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

Abstract

Since its inception, artificial intelligence has relied upon a theoretical foundation centred around perfectly rationality as the desired property of intelligent systems. We argue, as others have done, that this foundation is inadequate because it imposes fundamentally unsatisfiable requirements. Theoretical results in such a framework are likely to be of little practical interest. We propose instead a property called *bounded optimality*. Roughly speaking, an agent is bounded optimal if its program is a solution to the constrained optimization problem presented by its architecture and the task environment. We show how to construct agents with this property for a simple class of machine architectures in a broad class of real-time environments. We provide experimental results in the context of scheduling zipcode recognition algorithms for an automated mail sorting facility. We also define a weaker property, *asymptotic bounded optimality* (ABO), and show that we can construct *universal* ABO programs. They can be used as building blocks for more complex systems.

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 [9] has proposed that AI itself be defined as the computational study of rational behaviour. The role of such definitions in AI is clear: if we define some desirable property P, then we ought to be able to design a system that provably possesses property P. Theory meets practice when our systems exhibit P in reality. Furthermore, that they exhibit P in reality should be something that we actually care about. We should also be able to use such a definition to design more complex systems without worrying about implementation details of the components. It would be nice if we could do all this for the property "behaves rationally."

AI has in fact concentrated on the property "eventually returns what would have been the rational choice at the time." There exist systems such as influence diagram evaluators that exhibit this property for a decision-theoretic definition of rational choice, and systems such as nonlinear planners that exhibit it for a logical definition of rational choice. This is assumed to be an interesting property for a system to exhibit since it constitutes an "in-principle" capacity to do the right thing. However, this is of limited value in practice, because of the total impracticality of reaching optimal decisions in non-toy problems. The actual behaviour exhibited by such systems is absurdly far from being rational; for example, a "perfectly rational" chess program may choose the right move, but may take 10^{50} times too long to do so. If theoretical physicists proposed fundamental laws giving answers that deviated from reality by a factor of 10^{50}, experimental physicists might well object, and funding agencies might well ask for their money back. Theoretical AI is subject to similar pressures.

In reality, the value of a particular action choice is a function of both its quality and the time taken to choose it. We will use the term "behaviour" to refer to the actions of a system and the times at which they

are taken. We could propose to study the property "immediately chooses the action of highest quality," true of systems that can calculate at infinite speeds, but we know that no actual systems can exhibit this property. In this paper, we propose instead to study the property "behaves as well as possible given its computational resources", the property of *bounded optimality*. We make four claims:

1. That a system exhibits bounded optimality is desirable in reality.
2. By definition, for any task environment and computing device, some program exhibits bounded optimality.
3. It is possible to construct provably bounded optimal programs.
4. Artificial intelligence can be usefully characterized as the study of bounded optimality, particularly in the context of complex task environments and reasonably powerful computing devices.

The first two claims are unlikely to be controversial. This paper supports the third claim in detail. The fourth claim may, or may not, stand the test of time.

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. We also show how the bounded optimal solutions we obtain can be applied in the context of an automated mail-sorting facility. We also define a weaker property, *asymptotic bounded optimality* (ABO), and show that we can construct *universal* ABO programs. A program is universally ABO if it is independent of the specific form of time dependence of its utility function. Universal ABO programs can be used as building blocks for more complex systems. Finally, we describe a set of open theoretical and experimental issues that may be more robust and tractable than the strict version of bounded and asymptotic bounded optimality.

2. Historical perspective

The classical idea of perfect rationality, which developed from Mill's utilitarianism, was put on a formal footing in *decision theory* [25, 29]. 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 [16], and reiterated strongly by Newell [17]. 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]

Economists have used rationality as an abstract model of economic entities, for the purposes of economic forecasting and designing market mechanisms. The assumption of perfect rationality makes it possible to prove simple theorems about the properties of markets in equilibrium. Unfortunately, as Simon [27] 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

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, that in the early stages of the field it was important to concentrate on "epistemological adequacy" before "heuristic adequacy". It is not clear if AI would have taken this tack had it not been operating in the halcyon days before formal intractability results put a damper on everyone's spirits.

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 general prescriptive framework for bounded rationality was developed. Although it proved possible to calculate optimal aspiration levels for certain problems, no structural variation was allowed in the agent design.

I. J. Good [12] distinguished 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.

Philosophy has also seen a gradual evolution in the definition of rationality. Some changes have come about because of difficulties with individual versus societal rationality, but limited reasoning capabilities have, for example, caused some to suggest that decisions should be made according to rules of behaviour that can be understood by persons of average intelligence [3]. Cherniak [6] 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" [8]. Dennett explicitly discusses the idea of reaching equilibrium within the space of feasible configurations of decision procedures. He uses as an example the PhD admissions procedure of a philosophy department.

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 [11] 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* [7] or *flexible* algorithms [13], 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 [4] apply similar ideas to controlling multiple decision procedures. Russell and Wefald [20] 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 [13] 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 [22] 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. 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.[4] 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. Simon [26] also says: "The global optimization problem is to find the least-cost or best-return decision, *net* of computational costs."
4. Recent work by Etzioni [10] and Russell and Zilberstein [23] can be seen as optimizing over a well-defined set of agent designs.

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, 5] 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.

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: which we assume to be "synchronized", in the sense that 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:*

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

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.

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*

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

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

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. $P(W_{ij} \mid E, A_i)$ is the probability that W_{ij} is the state resulting from the action A_i conditioned on knowledge E about the environment.

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. 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, no real program in $\mathcal{L}_M$ implements this function in a non-trivial environment, because optimal actions cannot usually be computed before the next percept arrives.

Suppose the environment consists of games of chess under tournament rules against some population of human grandmasters, and suppose M is some standard personal computer. Then f_{opt} describes an agent that always plays in such a way as to maximize its expected points total against the opposition, where the maximization is over the *moves* it makes. We claim that no possible program can play this way. It is quite possible, using depth-first alpha-beta search to termination, to execute the program that chooses the optimal minimax move in each situation, but the agent function induced by this program is not the same as f_{opt}. In particular, it ignores such percepts as the dropping of its flag indicating a loss on time.

To discuss Type II rationality in this situation, we need to specify what constitutes a deliberation sequence. Good proposes that a reasonable choice would be a sequence of node expansions in the game tree, followed by backing up of leaf node evaluations to show the best move. For simplicity we will assume a per-move time limit. Then a Type II rational agent will execute whichever sequence of node expansions chooses the best move, of all those that finish before the time limit. (One would imagine that in most cases the move selected will be the actual best move, even though further deliberation might cause the program to abandon it.) Again, we claim that no program in $\mathcal{L}_M$ can exhibit such behaviour, because the node expansions must result from some computation, and the problem of choosing node expansions is harder than the original problem.

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.

We calculate the value of a program $V_p(l, \mathbf{E}, M)$ over a distribution $p(E, W_0)$ defined over a set of (E, W_0) pairs. The environments E are drawn from a class $\mathbf{E}$ of environments, and W_0 ranges over possible initial states in E.

$$V_p(l, \mathbf{E}, M) = \sum_{E \in \mathbf{E}} p(E, W_0) V(l, M, E, W_0)$$

Definition 5 *Optimal agent program:*

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

We need environments in $\mathbf{E}$ to be unbounded in complexity measure $n(E)$. This can be enforced by the weak requirement that $\nexists N. \sum_{E \in \mathbf{E}, n(E) > N} p(E) = 0$. This says, that there is no N such that the probability of occurrence of an environment E with size greater than N, is zero. This forces the existence of environments of unbounded complexity, a fact we will use in a later definition of asymptotic bounded optimality in Section 5. We can enforce this by the stronger requirement that $\sum_{E \in \mathbf{E}, n(E) > N} p(E) \geq \frac{1}{f(n(E))}$ where f is a polynomial function.

Definition 6 *Bounded optimality: an agent with architecture M is bounded optimal for environment class $\mathbf{E}$ with distribution $p(E, W_0)$ if it executes the optimal agent program $l_{\text{opt}} = argmax_{l \in \mathcal{L}_M} V_p(l, \mathbf{E}, M)$.*

In the context of our chess example, suppose the computer has a total program memory of 8 megabytes. Then there are 2^{64M} possible programs that can be represented in the machine, of which a much smaller number play legal chess. Under tournament conditions, one or more of these programs will have the best expected performance. Suffice it to say that we are not yet ready to announce the identity of this class of programs, so we will begin with a more restricted problem.

4. Provably bounded optimal agents

In order to construct a provably bounded optimal agent, 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 bounded optimal agents.

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 our construction algorithm 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 7 *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 8 *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.[5]

Definition 9 *Stochastic deadline: for any action A, any state W, state sequences W_i,W_j, and for cumulative deadline distribution function P,*

$$U(W_i \cdot W \cdot [noop^t \cdot A \mid W] \cdot W_j) = \sum P(noop^t \mid W) U(W_i \cdot W \cdot [A \mid W] \cdot W_j)$$

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 $\mathbf{E}$ returns a program l for architecture M such that $V(l, \mathrm{E}, M) \approx V(l_{\mathrm{opt}}, \mathbf{E}, M)$. C will work by first observing a set E of training episodes in $\mathbf{E}$, 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 the rule set $\mathbf{R}$. In the following subsections we describe the three variants of *optimize*.

4.4 Results for fixed time cost

The optimal strategy in this case is one step long, consisting of the decision rule r_i for which $q_i - ct_i$ is highest. To prove this, we first note that the value of a strategy is given by

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

Lemma 1 *For any two strategies s_1 and s_2 under fixed time cost, either $V(s_1 s_2) \leq V(s_1)$ or $V(s_1 s_2) \leq V(s_2)$.*

Proof: From equation (1),

$$\begin{aligned} V(s_1 s_2) &= max_{i \in s_1 s_2} q_i - c \textstyle\sum_{j \in s_1 s_2} t_j \\ &= max[max_{i \in s_1} q_i, max_{i \in s_2} q_i] - c \textstyle\sum_{j \in s_1} t_j - c \sum_{j \in s_2} t_j \\ V(s_1) &= max_{i \in s_1} q_i - c \textstyle\sum_{j \in s_1} t_j \\ V(s_2) &= max_{i \in s_2} q_i - c \textstyle\sum_{j \in s_2} t_j \end{aligned}$$

Therefore, if $max_{i \in s_1} q_i > max_{i \in s_2} q_i$, then $V(s_1 s_2) = V(s_1) - c \sum_{i \in s_2} t_i$. Otherwise, $V(s_1 s_2) = V(s_2) - c \sum_{i \in s_1} t_i$. Thus either $V(s_1 s_2) \leq V(s_1)$ or $V(s_1 s_2) \leq V(s_2)$ depending on whether the element with the highest quality occurs in s_1 or s_2. The inequalities become strict if neither s_1 nor s_2 is the empty sequence.□

5. 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).

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

Proof: By Lemma 1, we know that the value of a sequence is less than at least one of its non-empty proper subsequences. Thus we can keep improving the value of a sequence by finding these subsequences — the process terminates when the length of the sequence is 1. By definition, the optimal strategy s is one that maximizes $V(s)$, and since the optimal strategy is a singleton, the theorem follows. Note that this strategy can be computed in $O(n)$, where n is number of decision procedures in $\mathbf{R}$.□

4.5 Results for fixed deadline

The value of a strategy s is given by:

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

A strategy s can be discarded if $\sum_{i \in s} t_i > T_d$, since its value is $-\infty$. Strategies that can execute within the deadline are called within-time strategies. We can establish a lemma similar to Lemma 1 for such strategies. Then we can show that the optimal strategy is a singleton strategy i with the highest q_i among all within-time decision procedures.

Lemma 2 *For any two within-time strategies* s_1 *and* s_2 *under a fixed deadline,* $V(s_1 s_2) \leq V(s_1)$ *or* $V(s_1 s_2) \leq V(s_2)$.

Proof: From equation (1),

$$\begin{aligned} V(s_1) &= max_{i \in s_1} q_i \\ V(s_2) &= max_{i \in s_2} q_i \\ V(s_1 s_2) &= \begin{cases} max[max_{i \in s_1} q_i, max_{i \in s_2} q_i] & \text{if } \sum_{i \in s_1 s_2} t_i \leq T_d \\ -\infty & \text{otherwise} \end{cases} \end{aligned}$$

Therefore, $V(s_1 s_2) \leq V(s_1)$, if $max_{i \in s_1} q_i > max_{i \in s_2} q_i$. Every sequence can be improved by finding an appropriate non-empty proper subsequence. □

Now we are ready to derive the optimal strategy.

Theorem 2 *The optimal strategy is the singleton sequence* i, *where* q_i *is the highest among all within-time decision procedures, i.e.,* $t_i \leq T_d$.

Proof: Identical to proof for Lemma 1.□

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:

$$V(s) = \sum_{i=1}^{m} [P(\textstyle\sum_{j=1}^{i+1} t_j) - P(\sum_{j=1}^{i} t_j)] M_i \tag{3}$$

where $M_i = max(q_1, \ldots, q_i)$, and for convenience we define $t_{m+1} = \infty$.

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

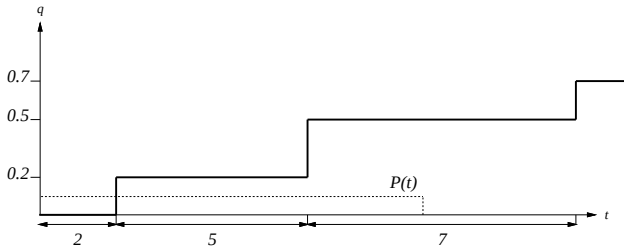

Figure 1: Performance profile for $r_1 r_2 r_3$, with p superimposed.

$r_2 = (0.5, 5), r_3 = (0.7, 7)$. 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.

Definition 10 *Performance profile: For a strategy s, the performance profile $Q_s(t)$ gives the quality of the action returned if the agent is interrupted at t:*

$$Q_s(t) = max\{q_i : \sum_{j=1}^{i} t_j \leq t\}$$

Lemma 3 *The performance profile of any strategy is monotonically nondecreasing.*

Proof: follows immediately from equation (3), since the quality of the action returned at the interrupt is the maximum of the qualities of all rules completed to date.□

We also have the obvious property that a strategy with higher quality decisions at all times is a better strategy:

Lemma 4 *If $\forall t\ Q_{s_1}(t) \geq Q_{s_2}(t)$, then $V(s_1) \geq V(s_2)$.*

Proof: follows directly from equation (3).□

Equation (3) sums the areas along vertical time slices. We could as well calculate the areas along horizontal quality slices giving us

$$V(r_1 r_2 r_3) = [.2 - 0][1 - .2] + [.5 - .2][1 - .7] + [.7 - .5][1 - 1] = .25$$

Thus the value V(s) of a strategy $s = (s_1 s_2 \ldots s_m)$ can also be written as

$$V(s) = \sum_{i=0}^{m-1} [max(q_{s_{i+1}}, M_i) - M_i] * [1 - P(\textstyle\sum_{j=1}^{i} t_{s_j})] \tag{4}$$

This formula holds for any distribution P (proof omitted). We use both formulations of the value function below.

First, we observe the equivalence between a multi-step strategy $s = s_1, \ldots, s_m$ and a singleton strategy e where $V(e) = V(s)$. This allows us to replace every compound strategy by a simple one-step strategy in the proofs below.

Lemma 5 Simple Strategy-Equivalence *The strategy $s = s_1, \ldots, s_m$ is equivalent to the single step strategy e where $V(e) = q_e[1 - P(t_e)] = V(s)$.*

Proof: Follows directly from the definitions. □

A more interesting, in-context replacement form of the strategy equivalence lemma is proved below. It involves replacing the subsequence $s_i \ldots s_j$ in the strategy $s_1 \ldots s_{i-1} s_i \ldots s_j s_{j+1} \ldots s_m$ by a single step e yielding the sequence $s_1 \ldots s_{i-1} e s_{j+1} \ldots s_m$.

Lemma 6 Strategy Equivalence Theorem: *The strategy*

$$s = s_1 \ldots s_{i-1} s_i \ldots s_j s_{j+1} \ldots s_m$$

is equivalent to the strategy

$$s' = s_1 \ldots s_{i-1} e s_{j+1} \ldots s_m$$

where

$$V(e, \sum_{k=1}^{i-1} t_k) = V(s_i \ldots s_j, \sum_{k=1}^{i-1} t_k)$$

Proof: By $V(s,t)$ we mean the value of strategy s when the strategy starts at time t. So far we have worked with strategies that start execution at time 0. This theorem holds by construction of a job e with time t_e and quality q_e such that

$$V(s_i, \ldots, s_j, \sum_{k=1}^{i-1} t_k) = \sum_{l=i}^{j} [P(\sum_{r=1}^{l} t_r) - P(\sum_{r=1}^{l-1} t_r)] q_l = V(e, \sum_{k=1}^{i-1} t_k)$$

□

We now prove that a lower quality decision procedure cannot follow a higher quality one in an optimal strategy.

Lemma 7 *Let s_1 and s_2 be two strategies. If $V(s_2) < V(s_1)$ then $V(s_1 s_2) < V(s_1)$.*

Proof: Using Lemma 6, we will treat s_1 and s_2 as two single-step strategies e_1 and e_2.

$$\begin{aligned} V(e_1 e_2) &= [(P(t_{e_2} + t_{e_1}) - P(t_{e_1})) * q_{e_1}] + [(1 - P(t_{e_1} + t_{e_2})) * q_{e_2}] \\ V(e_1) &= [1 - P(t_{e_1})] * q_{e_1} \end{aligned}$$

If $q_{e_2} < q_{e_1}$, then it is easy to see that $V(e_1 e_2) < [1 - P(t_{e_1})] * q_{e_1} = V(e_1)$. □

Lemma 8 *There exists an optimal sequence that is sorted in increasing order of q's.*

Proof: Suppose this is not the case, and s is an optimal strategy. Then there must be two adjacent jobs i, $i+1$ where $q_i > q_{i+1}$ (see figure 2). Removal of job $i+1$ yields a strategy s' such that $Q_{s'}(t) \geq Q_s(t)$, from lemma 3 and the fact that $t_{i+2} \leq t_{i+1} + t_{i+2}$. By lemma 4, s' must also be optimal. We can repeat this removal process until s' is ordered by q_i, proving the theorem by reductio ad absurdum.□

Without lemma 8, there are $\sum_{i=1}^{n} \frac{n!}{i!}$ possible strategies to consider. The ordering constraint eliminates all but $\sum_{i=1}^{n} \frac{n!}{(n-i)!i!}$ strategies. It also means that in proofs of properties of strategies, we now need consider only ordered strategies. This means that M_i in equations (3) and (4) 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:

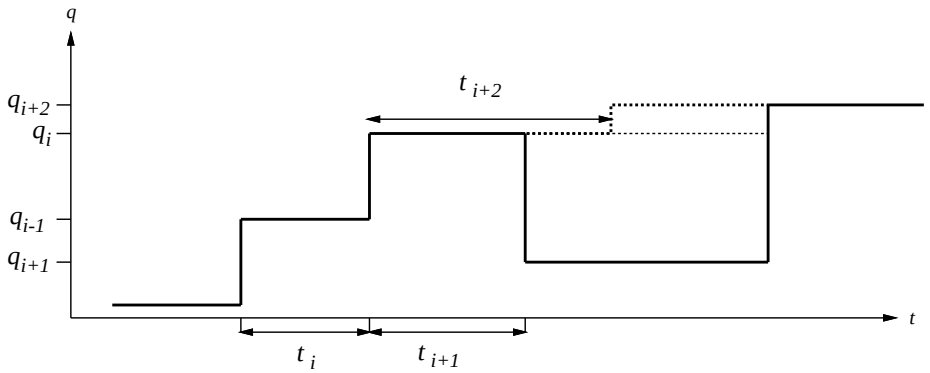

Figure 2: Proof for ordering by q_i; lower dotted line indicates original profile; upper dotted line indicates profile after removal of job $i+1$.

Lemma 9 *For every sequence $s = s_1, \ldots, s_m$ sorted in increasing order of quality, and single step z with $q_z \geq q_{s_m}$, $V(sz) \geq V(s)$.*

Proof: We calculate $V(sz) - V(s)$ using equation 3 and show that it is non-negative:

$$\begin{aligned} V(sz) - V(s) &= q_z[1 - P((\textstyle\sum_{j=1}^{m} t_{s_j}) + t_z)] - q_{s_m}[1 - P((\textstyle\sum_{j=1}^{m} t_{s_j}) + t_z)] \\ &= (q_z - q_{s_m})[1 - P((\textstyle\sum_{j=1}^{m} t_{s_j}) + t_z)] \end{aligned}$$

which is non-negative since $q_z \geq q_{s_m}$.□

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

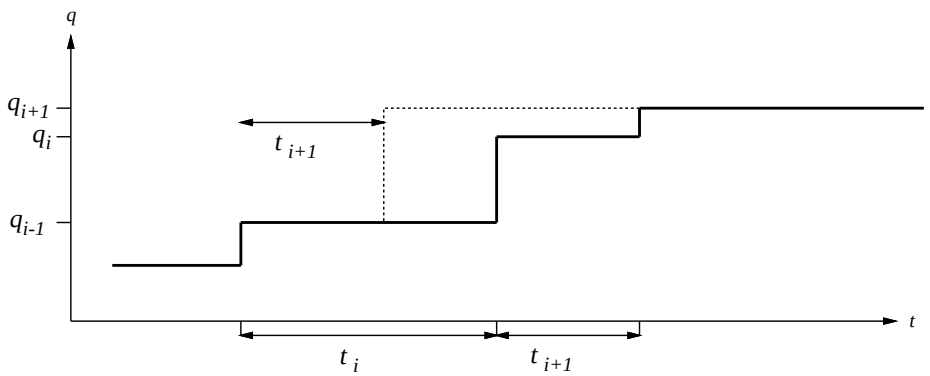

Figure 3: Proof for ordering by t_i; dotted line indicates profile after removal of job i.

The following lemma reflects the obvious intuition that if one can get a better result in less time, there's no point spending more time to get a worse result:

Lemma 10 *There exists an optimal strategy whose jobs are in nondecreasing order of t_i.*

Proof: Suppose this is not the case, and s is an optimal strategy. Then there must be two adjacent jobs i, $i+1$ where $q_i \leq q_{i+1}$ and $t_i > t_{i+1}$ (see figure 3). Removal of job i yields a strategy s' such that $Q_{s'}(t) \geq Q_s(t)$, from lemma 3. By lemma 4, s' must also be optimal. We can repeat this removal process until s' is ordered by t_i, proving the theorem by reductio ad absurdum.□

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 lemma 8, we assume that optimal strategies are ordered by q_i.

We construct an optimal sequence by using the definition of $V(s)$ in, equation (4), and produce optimal strategies that are ordered by q_i. The decomposition of the task of constructing the optimal sequence is: to find the best sequence that ends at time T, find the rule i such that the value of i appended to the sequence that ends at time $T - t_i$ is the highest. We construct 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:

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

Sequences that are optimal for the time interval $[0,T]$ contain subsequences that are optimal with respect to smaller time intervals $[0, T - t_i]$. One may wonder whether a local greedy approach would not work as well since the optimal substructure property that is common among all greedy algorithms is present in this problem. Informally, for a greedy strategy to work, we need to be able to solve subproblems independently. Unfortunately, there is dependence among solutions to subproblems as is evident in the update rule shown above.

From Corollary 1, we can read off the best sequence from the highest value in row n of the matrix M. For the case that the deadline distribution is bounded, i.e., there exists a finite $T \in \mathcal{R}^+$ such that $P(T) = 1$, we only need to consider times till T. Thus the complexity of the problem is $O(n^2T)$ since nT entries are filled in the DP table and time to compute each entry is $O(n)$. For deadline distributions that are defined over $[0, \infty]$, the complexity of finding the optimal sequence is $O(n^2L)$.

Theorem 3 *The DP algorithm computes the optimal sequence in time $O(n^2B)$ where n is the number of decision procedures in $\mathbf{R}$, and $B = min(T, L)$.*

At first glance, it may appear that we have a polynomial time method for solving the optimal sequence generation problem under stochastic deadlines. Unfortunately, this is not the case. The reason is that, by the conciseness requirement for reasonable encoding schemes, the deadline bound T as well as the sum of rule times L can be encoded by strings of length $O(logT)$ and $O(logL)$ respectively. The output, of size $O(n^2L)$ is not bounded by any polynomial function of $O(logL)$.

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. With a uniform distribution, the probability that the deadline arrives during job s_i of sequence s is independent of the time at which the job starts. This permits a simpler form of equation equation (4):

$$V(s) = q_{s_m}(1 - \sum_{i=1}^{m} P(t_{si})) + \sum_{i=1}^{m-1} P(t_{s_{i+1}}) q_{s_i}$$

Hence we have a simple recursive specification of the value $V(as)$ of a sequence beginning with job a:

$$V(as) = q_a P(t_{s_1}) - q_{s_m} P(t_a) + V(s). \tag{5}$$

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 8 and equation 5, 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.*

Exponential distributions When the deadline is drawn from an exponential distribution, the optimal sequence of decision rules can be obtained in time polynomial in the number of decision rules, and independent of the actual running times of the procedures.

Before we show this, we provide a quick primer on exponential distributions. The random variable X has an exponential distribution with parameter β ($\beta > 0$), if

$$f(X \mid \beta) = \begin{cases} \beta e^{-\beta X} & \text{for } X > 0 \\ 0 & \text{for } X \leq 0 \end{cases}$$

The mean and variance are $E(X) = \frac{1}{\beta}$ and $Var(X) = \frac{1}{\beta^2}$. The exponential distribution represents the distribution of time that elapses before the occurrence of some event, or the period between two successive events, where the events occur according to a Poisson distribution. The cumulative distribution function can be derived as follows, for $t > 0$:

$$P(t) = \int_0^t \beta e^{-\beta X} dX = 1 - e^{-\beta t}$$

The exponential distribution has the same memoryless property that the uniform distribution possesses. By this is meant that, for any $t, h > 0$,

$$Pr(X \geq t + h \mid X \geq t) = \frac{Pr(X \geq t + h)}{Pr(X \geq t)} = \frac{e^{-\beta(t+h)}}{e^{-\beta t}} = e^{-\beta h} = Pr(X \geq h)$$

That is, the probability that an event does not occur between 0 and h is the same the probability of it not occurring between t and $t + h$, given that it has not occurred before t.

Now we derive the polytime DP algorithm for the optimal sequence. We will let p_i stand for the probability that job i is interrupted, assuming it starts at 0. Then

$$p_i = P(t_i) = 1 - e^{-\beta t_i}$$

We will assume, without loss of generality, that the jobs are sorted in increasing order of quality and time. For the exponential distribution, $V(s)$ simplifies out as:

$$V(s) = \sum_{i=1}^{m-1} \left[\Pi_{j=1}^{i}(1 - p_j)\right] p_{i+1} q_i + \left[\Pi_{j=1}^{m}(1 - p_j)\right] q_m$$

We can interpret the two summands of this expression as follows:

1. If any of the jobs up to job m is interrupted, we obtain utility q_i from the previous job. The probability that job $i+1$ is interrupted is the probability that $1, \ldots i$ run without interruption, times the probability p_{i+1}, which is the probability of the job being interrupted *given no previous interruptions.*

2. The probability that the last job completes is the probability that none of the jobs are interrupted, in which case we obtain utility Q_m.

We can now derive a simple recursive specification of the value $V(as)$ of a sequence that begins with the job a.

$$V(as) = (1 - p_a)p_1 q_a + (1 - p_a)V(s)$$

We will use the state function $S_f(i)$ which represents the highest value of any rule sequence ending in f and starting with i.

$$S_f(i) = max_{j>i} \left[(1 - p_i)p_j q_i + (1 - p_i)S_f(j)\right]$$

with boundary condition $S_f(f) = q_f(1 - p_f)$. For any given f, $S_f(i)$ can be calculated in $O(n^2)$. Since the identity of the last rule is unknown (and it can any one of the n rules in R), we have the following.

Theorem 5 *The optimal sequence of decision procedures for an exponentially distributed stochastic deadline can be determined in $O(n^3)$ time where n is the number of decision procedures in R.*

The proof and argument are identical to the uniform distribution case.

4.7 Agnostic learning of decision rules

Our learning algorithm $L_{\mathcal{J},k}$ searches for the best decision rule in $\mathcal{J}_k$. In order to work in any environment, it must be *agnostic* [15] 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 [15]) 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 bounded optimality theorem

We can now state the bounded optimality theorem concerning the output of our construction procedure.

Theorem 6 *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.

4.9 Experimental results: an automated mail-sorter

We model a simple mail-sorting facility with an automated zip code recognition system (see [2, 24] for details of an actual system). Letters from a rotating bin drop on to a conveyer belt which positions them under a camera. A neural-net digit recognizer operates on the image, attempting to decipher the zipcode. There are four possible outcomes:

1. The zipcode is successfully read and the letter is sent to the correct bin for delivery.
2. The zipcode is misread and the letter goes to the wrong bin.
3. The recognizer cannot settle on a reading, and the letter is rejected.
4. The next letter arrives before the recognizer has finished, and the current letter is dispatched to whatever is currently thought to be the right zipcode, or rejected if there is no best guess.

Since incorrectly sorted letters and rejects must be sorted by hand, it is reasonable to assume they incur the same penalty. The value of a program therefore depends just on the probability p_s that it will find the correct zipcode before being interrupted.

The implementation in [2] uses a single neural network on a chip. We show that an optimized sequence of networks can do significantly better than any single network in terms of throughput or reliability. We examine the following experimental conditions:

- We assume that a network that executes in time t has a recognition quality q that depends on t. We consider $q = 1 - e^{-\lambda t}$ for $\lambda = 0.1, 0.5, 0.9$ (figure 4 (a)).
- We consider arrival time distributions that are Poisson with means $1 \ldots 10$ (figure 4 (b)). We create optimized sequences from sets of 10 networks with execution times taken at equal intervals from $t = 1$ to 10.
- We compare the optimal sequence with the best single rule; with the rule whose execution time is the mean of the distribution; and with the rule whose execution time guarantees it will complete in 90% of cases.
- We investigate both the percentage improvement in accuracy for given arrival rates, and the improvement in throughput achievable without sacrificing accuracy.

Figure 5 shows the percentage improvement in accuracy and throughput as a function of mean arrival time, compared to the best single rule. The graphs show that we can obtain improvements in throughput of up to 25%. Single rules chosen according to the other criteria (match time = mean, and 90% completion) gave throughputs up to 20 times worse than our optimal sequences.

5. 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. The same observation can be made in ordinary complexity theory: although absolute efficiency is the aim, asymptotic efficiency is the game. That a sorting algorithm is $O(n \log n)$ rather than n^2 is considered significant, but replacing a "multiply by 2" by a "shift-left 1 bit" is not considered a real advance. The slack allowed by the definitions of complexity classes is essential in allowing theoreticians to build on earlier results and to analyse the complexity of algorithms that use other algorithms as subroutines. In this section we propose a definition of asymptotic bounded optimality that we hope will have some of the same advantages, and report on some preliminary investigations into its use as a theoretical tool.

Recall that we defined a *class* of environments, $\mathbf{E}$, which is unbounded in a complexity measure on environments $n(E)$. Then we have

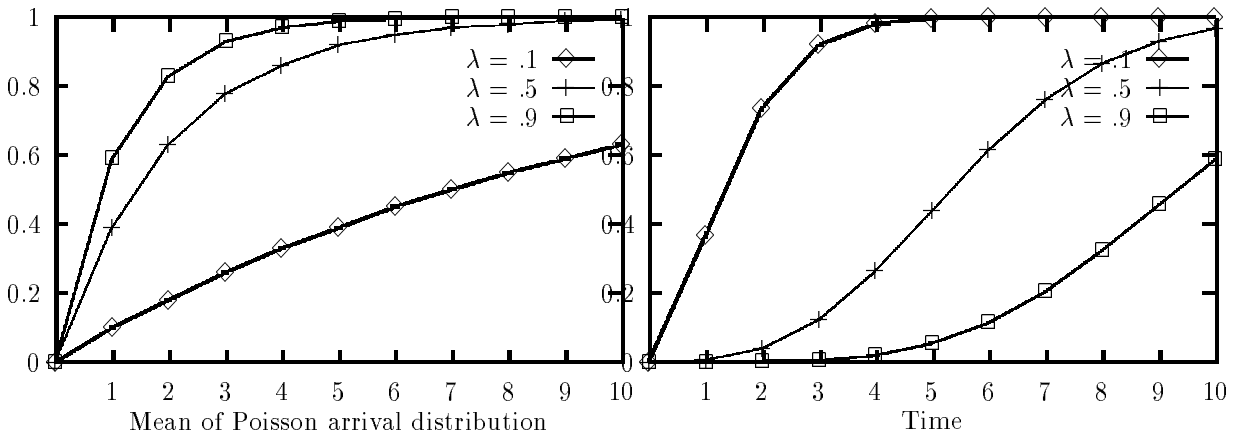

Figure 4: (a) Accuracy profiles ($1 - e^{-\lambda x}$) (b) Poisson arrival distributions (means 1, 5 and 9 shown)

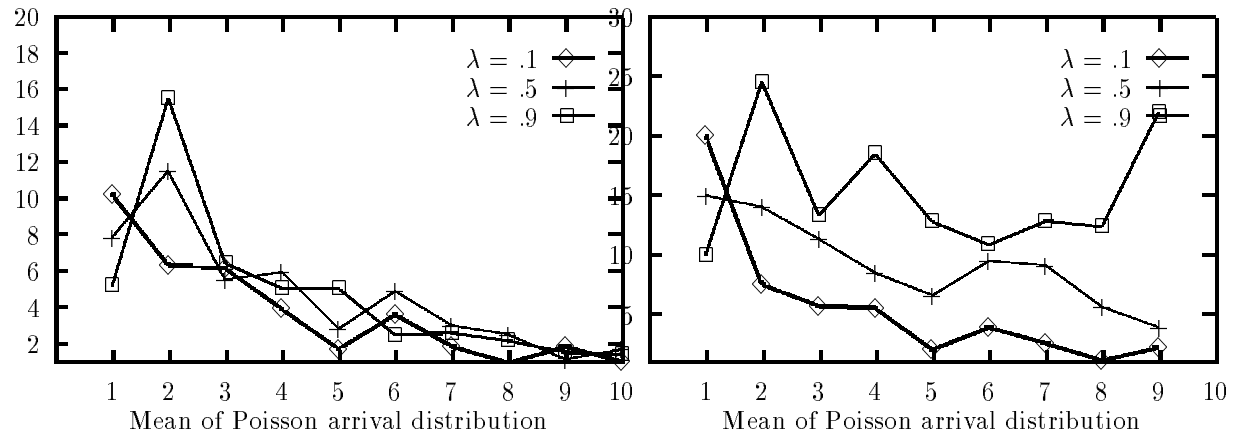

Figure 5: Percentage improvement over single-best-network: (a) accuracy with constant throughput (b) throughput with constant accuracy

Definition 11 *Asymptotic bounded optimality (ABO): We will say that an agent program l is timewise (or spacewise) asymptotically bounded optimal in* **E** *on M 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 (or with k times more memory), and l_{opt} is an optimal agent program for E on M.

In English, this means that the program is basically along the right lines if it just needs a faster (larger) machine to be as good as the best program on all problems above a certain level of difficulty. Unless k were enormous, we would be happy with a program that was ABO for a nontrivial environment on a nontrivial architecture.

In ordinary complexity theory, program l_1 is as good as program l_2 if it can solve any problem instance in time $\leq kt$, where t is the time taken by l_2 on the same problem instance. One might wonder why we chose to use the constant factor to expand the machine capabilities, rather than to increase the time available to the program. In the context of ordinary complexity theory, the two alternatives are exactly equivalent, but in the context of general time-dependent utilities, only the former is appropriate. It would not be possible to simply "let l run k times longer," because the programs we wish to consider control their own execution time, trading it off against solution quality. One could imagine slowing down the entire environment by a factor of k, but this is merely a less realistic version of what we propose.

5.1 Relationship to classical complexity theory

It is interesting to investigate the connection between ABO programs and traditional programs that have complexity equal to the lower bound for their problem class.[6] As we would hope, asymptotic bounded optimality generalizes the standard definition of asymptotic complexity. First, we define the special case of environments in which traditional programs are appropriate:

Definition 12 *Classical task environment: E is a classical task environment if*

$$V(l, E, M) \;\; = \;\; \begin{cases} f(time(l, E, M)) & \text{if } l \text{ outputs correct solution for } E \\ 0 & \text{otherwise} \end{cases}$$

where $time(l, E, M)$ is the running time for l in E on M, M is a universal Turing machine, and f is some decreasing function.

Theorem 7 *In a classical task environment, a program is timewise ABO iff it has asymptotic time complexity equal to the tight lower bound for the problem class.*

Proof: Let l be ABO in **E** on M. Then, by definition: $\exists k, n_0 \; \forall E \in \mathbf{E} \; n(E) \geq n_0 \Rightarrow V(l_{\text{opt}}, E, M) \leq V(l, E, kM)$. Given the above value function, l_{opt} is the most efficient program for the problem class. Since l is within a constant factor of l_{opt}'s execution time, it must be in the same classical complexity class. The argument in the reverse direction is similar. □

5.2 Universal asymptotic bounded optimality

Asymptotic bounded optimality is defined with respect to a specific value function V. In constructing real-time systems, we would prefer a certain degree of independence from the temporal variation in the value function. We can achieve this by defining a family $\mathcal{V}$ of value functions, differing only in their

6. This connection was suggested by Bart Selman.

temporal variation. By this we mean that the value function preserves the preference ordering of external actions over time, and all value functions in the family have the same preference ordering.[7]

For example, in the fixed-cost regime we can vary the cost c to generate a family of value functions; in the stochastic deadline case, we can vary the deadline distribution P to generate another family. Also, since each of the three regimes uses the same quality measure for actions, then the union of the three corresponding families is also a family. What we will show is that a single program, which we call a universal program, can be asymptotically bounded optimal regardless of which value function is chosen within any particular family.

Definition 13 *Universal asymptotic bounded optimality (UABO): Let $l^i_{\rm opt}$ be ABO over $\mathbf{E}$ on M for a given $V_i \in \mathcal{V}$. We will say that an agent program l_U is timewise (or spacewise) universally asymptotically bounded optimal in $\mathbf{E}$ on M over the family of value functions $\mathcal{V}$ iff*

$$\forall i\ \exists k\ \forall E \in \mathbf{E}\ V_i(l_U, E, kM) \geq V_i(l^i_{\rm opt}, E, M)$$

A UABO program must compete with the ABO programs for every individual value function in the family. A UABO program is therefore a universal real-time solution for a given task. Do UABO programs exist? If so, how can we construct them?

It turns out that we can use the scheduling device from [23] to construct UABO programs, provided the machine architecture M allows program concatenation (e.g., the LISP `prog` construct), a conditional-return construct, and includes the null progam Λ. This device was designed to reduce environments with unknown interrupt times to the case of known deadlines, and the same insight applies here. The universal program l_U has the form of a concatenation of individual programs of increasing runtime, with an appropriate termination test after each. l_U is given as an argument a "label" i which indicates the time dependence of the value function. It has the form

$$l_U = [l_0 \cdot l_1 \cdots l_j \cdots]$$

where each l_j consists of a program and a termination test. The program part in l_j is the program in $\mathcal{L}_M$ that is bounded optimal for the value function corresponding to a fixed deadline of $2^j\epsilon$, where ϵ is a time increment smaller than the execution time of any program in $\mathcal{L}_M$. The termination test causes l_U to return the best action found so far if the value of the action selected by the program part l_j is greater than or equal to the value $V_i(l^i_{\rm opt}, E, M)$, which is the value of the action chosen by the best program for the specific V_i.

Theorem 8 *l_U is UABO in $\mathbf{E}$ on M.*

Proof: We must show that

$$\forall i\ \exists k\ \forall E \in \mathbf{E}\ V_i(l_U, E, kM) \geq V_i(l^i_{\rm opt}, E, M)$$

Consider the situation just before the end of the j^{th} program in l_U. The total execution time consumed by l_U so far is $\leq (2^{j+1}-1)\epsilon/k$. The value of the best action obtained so far is is $V_i(l_{j-1}, E, M)$. Therefore l_U on kM will return a better action than $l^i_{\rm opt}$ on M provided $k \geq 4$. □

Consider the case where V_i correponds to a known deadline. In this case, we can calculate the exact runtime and design the best possible program to meet this deadline. What the theorem says is that even with complete ignorance of the time-dependency of the value function (or, even if we think the time dependency is a uniform time cost), we can construct a universal program that will do better with a machine that is four times faster. This is because the profile of l_U will dominate the profile of $l^i_{\rm opt}$. The key feature of the construction is that the time of the most recent program completed must be a constant fraction of the total time consumed so far; the factor of 2 is optimal [23].

7. The value function must therefore be *separable* [21], since this preservation of rank order allows a separate time cost to be defined.

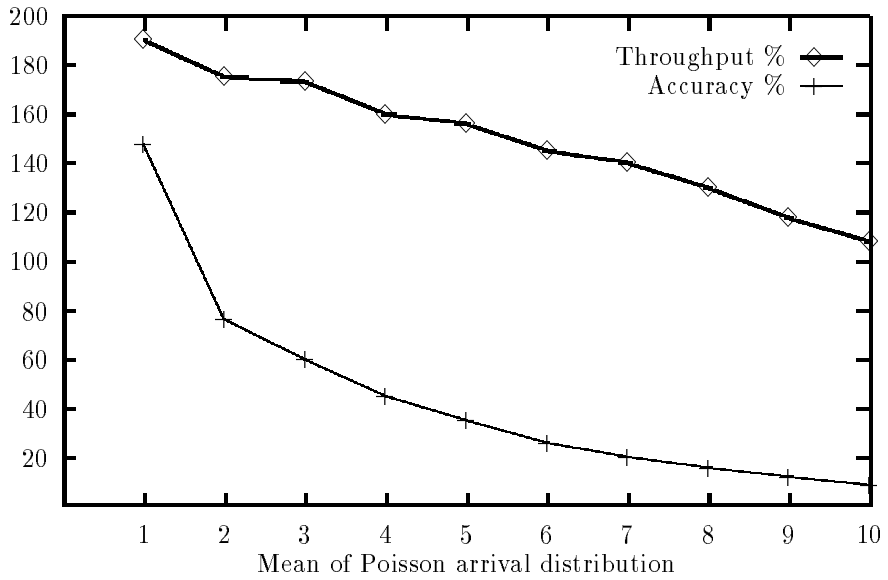

Figure 6: Throughput and acuracy improvement of l_U over l^i_{opt}, as a function of mean arrival time, for the accuracy profile $1 - e^{-0.5t}$.

Returning to the mail-sorting example, it is fairly easy to see that l_U (which consists of a sequence of networks, like the optimal programs for the stochastic deadline case) will be ABO in the fixed-deadline and fixed-cost regimes. It is not so obvious that it is also ABO in any particular stochastic deadline case — recall that all three regimes can be considered as a single family. We have programmed a constructor function for universal programs, and applied it to the mail-sorter networks. Figure 6 shows that l_U has higher throughput than l^i_{opt} across the entire range of arrival distributions.

Given the existence of UABO programs, it is possible to consider the behaviour of compositions thereof. The simplest form of composition is functional composition, in which the output of one program is used as input by another. More complex, nested compositional structures can be entertained, including loops and conditionals [30]. The main issue in constructing UABO compositions is how to allocate time among the components. Provided that we can solve the time allocation problem when we know the total runtime allowed, we can use the same construction trick as used above to generate composite UABO program, where optimality is among all possible compositions of the components. We leave discussion and proof of these results to a later paper.

6. Conclusions and Further work

In short, we are proposing a new line of inquiry into bounded optimal agents, which may provide a suitable basis for theoretical research in artificial intelligence. Asymptotic bounded optimality (ABO) in particular promises to yield useful results on composite agent designs, allowing us to separate the problem of designing complex ABO agents into a discrete structural problem and a continuous temporal optimization problem.

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. A theorem that exhibits a bounded optimal design translates, by definition, into an agent whose actual behaviour is desirable.

There appear to be plenty of worthwhile directions in which to continue the exploration of bounded optimality. From a foundational point of view, one of the most interesting questions is how the concept applies to agents which can incorporate a learning component (note that in our earlier derivations, the learning algorithm was external to the agent). In such a case, there will be not necessarily be a largely stable bounded optimal configuration if the agent program is not large enough; instead, the agent will have to adapt to a shorter-term horizon and rewrite itself as it becomes obsolete.

With results on the preservation of ABO under composition, we can start to examine much more interesting architectures than the simple production system studied above. For example, we can look at optimal search algorithms, where the algorithm is constrained to apply a metalevel decision procedure at each step to decide which node to expand, if any [20].

In the context of computational learning theory, it is obvious that the stationarity requirement on the environment, which is necessary to satisfy the preconditions of PAC results, is too restrictive. The fact that the agent learns may have some effect on the distribution of future episodes, and little is known about learning in such cases. We could also relax the episodic requirement to allow non-immediate rewards, thereby making connections to current research on reinforcement learning.

The computation scheduling problem we examined is interesting in itself, and does not appear to have been studied in the operations research or combinatorial optimization literature. We would like to resolve the formal question of its tractability in the general case, and also to look at cases in which the solution qualities of individual processes are interdependent (such as when one can use the results of another). Practical extensions include computation scheduling for parallel machines or multiple agents, and scheduling combinations of computational and physical (e.g., job-shop and flow-shop) processes, where objective functions are a combination of summation and maximization. The latter extension 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.

We believe that artificial intelligence can be usefully characterized as the study of bounded optimality, particularly in the context of complex task environments and reasonably powerful computing devices. We hope that others will take up the challenge. If bounded optimality is found wanting as a theoretical framework, then perhaps another will be created in the process.

Acknowledgements

We would like to acknowledge stimulating discussions with Michael Fehling, Michael Genesereth, Russ Greiner, Eric Horvitz, Henry Kautz and Bart Selman on the subject of bounded optimality; with Dorit Hochbaum, Nimrod Megiddo and Kevin Glazebrook on the subject of dynamic programming for scheduling problems; and with Nick Littlestone and Michael Kearns on the subject of agnostic learning. Many of the early ideas on which this work is based arose in discussions with the late Eric Wefald. Thanks also to Ron Parr, Rhonda Righter and Patrick Zieske for help with the DP algorithm. The first author was supported by NSF grants IRI-8903146 and IRI-9058427 (PYI), by a visiting fellowship from the SERC while on sabbatical in the UK, and by the NEC Research Institute. The second author was supported by NSF grant IRI-8902721.

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] Boser, B. E., Sackinger, E., Bromley, J., and LeCun, Y. (1992) Hardware requirements for neural network pattern classifiers — a case study and implementation. *IEEE Micro* **12**(1), 32-40.

[3] Brandt, R. (1953) In search of a credible form of rule utilitarianism. In Nakhnikian, G., and Castaneda, H. (eds.), *Morality and the Language of Conduct.*

[4] 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.

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

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

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

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

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

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

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

[19] Russell, S. J., and Subramanian, D. (1993) On provably rational agents with limited performance hardware. In Baum, E. (Ed.) *Computational learning and cognition: Proceedings of the Third NEC Symposium*, SIAM Press.

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

[21] Russell, S. J., and Wefald, E. H. (1989) On optimal game tree search using rational metareasoning. In *Proc. IJCAI-89.*

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

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

[24] Sackinger, E., Boser, B. E., Bromley, J., and LeCun, Y. (1992) Application of the ANNA neural network chip to high-speed character recognition. *IEEE Transactions on Neural Networks* **3**(3), 498-505.

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

[26] Simon, H. A. (1976) On how to decide what to do. In [27].

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

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

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

[30] Zilberstein, S. (1993) *Operational Rationality Through Compilation of Anytime Algorithms*, PhD thesis, Computer Science Division, University of California, Berkeley.