Provably bounded optimal agents

Stuart J. Russell
University of California, Berkeley
CA 94720, USA
russell@cs.berkeley.edu

Devika Subramanian
Cornell University
Ithaca, NY 14853, USA
devika@cs.cornell.edu

Ronald Parr
University of California, Berkeley
CA 94720, USA
parr@guard.berkeley.edu

Abstract

A program is *bounded optimal* for a given computational device for a given environment, if the expected utility of the program running on the device in the environment is at least as high as that of all other programs for the device. Bounded optimality differs from the decision-theoretic notion of rationality in that it explicitly allows for the finite computational resources of real agents. It is thus a central issue in the foundations of artificial intelligence. In this paper we consider a restricted class of agent architectures, in which a program consists of a sequence of decision procedures generated by a learning program or given *a priori*. For this class of agents, we give an efficient construction algorithm that generates a bounded optimal program for any episodic environment, given a set of training examples. The algorithm includes solutions to a new class of optimization problems, namely scheduling computational processes for real-time environments. This class appears to contain significant practical applications.

1 Introduction

Since before the beginning of artificial intelligence, philosophers and economists have looked for a satisfactory definition of rational behaviour. This is needed to underpin theories of ethics, inductive learning, reasoning, decision-making and economic modelling. Doyle [7] has proposed that AI itself be defined as the computational study of rational behaviour. The decision-theoretic definition of rational behaviour as maximization of expected utility, and analogous definitions in the context of logical planning, have been extremely valuable in allowing AI research to be done at an abstract level, independent of specific implementations. Several systems can already be said to satisfy the required input-output relations.

Unfortunately, these approaches are of limited value because they ignore the total impracticality of reaching optimal decisions in realistic situations. In this paper, we propose instead the concept of *bounded optimality* in which the utility of a decision is a function of both its quality and the time taken to choose it. We give a formal definition below; informally, we say that an agent exhibits bounded optimality if its program is a solution to the constrained optimization problem presented by its architecture.

We begin in section 2 with a necessarily brief discussion of the relationship between bounded optimality and earlier notions of rationality. We note in particular that some important distinctions can be missed without precise definitions of terms. Thus in section 3 we introduce formal definitions of agents[1], their programs, their behaviour and their rationality. Section 4 examines a class of agent architectures for which the problem of generating bounded optimal configurations is efficiently soluble. The solution involves a new class of interesting and practically-relevant optimization problems. We examine several types of episodic, real-time environments and derive bounded optimal programs for these regimes. Finally, we describe a set of open theoretical and experimental issues, including an asymptotic version of bounded optimality that may be more robust and tractable than the strict version.

2 Historical perspective

The classical idea of perfect rationality, which developed from Mill's utilitarianism, was put on a formal footing in von Neumann's *decision theory* [25]. It stipulates that a rational agent always act so as to maximize its *expected* utility. The expectation is taken according to the agent's own beliefs; thus, perfect rationality does not require omniscience.

In artificial intelligence, the logical definition of rationality, known in philosophy as the "practical syllogism", was put forward by McCarthy [14], and reiterated strongly by Newell [15]. Under this definition, an agent should take any action that it believes is guaranteed to achieve any of its goals. If AI can be said to have had a theoretical foundation, then this definition of rationality has provided it.[2]

[1] We use the term *ralph* (rational agent with limited performance hardware) to denote an agent that exhibits bounded optimality.

[2] This is not to say that this was the wrong approach at the time. McCarthy believed, probably correctly, in the halcyon

Economists have used rationality as an abstract model of economic entities, for the purposes of economic forecasting and designing market mechanisms. Unfortunately, as Simon [23] pointed out, real economic entities have limited time and limited powers of deliberation. He proposed the study of *bounded rationality*, investigating "... the shape of a system in which effectiveness in computation is one of the most important weapons of survival." Simon's work focussed mainly on *satisficing* designs, which deliberate until reaching some solution satisfying a preset "aspiration level." The results have descriptive value for modelling various actual entities and policies, but no prescriptive framework for bounded rationality was developed.

I. J. Good [10] emphasized the conceptual distinction between classical or "type I" rationality, and what he called "type II" rationality, or the maximization of expected utility *taking into account deliberation costs*.[3] What this means is that an agent exhibits type II rationality if at the end of its deliberation and subsequent action, its subjective utility is maximized compared to all possible deliberate/act pairs in which it could have engaged. Good does not define the space of possible deliberations, but from his informal descriptions, it is clear that Type II rationality is intended to prescribe optimal sequences of computational steps. Unfortunately, these may be even harder to select than actions themselves.

Recognizing these problems, Cherniak [4] suggested a definition of "minimal rationality", specifying lower bounds on the reasoning powers of any rational agent, instead of upper bounds. A philosophical proposal generally consistent with the notion of bounded optimality can be found in Dennett's "Moral First Aid Manual" [6].

Many researchers in AI, some of whose work is discussed below, have worked on the problem of designing agents with limited computational resources; the 1989 AAAI Symposium on AI and Limited Rationality [9] contains an interesting variety of work on the topic.

Metareasoning — reasoning about reasoning — is an important technique in this area, since it enables an agent to control its deliberations according to their costs and benefits. Combined with the idea of *anytime* [5] or *flexible* algorithms [11], that return better results as time goes by, a simple form of metareasoning allows an agent to behave well in a real-time environment. Breese and Fehling [2] apply similar ideas to controlling multiple decision procedures. Russell and Wefald [18] give a general method for precompiling certain aspects of metareasoning so that a system can efficiently estimate the effects of individual computations on its intentions, giving fine-grained control of reasoning. These techniques can all be seen as approximating Type II rationality; they provide useful insights into the general problem of control of reasoning, but there is no reason to suppose that the approximations used are optimal in any sense.

Horvitz [11] uses the term *bounded optimality* to refer to "the optimization of computational utility given a set of assumptions about expected problems and constraints in reasoning resources." Russell and Wefald [19] say that an agent exhibits bounded optimality "if its program is a solution to the constrained optimization problem presented by its architecture." The philosophical "move", from optimizing over actions or deliberation sequences to *optimization over programs*, is the key to our proposal.[4] In reality, designers of intelligent agents do not have *direct* control over the agent's actions or deliberations; these are generated by the operation of the agent's program. Prescriptive specification of actions (Type I) or deliberations (Type II) may produce impossible constraints, if these specifications are not realized by any program for the agent. Precise definition of the space of agent programs is therefore an important part of the study of bounded optimality.[5] In the next section, we build a suitable set of definitions from the ground up, so that we can begin to demonstrate examples of provably bounded optimal agents.

3 Agents, architectures and programs

Intuitively, an *agent* is just a physical entity that we wish to view in terms of its *perceptions* and *actions*. What counts in the first instance is what it does, not necessarily what it thinks, or even whether it thinks at all. This initial refusal to consider further constraints on the internal workings of the agent (such as that it should reason logically, for example) helps in three ways: first, it allows us to view such 'cognitive faculties' as planning and reasoning as occurring *in the service of* finding the right thing to do; second, it makes room for those among us [1, 3] who take the position that systems can do the right thing without such cognitive faculties; third, it allows more freedom to consider various specifications, boundaries and interconnections of subsystems.

An agent can be described abstractly as a mapping (the *agent function*) from percept sequences to actions; this mapping is implemented by an *agent program*. The design and evaluation of agents are based on the behaviour of the agent program in an environment.

Let $\mathbf{P}$ be the set of percepts that the agent can receive at any instant, and $\mathbf{A}$ be the set of possible actions the agent can carry out in the external world. Then we have

Definition 1 *Agent function: a mapping*

$$f : \mathbf{P}^* \rightarrow \mathbf{A}$$

where $\mathbf{P}^*$ is the set of all possible percept sequences. The agent function is an entirely abstract entity, unlike the agent program. Computability theory relates these abstract functions to their finite representations as programs running on a machine.

days before formal intractability results in computation were known, that in the early stages of the field it was important to concentrate on "epistemological adequacy" before "heuristic adequacy".

[3] Simon [22] also says: "The global optimization problem is to find the least-cost or best-return decision, *net* of computational costs."

[4] This move is analogous to the development of 'rule utilitarianism' from 'act utilitarianism'.

[5] Recent work by Etzioni [8] and Russell and Zilberstein [20] can be seen as optimizing over a well-defined set of agent designs.

We will consider a physical agent as consisting of an architecture and a program. The architecture is responsible for interfacing between the program and the discrete, deterministic environment, and for running the program itself. With each architecture M, we associate a finite programming language $\mathcal{L}_M$, which is just the set of all programs runnable by the architecture. An *agent program* is a program $l \in \mathcal{L}_M$. When a sequence of percepts is provided by the architecture to the program, a sequence of actions is generated: the i^{th} action occurs between percept i and percept $i+1$. Thus an architecture maps a percept sequence of length k, for any k, to an action sequence of the same length according to the program it is running. The *behaviour* of an agent is the sequence of actions it generates.

Definition 2 *Architecture: a fixed interpreter M for the agent's program:*

$$M : \mathcal{L}_M \times \mathbf{P}^k \rightarrow \mathbf{A}^k$$

For the purposes of this paper, we will define the environment as a set of world states together with mappings defining the effects of actions and the generation of percepts:

Definition 3 *Environment: a set of world states* $\mathbf{W}$ *and mappings*

$$T_A : \mathbf{A} \times \mathbf{W} \rightarrow \mathbf{W}$$

$$T_P : \mathbf{W} \rightarrow \mathbf{P}$$

It is important to note that although every program induces a mapping in the above sense, the action following a given percept is not necessarily the agent's "response" to that percept; because of the delay incurred by deliberation, it may only reflect percepts occurring much earlier in the sequence, and it may not be possible to associate each action with a particular prior percept sequence.

3.1 Bounds on rationality

The expected utility of an action A_i that has possible outcomes W_{ij}, for an agent with prior evidence E about the environment, is given by

$$V(A_i) = \sum_j U(W_{ij}) P(W_{ij} \mid E, A_i)$$

where U is a real-valued utility function on states.

Definition 4 *Perfect rationality: an agent is perfectly rational iff it selects action $A = argmax_i V(A_i)$, so as to maximize its expected utility.*

This definition is a persuasive specification of the agent function f and underlies several current projects in intelligent agent design. A direct implementation of this specification, which ignores the delay incurred by deliberation, does not yield a reasonable solution to our problem – the calculation of expected utilities takes time for any real agent.

By neglecting the fact of limited resources for computation, classical decision theory fails to provide an adequate *theoretical* basis for artificial intelligence. The 'finitary predicament' [4] arises because real agents have only finite computational power and because they don't have all the time in the world. In terms of our simple formal description of agents introduced above, it is easy to see where the difficulty has arisen. In designing the agent program, logicists and decision theorists have concentrated on specifying an optimal agent function f_{opt} in order to guarantee the selection of the best possible action A in each situation. The function f_{opt} is independent of the architecture M. Unfortunately, the *behaviour* of any program that implements this function may not be desirable. The delay in computing $f_{\text{opt}}(P_1, \ldots, P_k)$ means that the k^{th} action may be optimal only with respect to some much earlier subsequence, and is now totally inappropriate.

3.2 Bounded optimality

To escape this quandary, we propose a machine-dependent standard of rationality, in which optimality constraints are imposed on *programs* rather than *agent functions*, *deliberations* or *behaviours*. To formalize this idea, it will be helpful to assume a real-valued utility function U on *histories*, that is, sequences of world states. Then we assign values to a program l based on the sequence of states through which it "drives" the environment E when run on M starting in the world W_0:

$$U : \mathbf{W}^* \rightarrow \mathcal{R}$$

$$V(l, M, E, W_0) = U(result(l, M, E, W_0))$$

where *result* denotes the state sequence generated by the execution of the program l, defined in the obvious way using T_A and T_P.

If the initial state W_0 and environment E are known, then the optimal agent program is given by:

Definition 5 *Optimal agent program:*

$$l_{\text{opt}} = argmax_{l \in \mathcal{L}_M} V(l, M, E, W_0)$$

If instead the "designer" has only a probability distribution over the initial state and environment model, then this will introduce a distribution over the state sequences generated, and the optimal program has the highest expected value $V(l, M, E)$ over this distribution.

4 Provably ralphs

In order to construct a provably rational agent with limited performance hardware (ralph), we must carry out the following steps:

- Specify a class of machines on which programs are to be run.
- Specify the properties of the environment in which actions will be taken, and the utility function on the behaviours.
- Propose a construction method.
- Prove that the construction method succeeds in building ralphs.

4.1 Production system architectures

We begin our study with a simple form of *production system*, in which condition-action rules of the form *"If ϕ_i then do A_i"* are applicable whenever their left-hand sides become true. Even such a simple system can easily overtax the resources of a real-time agent, if the rule base becomes large and the conditions become complex. An agent implemented as a production system will therefore contain *approximate* rules, for which the action A_i is not guaranteed to be the best possible whenever condition ϕ_i obtains in the world.

An illustrative example is provided by the image processing algorithms in an automated mail sorter. These machines scan handwritten or printed addresses on mail pieces and dispatch them accordingly. The scanned image is processed by any or all of several procedures designed to read with varying degrees of accuracy and resource expenditure; each procedure may have many possible variants — for example, we can vary the number of hidden nodes in a neural network recognizer. To prevent jams, the mail piece must be sorted appropriately, or rejected, in time for the (stochastic) arrival of the next piece. The object is to maximize the accuracy of sorting while minimizing the reject percentage and avoiding jams.

Here we consider only a special case of production system, in which *complete* rules (or *decision rules*) are matched in a *fixed* sequence. We call a sequence of production rules a *strategy*. Each rule i has associated with it a match time t_i and a quality q_i, which corresponds to the utility of the rule's recommended action if taken at the beginning of the episode.

We assume that each rule is drawn from some rule language $\mathcal{J}$. (In keeping with the scheduling literature, we will often use the term "job" to refer to a rule in a sequence.) Let $M_{\mathcal{J},n}$ denote a production system architecture that can accommodate rules of maximum size n from rule language $\mathcal{J}$. For example, we might consider feedforward neural networks with at most n nodes.

In executing a typical sequence $R = \langle r_1 \ldots r_m \rangle$, the agent matches each rule in turn against the current percept, generating a recommended action. At some time t after the beginning of the episode, it will decide to act, selecting the highest-quality action recommended by any of the rules it has previously matched. If the quality of the rule chosen is q_i, the value of the episode for the agent will be some function of q_i and t, according to the regimes described below. Before the episode begins, the agent may not know when it will act (for example, in the stochastic deadline case), so generally speaking the value of a given strategy will be an expectation.

4.2 Episodic real-time environments

First, we place some restrictions on the environment in order to simplify our problem. We need C to learn utility information, so in order to avoid (temporarily) the *credit assignment* problem we assume an "episodic" environment, where after each non-null action by the agent, a reward is received and the environment then reaches an unknown state drawn from a probability distribution that remains stationary over time. The initial state W_0 is also drawn from this distribution. In the mail-sorting example, the reward is 1 for a correct dispatch; 0 for an incorrect dispatch; 0.2, say, for a reject; and $-\infty$ for a jam.

In order to talk about deadlines and time cost, it is useful to have a notion of inaction. Let $noop^t$ denote a sequence of null actions lasting t time steps. Also, $[A \mid W]$ will denote the sequence of world states through which the action sequence A drives the environment given an initial state W. We use '$\cdot$' to denote concatenation of sequences.

Now we can define three typical real-time regimes: fixed time cost, fixed deadline and stochastic deadline.

Definition 6 *Fixed time cost: for any action A, any state W and any state sequences W_i, W_j, $U(W_i \cdot W \cdot [noop^t \cdot A \mid W] \cdot W_j) = U(W_i \cdot W \cdot [A \mid W] \cdot W_j) - ct$. for some constant c.*

Definition 7 *Fixed deadline: for any action A, any state W, state sequences W_i, W_j, for some constant T_d,*

$$U(W_i \cdot W \cdot [noop^t \cdot A \mid W] \cdot W_j) = \begin{cases} U(W_i \cdot W \cdot [A \mid W] \cdot W_j) & \text{if } t \leq T_d \\ -\infty & \text{otherwise} \end{cases}$$

Deadlines are thus represented by a utility "cliff" occurring at some time T_d after the beginning of an episode. In the case of a stochastic deadline, which describes the mail sorter, the location of the utility cliff is not known exactly. We assume that, for the designer, the value of $[noop^t \mid W]$ is a random variable, distributed in such a way that an action following it has probability $P(t)$ of being of zero utility for the episode. This corresponds to P being the cumulative distribution function for the deadline arrival time.[6]

4.3 The construction algorithm

Our general scheme at present is to exhibit an algorithm that is capable of learning sets of individual decision procedures, and arranging them in a sequence such that the performance of an agent using the sequence is at least as good as that of any other such agent. That is, the construction algorithm C, operating in E, returns a program l for architecture M such that $V(l, M, E) \approx V(l_{\text{opt}}, M, E)$. C will work by first observing a set $\mathbf{e}$ of training episodes in E, and then building an approximately optimal strategy using a learning algorithm $L_{\mathcal{J},k}$, for learning rules of size k in the rule language $\mathcal{J}$.

procedure $C(n, \mathbf{e}, P)$

1. $\mathbf{R} \leftarrow \{L_{\mathcal{J},k}(\mathbf{e}) : 1 \leq k \leq n\}$
2. $S \leftarrow optimize(\mathbf{R}, P)$

In addition to constructing the decision rules, $L_{\mathcal{J},k}$ outputs estimates of the rule quality q_i. For now, we will assume that the rule match times t_i and the deadline distribution P (for the stochastic case) are known. Algorithm *optimize* extracts an optimal strategy from

[6] We assume that some percept immediately precedes the actual deadline, allowing the agent to respond at the next step. Without this, the agent is walking blindfolded towards the utility cliff (also an interesting problem).

the rule set **R**. In the following subsections we describe the three variants of *optimize*.

4.4 Results for fixed time cost

This case is straightforward. The value of a strategy is given by

$$V(s) = max_{i \in s} q_i - c \sum_{j \in s} t_j \quad (1)$$

Theorem 1 *The optimal strategy for fixed time cost utility functions is the singleton* $argmax_{i \in \mathbf{R}}[q_i - ct_i]$.

4.5 Results for fixed deadline

This case is also straightforward. The value of a strategy s is given by:

$$V(s) = \begin{cases} max_{i \in s} q_s & \text{if } \sum_{i \in s} t_i \leq T_d \\ -\infty & \text{otherwise} \end{cases} \quad (2)$$

Theorem 2 *The optimal strategy is the singleton sequence i, where* q_i *is the highest among all rules for which* $t_i \leq T_d$.

4.6 Results for stochastic deadlines

With a stochastic deadline distributed according to $P(t)$, the value of a strategy $s = s_1, \ldots, s_m$ is an expectation. It is calculated as a summation, over the jobs that can be interrupted, of the probability of interruption times the quality of the best completed job:

$$V(s) = \sum_{i=1}^{m} [P(\sum_{j=1}^{i+1} t_{s_j}) - P(\sum_{j=1}^{i} t_{s_j})] M_i \quad (3)$$

where $M_i = max(q_{s_1}, \ldots, q_{s_i})$, and for convenience we define $t_{s_{m+1}} = \infty$. We will use this formula to prove a number of properties of optimal strategies.

A simple example serves to illustrate the value function. Consider $R = \{r_1, r_2, r_3\}$. The rule r_1 has a quality of 0.2 and needs 2 seconds to run: we will represent this by $r_1 = (0.2, 2)$. The other rules are $r_2 = (0.5, 5), r_3 = (0.7, 7)$. The deadline density function p ($= dP/dt$) is a uniform distribution over 0 to 10 seconds. The value of the sequence $r_1 r_2 r_3$ is

$$V(r_1 r_2 r_3) = [.7 - .2].2 + [1 - .7].5 + [1 - 1].7 = .25$$

A geometric intuition is given by the notion of a *performance profile* shown in Figure 1. For a uniform deadline density function, the value of a sequence is proportional to the area under the performance profile up to the last possible interrupt time. Note that the height of the profile during the interval of length t_i while job i is running is the quality of the best of the *previous* jobs.

The following lemma gives an extremely useful property of optimal sequences:

Lemma 1 *There exists an optimal sequence that is sorted in increasing order of* q_i.

Henceforth we need consider only q-ordered strategies. This means that M_i in equation (3) can be replaced by q_i.

The following lemma establishes that a strategy can always be improved by the addition of a better job at the end:

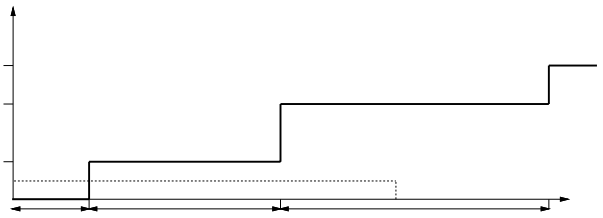

Figure 1: Performance profile for $r_1 r_2 r_3$, with p superimposed.

Lemma 2 *For every sequence* $s = s_1, \ldots, s_m$ *sorted in increasing order of quality, and single step* z *with* $q_z \geq q_{s_m}$, $V(sz) \geq V(s)$.

Corollary 1 *There exists an optimal strategy ending with the highest-quality job in* **R**.

Proofs of all the above results appear in [17].

4.6.1 A Dynamic Programming Algorithm

The *dynamic programming* method can be used to obtain an optimal sequence of decision rules in pseudo-polynomial time. We assume that the time t_i associated with each rule is an integer (this is almost without loss of generality, using standard rounding and scaling methods). In keeping with the lemma above, we assume that optimal strategies are ordered by q_i.

The algorithm constructs the table $S(i, T)$, where each entry in the table is the highest value of any sequence that ends with rule i at time T. We assume the rule indices are arranged in increasing order of quality, and T ranges from the start time 0 to the end time $L = \sum_{i \in \mathbf{R}} t_i$. The update rule is:

$$S(i, T) = max_{k \in [0 \ldots i-1]} [S(k, T - t_i) + (q_i - q_k)[1 - P(T)]]$$

¿From Corollary 1, we can read off the best sequence from the highest value in row n of the matrix M.

Theorem 3 *The DP algorithm computes the optimal sequence in time* $O(n^2 L)$ *where* n *is the number of decision procedures in* **R**.

Uniform distributions If the deadline is uniformly distributed over some initial interval, we can obtain an optimal sequence in strongly polynomial time. Initially, we assume an interval longer than any possible sequence. Then the probability that the deadline arrives during job s_i of sequence s is just $P(t_{s_i})$. Hence we have a simple recursive specification of the value $V(as)$ of a sequence beginning with job a:

$$V(as) = q_a P(t_{s_1}) - q_{s_m} P(t_a) + V(s). \quad (4)$$

A dynamic programming algorithm can then use the state function $S_f(i)$, which is the highest value of any rule sequence ending in f and beginning with rule i. From lemma 1 and equation 4, the update rule for $S_f(i)$ is

$$S_f(i) = \max_{j > i} [q_i P(t_j) - q_f P(t_i) + S_f(j)]$$

with boundary condition $S_f(f) = q_f(1 - P(t_f))$. For any given f, $S_f(i)$ can therefore be tabulated for $i = 1 \ldots n$

in time $O(n^2)$, from which we can obtain the optimal sequence.

If all the rules can fit before the end of the deadline distribution, then from Corollary 1 above the last rule f must be the rule with highest quality. Otherwise, any rule might be the last rule with a non-zero chance of completing, so we check each candidate:

Theorem 4 *The optimal sequence of decision procedures for a uniformly distributed stochastic deadline can be determined in $O(n^3)$ time where n is the number of decision procedures in R.*

4.7 Agnostic Learning of Decision Rules

Our learning algorithm $L_{\mathcal{J},k}$ searches for the best rule in $\mathcal{J}, k$. In order to work in any environment, it must be *agnostic* [13] in that it makes no assumptions about the target function, that is, the form of the *correct* decision rule. It searches for the best subject to a complexity parameter k that bounds the size of the rules. Kearns, Schapire and Sellie have shown (Theorems 4 and 5 in [13]) that, for some languages $\mathcal{J}$, the error in the learned approximation can be bounded to within ϵ of the best rule in $\mathcal{J}, k$ that fits the examples, with probability $1-\delta$. The sample size needed to guarantee these bounds is polynomial in the complexity parameter k, as well as $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Once an approximate rule with a certain complexity $k \in \{1, \ldots n\}$ is learned, we statistically estimate its quality q. Standard Chernoff-Hoeffding bounds can be used to limit the error in the quality estimate to be within ϵ_q with probability $1-\delta_q$. The sample size needed is also polynomial in $\frac{1}{\epsilon_q}$ and $\frac{1}{\delta_q}$.

Thus the error in the agnostically learned rules is bounded to within ϵ of the best rule in its complexity class with probability $1-\delta$. The error in the quality estimation of these rules is bounded by ϵ_q with probability $1-\delta_q$. We can show that the policy selection methods for the three real-time regimes will incur a deficit of at most $\epsilon + 2\epsilon_q$ in the choice of the optimal sequence of rules.

4.8 The bottom line

We can now state the bounded optimality theorem for our construction procedure.

Theorem 5 *Assume a production system architecture $M_{\mathcal{J},n}$, operating in any episodic environment. For each of the three real-time regimes defined above, with probability greater than $1-\delta-\delta_q$ the construction procedure C returns a program l such that $V(l_{\text{opt}}) - V(l) < \epsilon + 2\epsilon_q$ after seeing a number of episodes that is polynomial in $n, \frac{1}{\epsilon}, \frac{1}{\epsilon_q}, \frac{1}{\delta}$ and $\frac{1}{\delta_q}$. Furthermore, the computation time of C is (at worst) polynomial in the above quantities and in L, the sum of the rule execution times.*

Although the theorem has practical applications, it is mainly intended as an illustration of the derivation of a bounded optimal agent. With some additional work, more general error bounds can be derived for the case in which the rule execution times t_i and the real-time utility variation (time cost, fixed deadline or deadline distribution) are all estimated from the training episodes.

5 Further work

We plan to extend this work in several directions, as follows.

1. Foundational issues:

Learning agents: When the agent, whose design is to be optimized, includes a learning component, the notion of bounded optimality becomes even more interesting because we must take into account how the agent's configuration will evolve over time, reflecting its own expected obsolescence.

Asymptotic bounded optimality: The strict notion of bounded optimality may be a useful philosophical landmark from which to explore artificial intelligence, but it may be too strong to allow many interesting, general results to be obtained. Just as in complexity theory, where absolute efficiency is the aim but asymptotic efficiency is the game (so to speak), in studying bounded optimality an asymptotic version might help. First, we need a *class* of environments, $\mathbf{E}$, which is unbounded in a complexity measure on environments $n(E)$. Then we will say that an agent program l is timewise asymptotically bounded optimal iff

$$\exists k, n_0 \; \forall E \in \mathbf{E} \;\; n(E) > n_0 \Rightarrow V(l, E, kM) \geq V(l_{\text{opt}}, E, M)$$

where kM denotes a version of the machine M speeded up by a factor k. In English, this means that the program is basically along the right lines if it just needs a faster machine to be as good as the best program on all problems above a certain level of difficulty.

Asymptotic bounded optimality generalizes the standard definition of asymptotic complexity. Let $\mathbf{E}$ be a class of environments in which a problem is input to the machine at time $t = 0$, where $n(E)$ is the input size measure. Let V return $1/t$ (or any decreasing function of t) for a program that outputs the correct solution at time t, and 0 for all other programs. Then a program is bounded optimal iff it has asymptotic complexity equal to the tight lower bound for the class $\mathbf{E}$. Note that in standard complexity, we allow any constant factor c in the *execution time* of the program, whereas our definition uses a constant factor in the speed of the machine. In the standard setting these are equivalent, but with more general time-dependent utilities only the latter is appropriate.

2. Scheduling issues:

The existence of polynomial-time algorithms and approximation schemes for variants of the computation scheduling problem; scheduling algorithms for situations in which the solution qualities of individual processes are interdependent (such as when one can use the results of another); scheduling combinations of computational and physical (e.g., job-shop and flow-shop) processes, where objective functions are a combination of summation and maximization; and computation scheduling for parallel machines or multiple agents.

3. Learning issues:

Relaxing the stationarity requirement on the environment — this entails generalizing the PAC model to handle the case in which the fact that the agent learns may have some effect on the distribution of future

episodes; relaxing the episodic requirement to allow non-immediate rewards — this entails addressing the credit assignment problem; examining variants of the agnostic learning model to find the most practically useful theoretical scenario.

4. Applications:

Despite the deliberate simplicity of the architecture, our construction algorithm can be applied directly to the problems such as scheduling image-processing algorithms for a mail sorter. Scheduling of mixed computational and physical processes, mentioned above, broadens the scope of applications considerably. An industrial process, such as designing and manufacturing a car, consists of both computational steps (design, logistics, factory scheduling, inspection etc.) and physical processes (stamping, assembling, painting etc.). One can easily imagine many other applications in real-time financial, industrial and military contexts.

6 Conclusions

In short, we are proposing a new line of inquiry into bounded optimal agents in which the value of a decision is judged in terms of the effect it has on the actions performed by the agent, noting that both actions and computations have time value. Bounded optimality may provide a suitable basis for theoretical research in artificial intelligence. Asymptotic bounded optimality in particular promises to yield useful results on composite agent designs, using the optimality-preserving composition methods in [20]. As a robust measure of rationality, it's possible that it could do for AI what "big-O" descriptions did for complexity theory.

Bounded optimality also philosophically interesting implications. For example, like the rule-utilitarians we no longer talk about rational *actions*, because individual actions, and even deliberations, by a bounded optimal agent may be arbitrarily irrational in the classical sense.

Furthermore, this theoretical research should, by design, apply to the *practice* of artificial intelligence in a way that idealized, infinite-resource models may not. We have given, by way of illustrating this definition, a bounded optimal agent: the design of a simple condition-action rule system that, given a learning mechanism, provably and efficiently converges to a rational configuration.

References

[1] Agre, P., and Chapman, D. (1987) Pengi: An implementation of a theory of activity. In *Proc. 6th National Conference on Artificial Intelligence*, Seattle, WA: Morgan Kaufmann.

[2] Breese, J. S., and Fehling, M. R. (1990) Control of problem-solving: Principles and architecture. In Shachter, R. D., Levitt, T., Kanal, L., and Lemmer, J. (Eds.) *Uncertainty in Artificial Intelligence 4*. Amsterdam: North Holland.

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

[4] Cherniak, C. (1986) *Minimal rationality*. Cambridge: MIT Press.

[5] Dean, T. and Boddy, M. (1988) An analysis of time-dependent planning. In AAAI-88, 49-54.

[6] Dennett, D. (1986) *The moral first aid manual*. Tanner lectures on human values, University of Michigan.

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

[8] O. Etzioni. Tractable decision-analytic control. Proc. of 1st International Conference on Knowledge Representation and Reasoning, 114-125, 1989.

[9] Fehling, M., and Russell, S. J. (Eds.) (1989) *Proceedings of the AAAI Spring Symposium on Limited Rationality*. Stanford, CA.

[10] Good, I. J. (1971) Twenty-seven principles of rationality. In Godambe, V. P., and Sprott, D. A. (Eds.) *Foundations of Statistical Inference*. Toronto: Holt, Rinehart, Winston.

[11] Horvitz, E. J. (1988) Reasoning about beliefs and actions under computational resource constraints. In Levitt, T., Lemmer, J., and Kanal, L. (Eds.) *Uncertainty in Artificial Intelligence 3*. Amsterdam: North Holland.

[12] Horvitz, E., Cooper, G., and Heckerman, D. (1989) Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI: Morgan Kaufmann.

[13] Kearns, M., Schapire, R., and Sellie, L. (1992) Toward efficient agnostic learning. In *Proc. 5th Ann. Workshop on Computational Learning Theory*. Pittsburgh, PA: Morgan Kaufmann.

[14] McCarthy, J. (1958) Programs with common sense. In *Proceedings of the Symposium on the Mechanization of Thought Processes*, Teddington, England: HMSO.

[15] Newell, A. (1981) The knowledge level. *AI Magazine* **2**, 1-20.

[16] Nilsson, N. J. (1991) Logic and artificial intelligence. *Artificial Intelligence* **45**, 31-56.

[17] Russell, S. J., and Subramanian, D. (1992) On provably rational agents with limited performance hardware. In Baum, E. (Ed.) *Computational learning and cognition: Proceedings of the Third NEC Symposium*, SIAM Press.

[18] Russell, S. J., and Wefald, E. H. (1989) Principles of metareasoning. In *Proc. KR-89*.

[19] Russell, S. J., and Wefald, E. H. (1991) *Do the right thing: Studies in limited rationality*. Cambridge, MA: MIT Press.

[20] Russell, S. J., and Zilberstein, S (1991) Composing real-time systems. In *Proc. IJCAI-91*, Sydney.

[21] Savage, L. J. (1972) *The foundations of statistics*, 2nd rev. ed. New York: Dover.

[22] Simon, H. A. (1976) On how to decide what to do. In [23].

[23] Simon, H. A. (1982) *Models of bounded rationality, Volume 2*. Cambridge: MIT Press.

[24] Valiant, L. G. (1984) A theory of the learnable. *Communications of the ACM* **18**(11), 1134-42.

[25] von Neumann, J., and Morgenstern, O. (1947) *Theory of games and economic behavior*. Princeton: Princeton University Press.