changing the value of node n can immediately cause the value of α to drop below $\hat{U}^{\mathbf{S}}(\beta_1)$. Similarly, we will say that node n is *Relevant to* move β_i, for $i \geq 1$, if and only if changing the value of node n can immediately cause the value of β_i to rise above $\hat{U}^{\mathbf{S}}(\alpha)$.

If S_j is the search action of expanding leaf node j, recall that $\hat{U}^{\mathbf{S}.S_j}(\alpha)$ is the value of move α after the search action is performed (at which time α may no longer be the best move). Similarly, $\hat{U}^{\mathbf{S}.S_j}(\beta_i)$ is the value of β_i after search action S_j.

It is clear, from the subtree independence assumption, that a given node can be Relevant to at most one move, namely its top level ancestor in the tree. Further, expanding a node that is not Relevant to any top level move cannot cause an *immediate* change in the choice of best move, and hence expanding such a node will have zero benefit on the single-step assumption.

The following theorem yields a formula for computing the expected value of searching a leaf node k that is Relevant to α. A dual of this theorem also holds for nodes Relevant to β_i for $i \geq 1$.

Theorem 1 (Relevant nodes):

1. The following computes the set of nodes Relevant to α:
   (a) α is Relevant to α.
   (b) If n is a min node and n is Relevant to α, then all children of n are Relevant to α.
   (c) If n is a max node and n is Relevant to α, then the best child of n is Relevant to α *provided* the second-best child of n has current value $< \hat{U}^{\mathbf{S}}(\beta_1)$. Otherwise *no* descendant of n is Relevant to α.
2. If a node n is Relevant to α, then $\hat{U}^{\mathbf{S}.S_k}(n) < \hat{U}^{\mathbf{S}}(\beta_1)$ if and only if $\hat{U}^{\mathbf{S}.S_k}(\alpha) < \hat{U}^{\mathbf{S}}(\beta_1)$.
3. If n is Relevant to α, then if $\hat{U}^{\mathbf{S}.S_k}(n) < \hat{U}^{\mathbf{S}}(\beta_1)$ then $\hat{U}^{\mathbf{S}.S_k}(\alpha) = \max(\delta, u_{nk})$, where $\delta = \max\{\delta_1, \ldots, \delta_k\}$, where $\{\delta_1, \ldots, \delta_k\}$ are the values of all of the second-best children of max nodes on the path from α to n.[21]

All three statements are easily proven by induction on the depth of node n, and the proof is omitted here.

Consider now a leaf node k that is Relevant to α. On the single-step assumption, the action of expanding k will have positive value only if it results in the lowering of α's value below $\hat{U}^{\mathbf{S}}(\beta_1)$, i.e., only if $\hat{U}^{\mathbf{S}.S_k}(\alpha) < \hat{U}^{\mathbf{S}}(\beta_1)$. In fact, the expected benefit of expanding node k is given by

$$E(\Delta(S_k)) = \int_{-\infty}^{\hat{U}^{\mathbf{S}}(\beta_1)} \mathrm{p}_{\alpha k}(x)(\hat{U}^{\mathbf{S}}(\beta_1) - x)\mathrm{d}x, \tag{30}$$

where $\mathrm{p}_{\alpha k}$ is the probability density function of the random variable $\hat{U}^{\mathbf{S}.S_k}(\alpha)$. From the above theorem, parts 2 and 3, it follows that an equivalent expression for $\Delta(S_k)$ is

$$E(\Delta(S_k)) = \int_{-\infty}^{\delta} \mathrm{p}_{kk}(x)(\hat{U}^{\mathbf{S}}(\beta_1) - \delta)\mathrm{d}x - \int_{\delta}^{\hat{U}^{\mathbf{S}}(\beta_1)} p_{kk}(x)(\hat{U}^{\mathbf{S}}(\beta_1) - x)\mathrm{d}x, \tag{31}$$

where p_{kk} is the density function of the random variable $\hat{U}^{\mathbf{S}.S_k}(k)$.

We pass quickly over the dual of the above Theorem, which yields a formula for the value of searching a node j Relevant to a top level move other than the current best.

Theorem 2 (Relevant nodes):

[21] Compare footnote, p. 31 in section 5.2 above.

1. The following computes the set of nodes Relevant to β_i:
   (a) β_i is Relevant to β_i.
   (b) If n is a max node and n is Relevant to β_i, then all children of n are Relevant to β_i.
   (c) If n is a min node and n is Relevant to β_i, then the best child of n is Relevant to β_i *provided* the second-best child of n has current value $> \hat{U}^{\mathbf{S}}(\alpha)$. Otherwise *no* descendant of n is Relevant to β_i.
2. If a node n is Relevant to β_i, then $\hat{U}^{\mathbf{S}.S_j}(n) > \hat{U}^{\mathbf{S}}(\alpha)$ if and only if $\hat{U}^{\mathbf{S}.S_j}(\beta_i) > \hat{U}^{\mathbf{S}}(\alpha)$.
3. If n is Relevant to β_i, then if $\hat{U}^{\mathbf{S}.S_j}(n) > \hat{U}^{\mathbf{S}}(\alpha)$ then $\hat{U}^{\mathbf{S}.S_j}(\beta_i) = \min(\delta, \hat{U}^{\mathbf{S}.S_j}(n))$, where $\delta = \min\{\delta_1, \ldots, \delta_k\}$, where $\{\delta_1, \ldots, \delta_k\}$ are the values of all of the second-best children of min nodes on the path from β_i to n.

Again we omit the proof by induction on the depth of the tree. In this case, the theorem yields the following formula for the value of expanding a leaf node j that is Relevant to β_i for $i \geq 1$:

$$E(\Delta(S_j)) = \int_{\delta}^{\infty} \mathrm{p}_{jj}(x)(\delta - \hat{U}^{\mathbf{S}}(\alpha))\mathrm{d}x + \int_{\hat{U}^{\mathbf{S}}(\alpha)}^{\delta} \mathrm{p}_{jj}(x)(x - \hat{U}^{\mathbf{S}}(\alpha))\mathrm{d}x \qquad (32)$$

Methods for computing the integrals are given in section 5.3.4.

5.3.2 Implementation

In this section we will describe an implementation of the above ideas which we have carried out. Although we have implemented a program that works reasonably efficiently, we expect that many improvements are still possible, and will be needed to deal with extensions to the theory.

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 *GameValue* of the node, i.e., its *StaticValue* if it is a leaf node, or its current backed-up value otherwise;

4. the δ 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 link pointing to a list containing the node's *Children*. The list is maintained in order of *GameValue*.

Relevant leaf nodes with positive *SearchValue* 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 $E(\Delta(S_j))$ using equation 31 or 32.
   Estimate the time-cost $TC$ of expanding node $j$. If $E[\Delta(S_j) - TC] \leq 0$ then return
   the first element in the Children of the Root as the best move.
3. Otherwise
   (a) Carry out the computation $S_j$. 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 $\delta$ 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.
```

5.3.3 Estimating leaf-node error distributions

In order to compute the value of search steps, formulæ 31 and 32 require information as to the error distributions of the leaf nodes. For search steps consisting of a single node expansion, such as are used in MGSS*, this is simply defined as the *a priori* distribution of the difference between the leaf's static value and its backed-up value from a depth one search.

This distribution is thus statistically well-defined, and, like the choice of an evaluation function, should be computed not at run time, but at the time a program is developed for a specific game, by means of statistical sampling techniques. By collecting a large enough number of data points, and categorizing them according to game situation, one can obtain information about the functions p_{jj} accurate to an almost arbitrary standard of approximation. That is, we can approximate p_{jj} for a particular node j by a function $p_\sigma(u)$, where σ matches the game situation at node j. We can choose values for σ that are more or less fine-grained, depending on our computing resources and the limit of obtainable accuracy. Finding an error estimator should be much easier than finding a good evaluator, since the feedback is immediate rather than depending on the final game outcome.

If the distributions obtained in this way are normal distributions, then all necessary information about them can be stored using two parameters, the mean μ and standard deviation σ. The hypothesis of normal distributions is empirically testable. We have in fact found it to be true to a high degree of accuracy for a particular static evaluation function we are using for the game of Othello.

35,000 positions were generated, and error values were computed for each. Means and standard deviations were then computed for roughly 1000 buckets of data points categorized by six board features. Given these leaf distributions, the computation of node search values using formulæ 31 and 32 can be done using only one table of integrals, as shown below.

5.3.4 Computing the search-value expressions

Recall the equations 31 and 32 for computing the *SearchValue* (expected benefit) of the search action S_j applied to node j. In order to enable the expected benefit to be computed, the MGSS* algorithm maintains the values δ, $\hat{U}^{\mathbf{S}}(\alpha)$ and $\hat{U}^{\mathbf{S}}(\beta_i)$, and the distributions p_{jj} are available by computing the appropriate features of the board position j and using them to index into the stored table of error parameters μ and σ. Replacing p_{jj} with the normal distribution function $N_{\mu,\sigma}$, we obtain two equations composed of four integral expressions each containing four parameters. It seems it would be impossible to compute and store a four-dimensional integral table that would provide the needed discrimination along the dimensions. As it happens, by a miracle of high-school mathematics described in detail in [100], the integral expressions can be reduced to simple arithmetic expressions involving only one integral expression, namely

$$\Phi(x) = \int_{-\infty}^{x} N_{0,1}(x)dx.$$

This integral is tabulated in any statistical textbook or tables.

The new equations, assuming the leaf node error distributions can be approximated by $N_{\mu,\sigma}$, are

$$\begin{aligned}\Delta(S_k) &= \sigma\left(N_{0,1}\left(\frac{\hat{U}^{\mathbf{S}}(\beta_1)-\mu}{\sigma}\right) - N_{0,1}\left(\frac{\delta-\mu}{\sigma}\right)\right) \\ &- (\delta-\mu)\Phi\left(\frac{\delta-\mu}{\sigma}\right) + (\hat{U}^{\mathbf{S}}(\beta_1)-\mu)\Phi\left(\frac{\hat{U}^{\mathbf{S}}(\beta_1)-\mu}{\sigma}\right) \qquad (33)\end{aligned}$$

corresponding to (31), and

$$\begin{aligned}\Delta(S_j) &= \sigma\left(\mathcal{N}_{0,1}\left(\frac{\hat{U}^{\mathbf{S}}(\alpha)-\mu}{\sigma}\right)-\mathcal{N}_{0,1}\left(\frac{\delta-\mu}{\sigma}\right)\right)\\ &- (\delta-\mu)\Phi\left(\frac{\delta-\mu}{\sigma}\right)+(\hat{U}^{\mathbf{S}}(\alpha)-\mu)\Phi\left(\frac{\hat{U}^{\mathbf{S}}(\alpha)-\mu}{\sigma}\right)\\ &+ (\delta-\hat{U}^{\mathbf{S}}(\alpha))\end{aligned} \tag{34}$$

corresponding to (32). Armed with these formulæ, computing search values is usually somewhat quicker than computing the static evaluation function.

5.3.5 Estimating deliberation cost

The cost estimation function can also in principle be determined empirically, though one would probably need a much larger amount of data to get accurate values. Assume that the average time needed to perform a search step is t, where t could be, say, a millisecond. Then we want to determine the effect of wasting a millisecond, in a given game situation, on the probability of ultimately winning the game. Given our probability-based definition of the evaluation function, we then have a common utility scale for moves and time costs. In practice, it is sufficient to have TC 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. Since Othello tournaments impose a per-game time limit, we decided to adopt such a limit for our implementation, even though we had no intention, at least at this early stage, of producing a tournament-level Othello program. (Besides, the time cost calculation given a per-move time limit is trivial; the cost of a node expansion is 0 as long as the time limit is not exceeded, and infinite if it is.) For the purpose of testing the implementation, we set a per-game time limit in terms of numbers of nodes expanded, rather than seconds, since we wanted a direct test of the selectivity and power of the decision-theoretic search, independently of its overhead.[22]

Another way to look at the time cost calculation is that it measures the trade-off between present and future node expansions. That is, the possible gain from a further node expansion in the present situation must be weighed against the possible loss from being deprived of the ability to perform a node expansion in some future situation. To analyze this trade-off, we made two heuristic assumptions, which to some extent are in tension with one another. The first is that, *other things being equal*, in default of any discriminating information, each move in the game should be allocated a roughly equal share of whatever nodes remain. The second heuristic assumption was that numerically equal gains in the

[22]In fact, the overhead was low enough to be very encouraging, considering that this was a first implementation, not carefully coded for speed. While we saved about 40% over alpha-beta in terms of node expansions, we roughly broke even in terms of total time per game.

backed-up values of moves at different points in the game are of equal utility. Based on the second assumption, we assumed that the cost of a single node expansion is given approximately by the *average gain* from expanding a node over the remaining moves of the game, where this average gain, like the error distributions, has been pre-computed statistically. To give weight to the first assumption, we did not simply set the time cost equal to the average gain per node expanded. Instead, we assumed that the cost of expanding another node in the current situation is given by $TC = f(e) * a$, where a is the average gain from a node expansion over the remaining moves, and $e =$ the elapsed time for this move; i.e., the number of nodes expanded on the current move so far. Further, f is an increasing function of e, such that $f(t) = 1$, where $t =$ (number of nodes remaining)/(number of moves remaining); i.e., t is the number of nodes per move if all moves are allocated an equal share of the remaining nodes. In our implementation, we simply used $f(e) = e/t$.

Obviously, there is a great deal of room for invention and trial and error in the design of a heuristic time cost estimator. The problems involved constitute in themselves a whole area of research which to date remains largely unexplored. We hope that further refinements in this area will be forthcoming as we add new experience with implementations in a variety of domains.

5.3.6 Performance

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 5.1.3).

It may be possible to show, although we have not done so, that the pruning scheme of alpha-beta search is generated automatically by the formulæ 24 and 25. Examination of the trees MGSS* generates, and of the general depth-two tree, indicates that this may in fact be the case. More importantly, the generation of the pruning strategy has been a *constructive* process.

In order to demonstrate the falsity of the single-step assumption in practice, we have implemented a version of MGSS* that assumes that *all* Relevant nodes should be expanded. This algorithm usually terminates wih a tree of about 500-1000 nodes, roughly equivalent to a full depth-three search, although the actual trees had depths up to 20. The termination indicates that at some point, it will be the case that no single search steps have value, although sequences of steps may still be useful. Note that termination occurs regardless of the cost of time used. Even without search ordering, this algorithm seems to perform as well as an alpha-beta search using the same evaluation function, and roughly the same number of nodes searched per game.

With search ordering using $E(\Delta(S))$ and time costs for pruning, our algorithm wins about 44% of its games against alpha-beta, while expanding about 40% fewer nodes per game than alpha-beta expands. In terms of numbers of discs won, MGSS* looks even better, winning a total of about 95% as many discs as alpha-beta.

Surprisingly, our implementations show that the 'constant factor', time per node expanded, for MGSS* is comparable to that of an alpha-beta search using the same basic technology. In other words, the overhead of meta-level control is relatively small. Neither our implementation of alpha-beta nor of MGSS* was very carefully engineered for speed. On average, alpha-beta used about .13 seconds per node expanded, while MGSS* used about .22 seconds per node.[23]

The fact that MGSS* search seems to have a low upper limit on the number of nodes it will ever search means that comparisons against a deeper-searching alpha-beta (say 100,000 nodes per move) are unlikely to be favourable, although it does sometimes search interesting lines very deeply. 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. An extremely simplified form of this method appears in the work on *singular extensions* by members of Berliner's group [15].

5.3.7 Detailed comparison to other work on game-playing

The most closely related work, in spirit, to our decision-theoretic approach to game-playing has been the proposals due to Good [37,38]. Good suggests using a real utility measure on search steps to decide when to prune (although not how to order search). His formula for the utility , as far as can be determined, seems to be irrational in practice, in that he compares the utility of the best move after the search step with the *current* utility of the current best move. This assigns a high value to search steps which simply revise the value of the current best move upwards; but this gain is illusory, since the true value of the move remains fixed. Similarly, a search step that shows the current best move to be a blunder is given a negative value, whereas it is actually highly desirable.

Although Rivest [82] does not explicitly employ probabilistic considerations, there is some affinity between his method and ours. Consider the line of best play, starting at the root node. If we could infallibly discover this line of play in advance, we could completely determine the value of the root without having to search any of the rest of the tree. Simply "guessing" the line of best play, for instance on the basis of the static evaluations of the nodes along a given line, would not be adequate, since such guesses could often go drastically wrong. Any method which tends to concentrate search along the line of best play, while searching with sufficient width to lessen the probability of missing the actual line of best play entirely, will result in gains from the point of view of use of resources. Rivest's method and ours both try to make this width/depth tradeoff. However, we are not entirely convinced by his criterion for selecting search actions. In the iterative context, it seems to us that we should expand that tip node whose expansion will have the greatest *expected benefit* in terms of move quality. Rivest's policy seems to approve of a search action that simply revises the utility of the current best move upwards, whereas in our framework this is valueless. Clearly, there exists a connection between the two approaches,

[23]Numerous savings in this overhead are readily available; for instance, in our initial implementation, the queue of Relevant nodes was simply a sorted list, but it could instead be implemented as a priority queue. Our next implementation should be more efficient.

but we have been unable to ascertain the link precisely.

As mentioned above, McAllester's theory of conspiracies [69] attempts to find sets of nodes such that searching them, and thereby changing their values, will have the largest effect on the root value, and thus seems to fall into the same problems. Also, although it seems intuitively plausible that larger conspiracies are less likely than smaller ones, the algorithm ignores the quantitative probabilities of the value changes being counted and the fact that node value changes in a game tree are far from independent. However, the methods used to identify the conspiracies are particular ingenious, and may be adapted for our purposes to identify sets of nodes that can change the current best move, in cases where the meta-greedy assumption breaks down.

The line of research on B* algorithms [10,77,78] is also closely related to ours in that it uses an explicit meta-level policy, and it uses error distributions. Berliner's original motivations, arising from the apparent irrationality of alpha-beta in doing search with only one legal move, are similar to our own. However, there are significant theoretical and practical differences. It seems that B*'s policy, of attempting to find the shortest proof that some move is better than all others, suffers from the difficulty, alluded to above, of trying to distinguish between close moves when it should be indifferent between them (see Figure 5.1.3). This appears to be because the meta-level goal being pursued is a quasi-utility, and does not equate to winning the game. This irrationality at the meta-level can lead to an infinite search when, for example, the system tries to choose between symmetrical moves. Moreover, we believe that a search of a small portion of the entire game tree can never 'prove' that one move is best — all it can do is suggest a probability. Therefore termination of search *must* depend on some notion of the current cost of time, rather than on any property of the partial game tree alone.

In addition, B*'s concept of error is quite different from that proposed in this paper. In B*, error is a deviation of the static value from the 'delphic' value of a position, that is, the value assigned by an idealized 'expert'. In other words, Berliner and Palay propose that there exists some correct number to which the evaluation function is an approximation. The intended semantics of this notion cannot be made coherent, since the true value is 0 or ± 1, and any estimate is a function of resource bounds. This difficulty can be avoided by using the value *after search*, as compared to the static value. If we can estimate the distribution of the value after search before the search is performed, this can help us to decide beforehand whether the search is likely to be worthwhile.

5.3.8 Further work: recursive search algorithms

A recursive, or problem-reduction algorithm could also be implemented, which would be less costly both in terms of time and space. This algorithm would work by iterative deepening. The algorithm would start by expanding the current position to determine the possible moves, and then call itself on the successors. The meta-level decision at each call would then be whether to expand a given node to another ply of search. The algorithm would use only linear space. Further, the cost-benefit comparison of possible search steps would be done at the top level of each recursive call, rather than in a best-first fashion over the whole subtree. The necessary error data would be the variance of the static evaluation function versus search to depth k, for different values of k.

A variant of this is the *iterative expansion* approach, which involves the use of an explicit resource bound, or node limit, rather than a depth limit. That is, the algorithm would be called with a number of nodes allocated to the search of the current move. It would then decide to allocate a portion of the node limit to the search of a particular move, and call itself recursively with that node limit on that move. The control decision at each step would then be a decision whether to search the same move with a greater node limit, to search a different move, or to terminate the search. To achieve the asymptotic behavior of iterative deepening, the node limit would be multiplied by a fixed constant each time a given node is searched.

By deliberating at the meta-level only about large collections of base-level steps, considerable time is saved. However, this naturally leads to more wasteful base-level search actions. Appropriate trade-offs remain to be investigated.

6 Summary

We see computational resource limitations as a major influence on the design of optimal agents. This influence has been neglected in classical theories of normative behaviour, with the result that practical AI systems for non-trivial domains are constructed in an *ad hoc* fashion. We have therefore undertaken research into three tightly-interconnected areas of study:

- the theory and practice of normative meta-level control of reasoning;
- the construction of a universal-subgoaling, decision-theoretic architecture;
- the possible forms of compiled and uncompiled knowledge within such an architecture, and compilation mechanisms for transforming between them.

In this report, we developed a framework for estimating the utility of computations, by focusing on the effects of computations in revising an agent's intended actions. The notion of real-time problem-solving was formalized within the framework, by taking into account the time-dependence of the utility of actions. We have applied the formal framework to analyze both single-agent problem-solving and competitive game-playing, in each case yielding new algorithms with improved performance. Performance systems for a variety of domains can be derived by making suitable approximations to the normative formula for meta-level control, and by proving associated simplifying theorems.

Overall, the meta-level architecture and the set of compiled knowledge types generates a rich space of possible agent designs. But more importantly, the fact that all the components in our proposed system have a well-defined semantics, and the fact that they are connected by normative inferential links, means that an agent can make well-motivated changes in its own configuration as it searches the space of possible designs for an optimal state.

References

[1] Agre, P. and Chapman, D. (1987) Pengo: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA: Morgan Kaufmann.

[2] Agogino, A. M. (1987) IDES: Influence Diagram Based Expert System. *Mathematical Modelling*, **8**, 227-233.

[3] Allard, J. R. and Kaemmerer, W. F. (1987) The goal/subgoal knowledge representation for real-time process monitoring. In *Proc. Sixth National Conf. on Artificial Intelligence*, Los Altos: Morgan Kaufmann, 394-398.

[4] Anderson, J. R. (1986) Knowledge Compilation: The General Learning Mechanism. In Michalski, R., Carbonell, J., and Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. II. Los Altos, CA: Morgan Kaufmann.

[5] Anderson, B. M., Cramer, N. L., Lineberry, M., Lystad, G. S., and Stern, R. C. (1984) Intelligent automation of energency procedures in advanced fighter aircraft. In *Proc. First Conf. on Artificial Intelligence Applications*, Washington, D. C.: IEEE Computer Society, 496-501.

[6] Andersson, R. L. (1987) Real time expert system to control a robot ping pong player. Ph.D. Dissertation, Department of Computer Science, University of Pennsylvania.

[7] Batali, J. (1985) A computational theory of rational action (draft). Cambridge: MIT AI Lab.

[8] Benjamin, D. P. (1987) Learning Strategies by Reasoning about Rules. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy: Morgan Kaufmann.

[9] Benjamin, D. P. (1988) A metalevel manifesto. In *Proceedings of the International Workshop on Machine Learning, Metareasoning and Logics*, Sesimbra, Portugal.

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

[11] Bratman, M., Israel, D., and Pollack, M. (in press) Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, to appear.

[12] Braverman B., and Russell, S. J. (1988) IMEX: Overcoming Intractability in Explanation-Based Learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, MN: Morgan Kaufmann, 575-579.

[13] Brooks, R. A. (1986) A robust, layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, **2**(1), 14-23.

[14] Bundy, A., Byrd, L, Luger, G., Mellish, C., Milne, R., and Palmer, M. (1979) Solving mechanics problems using meta-level inference. In *Proceedings of the Sixth International Joint Conf. on Artificial Intelligence*, Tokyo: Morgan Kaufmann.

[15] 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.

[16] Chen, D. C. (1985) Progress in knowledge-based flight monitoring. In *Proc. Second Conf. on Artificial Intelligence Applications*, Washington, D.C.: IEEE Computer Society, 441-446.

[17] Chester, D., Lamb, D., and Dhurjati, P. (1984) Rule-based computer alarm analysis in chemical process plants. In *Proc. Seventh Annual Conf. on Computer Technology*, Washington, D.C.: IEEE Computer Society, 22-29.

[18] Clippinger, J. H. (1983) An artificial intelligence system for the realtime monitoring and analysis of textual information. In *Proc. Trends and Applications Conf.*, Washington, D.C.: IEEE Computer Society, 65-67.

[19] D'Ambrosio, B., Fehling, M., Forrest, S., Raulefs, P., and Wilbur, B. (1987) Realtime process management for materials composition in chemical manufacturing. *IEEE Expert* **2** (2), 80-89.

[20] de Kleer, J., Doyle, J., Steele, G. L., Jr., and Sussman, G. J. (1977) AMORD: Explicit control of reasoning. *Proc. ACM Conference on AI and Programming Languages*, Rochester, New York.

[21] Dean, T. (1987) Intractability and time-dependent planning. In *The 1986 Workshop on Reasoning about Actions and Plans* (M. P. Georgeff and A. L. Lansky, eds.), Los Altos: Morgan Kaufmann, 245-266.

[22] Dean, T., and Boddy, M. (1988) An Analysis of Time-Dependent Planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, MN: Morgan Kaufmann, 49-54.

[23] Dean, T., and Kanazawa, K. (1988) Probabilistic temporal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, MN: Morgan Kaufmann.

[24] DeJong, G., and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, **1**.

[25] Doyle, J. (1980) A model for deliberation, action, and introspection. Cambridge: MIT Aritificial Intelligence Laboratory, Technical Report TR-581.

[26] Doyle, J. (1983) What is rational psychology? Toward a modern mental philosophy. *AI Magazine* **4** (3), 50-53.

[27] Doyle, J. (1988b) *Artificial Intelligence and Rational Self-Government.* Technical report no. CMU-CS-88-124. Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA.

[28] Doyle, J. (1988c) On rationality and learning. Computer Science Department Technical Report CMU-CS-88-123, Carnegie Mellon University.

[29] Durfee, E. H., and Lesser, V. R. (1988) Incremental planning to control a time-constrained, blackboard-based problem solver. In *IEEE Transactions on Aerospace and Electronic Systems*, (forthcoming). Also: Department of Computer and Information Science Technical Report 87-07, University of Massachusetts, Amherst, MA.

[30] Ennis, R. L., Klein, D., Milliken, K., Schor, M., Greismer, J., Hong, S., Karnaugh, M., Kastner, J., and Van Woerkom, H. (1986) A continuous real-time expert system for computer operations. *IBM Journal of Research and Development* **30** (1), 14-28.

[31] Fagan, L. (1980) VM: Representing time-dependent relations in a medical setting. Ph.D. Dissertation, Department of Computer Science, Stanford University.

[32] Fehling, M. R., and Breese, J. S. (1988) A computational model for decision-theoretic control of probkem-solving under uncertainty. In *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, Minneapolis, MN: AAAI.

[33] Fehling, M. R., Joerger, K., and Sagalowitz, D. (1986) Knowledge systems for process management. In *Proc. Instrument Society of America-86 Conf.* (Vol. 41, No. 3), Research Triangle Park, N.C.: Instrument Society of America, 1509-1526.

[34] Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972) Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, **4**, 251-288.

[35] Finger, J. A. (1987) *Exploiting Constraints in Design Synthesis.* Ph.D. Dissertation. Department of Computer Science, Stanford University.

[36] Genesereth, M., and Smith, D. (1981) Meta-level architecture. Stanford Heuristic Programming Project, Memo HPP-81-6, Stanford University, Stanford, CA.

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

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

[39] Harman, G. (1986) *Change of View: Principles of Reasoning.* Cambridge: MIT Press.

[40] Harmon, S. Y. (1983) Coordination between control and knowledge based systems for autonomous vehicle guidance. *Proc. Trends and Applications Conference*, Washington, D.C.: IEEE Computer Society, 8-11.

[41] Haussler, D. (1988) Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, **36**(2), 177-221.

[42] Hayes-Roth, B. (1987a) A multi-processor interrupt-driven architecture for adaptive intelligent control. Department of Computer Science Technical Report KSL-87-31, Stanford University.

[43] Heckerman, D., and Jimison, H. (1987) A perspective on confidence and its use in focusing attention during knowledge acquisition. In *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*, Seattle, WA: AAAI, 123-131.

[44] Horvitz, E. J. (1987) Problem-solving design: Reasoning about computational value, trade-offs, and resources. In *Proc. Second Annual NASA Research Forum*, Moffett Field, CA: NASA Ames, 26-43.

[45] Horvitz, E. J. (1988) Reasoning about beliefs and actions under computational resource constraints. In Uncertainty in Artificial Intelligence Vol. 3., (T. Levitt, J. Lemmer, and L. Kanal, eds.), Amsterdam: North Holland. Also: Department of Computer Science Technical Report KSL-87-29, Stanford University.

[46] Horvitz, E. J. (1988) Reasoning under varying and uncertain resource constraints. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, MN: Morgan Kaufmann, 139-144.

[47] Horvitz, E. J., Breese, J. S., and Henrion, M. (in press) Decision theory in expert systems and artificial intelligence. *Journal of Approximate Reasoning*, to appear.

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

[49] Hudlicka, E., and Lesser, V. R. (1984) Meta-level control through fault detection and diagnosis. In *Proc. Fourth National Conf. on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann, 153-161.

[50] Kao, S. M., Laffey, T. J., Schmidt, J. L., Read, J. Y., and Dunham, L. (1987) Real time analysis of telemetry data. In *Proc. Third Annual Expert Systems in Government Conf.*, Washington, D.C.: IEEE Comuter Society, 137-144.

[51] Korf, R. E. (1987) Real-time heuristic search: First results. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA: Morgan Kaufmann, 133-138.

[52] Korf, R. E. (1988) Real-time heuristic search: New results. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, MN: Morgan Kaufmann, 139-144.

[53] Knuth, D. E., and Moore, R. N. (1975) An analysis of alpha-beta pruning. *Artificial Intelligence* **6**, 293-326.

[54] Laffey, T. J., Cox, P. A., Schmidt, J. L., Kao, S. M., and Read, J. Y. (1988) Real-time knowledge-based systems. *AI Magazine*, **9**(1), 27-45.

[55] Laird, J. E. (1984) *Universal Subgoaling.* Doctoral dissertation, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA.

[56] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987) SOAR: An architecture for general intelligence. *Artificial Intelligence* **33**, 1-64.

[57] Laird, J., Rosenbloom, P., and Newell, A. (1986) Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning* **1** (1), 11-46.

[58] Leinweber, D., and Gidwani, K. (1986) Real time expert system development techniques and applications. In *Proc. WESTEX-86: IEEE Western Conference on Knowledge Based Engineering and Expert Systems*, Washington, D.C.: IEEE Computer Society, 69-77.

[59] Lenat, D. B. (1979) Cognitive economy in artificial intelligence systems. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo: Morgan Kaufmann, 531-536.

[60] Lenat, D., Davis, R., Doyle, J., Genesereth, M., Goldstein, I., and Shrobe, H. (1983) Reasoning about reasoning. *Building Expert Systems* (D. Waterman, R. Hayes-Roth, and D. Lenat, eds.), Reading: Addison-Wesley, 219-239.

[61] Lesser, V. R., and Corkill, D. D. (1983) The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine* **4**, 15-33.

[62] Lesser, V., Pavlin, J., and Durfee, E. (1988) Approximate processing in real-time problem-solving. *AI Magazine*, **9**(1), 49-61.

[63] Maes, P. (1986a) Introspection in knowledge representation. In *Proc. European Conf. on Artificial Intelligence*, Pisa, Italy.

[64] Maes, P. (1986b) Reflection in an object-oriented language. In *Proc. SPL-Insight Workshop*, unpublished.

[65] Maes, P. (1988a) Object-oriented reflection. In *Meta-Level Architectures and Reflection* (P. Maes and D. Nardi, eds.), Amsterdam: North Holland.

[66] Maes., P. (1988b) *Reflective, object-oriented architectures.* Ph.D. thesis, Artificial Intelligence Laboratory, Free University of Brussels.

[67] Marsh, J., and Greenwood, J. (1986) Real-time AI: Software architecture issues. In *Proc. IEEE 1986 National Aerospace and Electronics Conf.*, Washington, D.C.: IEEE Computer Society, 67-77.

[68] Masui, S., McDermott, J., and Sobel, A. (1983) Decision-making in time critical situations. In *Proc. Eighth International Joint Conf. on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann, 233-235.

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

[70] Minton, S. (1985) Selectively Generalizing Plans for Problem Solving In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA: Morgan Kaufmann.

[71] Minton, S. (1988) Quantitative Results Concerning the Utility of Explanation-Based Learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, MN: Morgan Kaufmann, 49-54.

[72] Minton, S. (1988) *Learning effective search control knowledge: An explanation-based approach.* Amsterdam: Kluwer Academic Publishers.

[73] Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, **1**, 47-80.

[74] Natarajan, B. K., and Tadepalli. P. (1988) Two new frameworks for learning. In *Proceedings of the Fifth International Machine Learning Conference*, Ann Arbor, MI: Morgan Kaufman, 402-415.

[75] Nilsson, N. J. (1980) *Principles of Artificial Inteligence.* Palo Alto, CA: Tioga.

[76] O'Reilly, C. A., and Cromarty, A. S. (1985) "Fast" is not "real-time" in designing effective real-time AI systems. In *Applications of Artificial Intelligence II 548*, Bellingham, WA: International Society of Optical Engineering, 249-257.

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

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

[79] Pardee, W. J., and Hayes-Roth, B. (1987) Intelligent real-time control of material processing. Rockwell International Science Center Technical Report 1, Palo Alto, CA.

[80] Pearl, J. (1984) *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Reading, MA: Addison-Wesley.

[81] Raulefs, P., D'Ambrosio, B., Fehling, M., Forrest, S., and Wilber, M. (1987) Real-time process management for materials composition. In *Proc. Third Conf. on Artificial Intelligence Applications*, Washington, D.C.: IEEE Computer Society, 120-125.

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

[83] Rosenbloom, P. S. (1983) *The Chinking of Goal Hierarchies: A Model of Practice and Stimulus-Response Compatibility.* Doctoral dissertation, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA.

[84] Rosenschein, S. J., and Kaelbling, L. P. (1986) The synthesis of digital machines with provable epistemic properties. *Theoretical Aspects of Reasoning about Knowledge* (J. Y. Halpern, ed.), Los Altos: Morgan Kaufmann, 83-98.

[85] Russell, S. J. (1985) *The Compleat Guide to MRS* Technical Report no. STAN-CS-85-1080, Computer Science Department, Stanford University, Stanford, CA.

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

[87] Sachs, P. A., Paterson, A. M., and Turner, M. H. M. (1986) Escort–An expert system for complex operations in real time. *Expert Systems* **3** (1), 22-29.

[88] Savage, L. J. (1972) *The Foundations of Statistics*, 2nd rev. ed. New York: Dover.

[89] Shaw, R. (1987) RESCU–On-line real-time artificial intelligence. *Computer-Aided Engineering Journal* **7** (3), 29-30.

[90] Simon, H. A. (1982) *Models of Bounded Rationality, Volume 2: Behavioral Economics and Business Organization.* Cambridge: MIT Press.

[91] Smith. B. (1982) Reflection and semantics in a procedural language. Laboratory for Computer Science Technical Report 272, MIT.

[92] Smith, D. E. (1985) Controlling inference. Ph.D. Dissertation, Department of Computer Science, Stanford University.

[93] Sorrells, M. E. (1985) A time-constrained inference strategy for real-time expert systems. In *Proc. IEEE 1985 National Aerospace and Electronics Conf.*, Washington, D.C.: IEEE Computer Society, 1336-1341.

[94] Stankovic, J. A., Ramamritham, K., and Cheng, S. (1985) Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. In *IEEE Transactions on Computers* **C-34** (12), 1130-1143.

[95] Stockman, G. C. (1979) A minimax algorithm better than alpha-beta? *Artificial Intelligence*, **12**.

[96] Treitel, R. J. (1986) Sequentialising Logic Programs Ph.D. Dissertation, Department of Computer Science, Stanford University.

[97] Valiant, L. G. (1984) A theory of the learnable. *Comm. A.C.M.* **18** (11), 1134-1142.

[98] Vidal, J. J. (1985) AI (Artificial Intelligence) based real-time support for high performance aircraft operations. NASA Technical Report NASA-CR-176906 (NAS 1.26:176906), NASA.

[99] von Neumann, J., and Morgenstern, O. (1947) *Theory of Games and Economic Behavior*. Princeton: Princeton University Press.

[100] Wefald, E. H. (1988) The Expected Value of Search: A Decision-Theoretic Framework for Game-playing Algorithms. MS Report, Computer Science Division, U.C. Berkeley.

[101] Wright, M., Green, M., Fiegl, G., and Cross, P. (1986) An expert system for real-time control. *IEEE Software* **March**, 16-24.