Constructing Bounded Optimal Systems

Stuart Russell Devika Subramanian

15 October 1992

1 Introduction

As observed in the call for proposals for this symposium, many AI problems in knowledge representation, planning, and machine learning, when stripped of their domain-specific details, turn out to problems classified as NP-hard by theoretical computer scientists. Both theorists as well as AI researchers have realized that the NP-hardness of many problems stem from defining solution optimality as a criterion for success. An obvious relaxation of this criterion is the notion of ϵ-approximation introduced by [27] and the probabilistic version introduced into the AI literature (machine learning) by [43]. Another approach to coping with NP-hardness results, which is prevalent in the inference community, and eloquently described by Kautz [18] and Levesque [20], is to concentrate on finding *polynomial-time* subclasses that allow for computation of optimal solutions: that is, circumscribe the range of problems within a class that can be solved efficiently in the worst case without losing the guarantees of solution optimality.

The approach we advocate here is aligned with the approximations camp, where we deviate is in the specification of the approximation. AI systems embedded in the world perform computation in the service of selecting a course of action. Thus, as Simon [36], Good [12], Horvitz [15], Hansson and Mayer [13] and many others have pointed out, what is needed is a theory of algorithms that maximize the *comprehensive value* of computation. In other words, since the utility of a computation and resulting action sequence is a function of both the quality of the resulting solution and the time taken to choose it, AI's mission can be seen as the development of methods for designing algorithms that maximize this combined utility. We will call systems that maximize this new utility measure *bounded optimal.* In this paper, we describe the relevance of bounded optimal systems to the foundations of AI and its relationship to traditional complexity theory, through the construction of a simple bounded optimal agent.

The problem of constructing bounded optimal systems that trade-off solution quality against solution time is a relatively new enterprise in AI with roots in early work in bounded rationality in economics [36] and more recent algorithmic work [6, 10, 15, 30, 29] in constructing anytime methods. The bounded systems we consider here select the computations that will yield the highest return in the shortest time, using as much knowledge of the domain as possible to carry out the selection.

A key question is a clean characterization of the class of problems for which bounded optimal (or approximately bounded optimal) systems can be constructed. The following intuitive account is offered here, with the hope that the presentation of the construction of the bounded agent that follows this section will sharpen the explanation. Bounded optimal systems take advantage of the structure of the domain – by structure we mean the non-uniformity of the distribution of utilities among all possible computation steps; if this distribution is both non-uniform and predictable, significant progress towards a good decision can be made quickly. *Structural variety* in an agent is therefore one of the keys to achieving bounded optimality: it makes available computation steps that can quickly narrow down the decision problem and identify actions that are likely to be close to optimal. In domains with little or no structure — for example, the problem of finding satisfying assignments for general Boolean formulæ— there will be no identifiable computation steps of high enough value to make the problem class *exactly* solvable in polynomial time. There will be no more structural regularity for the metalevel decision problem than there is for the object-level problem. Methods of the type we propose should therefore do well whenever possible, and won't do well when doing well is impossible.

This paper is organized as follows: we first define *bounded optimality*, which supersedes the notion of perfect rationality familiar from decision theory. We then examine a class of architectures for which the problem of generating bounded optimal configurations is soluble. We present a procedure that constructs approximate bounded optimal agents for a variety of time-cost regimes for any episodic environment.

2 Bounded Optimality

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 formally investigate the construction of bounded optimal programs, where a program is composed from a known set of processes generated by a learning algorithm or given *a priori*. The problem of dynamically scheduling computations to give bounded optimal performance arises naturally in real-time situations in which several possible methods are available for solving a problem, and the methods vary in their resource requirements and solution qualities. The value of a given solution depends both on its quality and on the time at which it becomes available. The natural objective function involves a *maximization* over the qualities of the solutions obtained, rather than a summation, as would be typical in job-shop scheduling; for this reason, this class of problems is not addressed by classical results. We show that for some non-trivial time cost functions (fixed time cost, known deadline, fixed but unknown deadline), it is possible to construct approximately bounded optimal programs efficiently for any episodic environment.

3 Constructing Bounded Optimal Programs

An *agent* is a physical entity that we wish to view in terms of its *perceptions* and *actions*. There are two different ways to think about an agent. An agent can be described as a mapping (the *agent function*) from percept sequences to actions; and as an *agent program* that implements the mapping. 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 mappings and their finite representations as programs running on a machine.

We will consider a physical agent as consisting of an architecture and its 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:*

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

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.

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

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

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.*

Decision theory provides a persuasive specification of the agent function f; this specification has motivated at least two extensive proposals for the design of general intelligent agents [8, 14]. A direct implementation of this specification 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 that the k-th action may be optimal only with respect to some much earlier subsequence, and is now totally inappropriate.

To escape this quandary, we propose a machine-dependent standard of rationality, in which optimality constraints are imposed on *programs* rather than *functions*, *deliberations* or *behaviours*. In common with Horvitz, we say that an agent exhibits bounded optimality if its program is a solution to the constrained optimization problem presented by its architecture.

To formalize this idea, it will be helpful to assume a 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 we can define the optimal agent program as follows:

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 induce a distribution over the state sequences generated, and the optimal program maximizes the expected utility over this distribution.

3.1 Provably ralphs

We will illustrate the construction of a bounded optimal agent with a simple form of a *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 limited rational agent, if the rule base becomes large and the conditions become complex. A limited rational 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.

Consider for example the game of tic-tac-toe (noughts and crosses). Rule conditions are constructed from boolean combinations of primitive propositions (such as "X is in the center square"), and utility is defined by winning, losing and drawing at the end of each "game" (a portion of the state sequence). A pretty good game can be played with even a very simple sequence of rules such as

1. Fill in a winning row if possible.
2. Block an opponent's winning row.
3. Construct a fork if possible.
4. If all else fails go in the center.
5. If that fails, go in a corner.
6. Otherwise go anywhere legal.

Our general scheme is to exhibit an algorithm that is capable of learning such sets of rules, and arranging them in a sequence such that the performance of an agent using the particular rule 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 $V(l, M, E, W_0) \approx V(l_{\text{opt}}, M, E, W_0)$. Moreover, we would like C to be efficient, and eventually to work for any environment and utility function, even when these are not known explicitly by the designer. As yet, we do not consider the case where C is also part of the agent design to be optimized.

In the following, we consider the utility model for various types of real-time environments, and formally specify two classes of production system architecture. We then derive suitable a construction algorithm C.

3.2 Utility specification for 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 order to talk about deadlines and time cost, it will be useful to have a notion of inaction. Let $noop^t$ denote a sequence of null actions lasting t time steps. Also, let $effects(A, W)$ denote the sequence of world states through which the action sequence A drives the environment given an initial state W. Finally, let the operator '$\cdot$' denote concatenation of sequences.

Then 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,*

$$U(W_i \cdot W \cdot \mathrm{effects}(\mathrm{noop}^t \cdot \mathrm{A}, \mathrm{W}) \cdot \mathrm{W_j}) = \mathrm{U}(\mathrm{W_i} \cdot \mathrm{W} \cdot \mathit{effects}(A, W) \cdot W_j) - ct$$

for some constant c.

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

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

for some constant T_d.

Deadlines are thus represented by a utility "cliff" occurring at some time T_d after the beginning of an episode. The case of stochastic deadlines, wherein the location of the utility cliff is not known exactly, is more complex. We assume that, for the designer, the value of $effects(noop^t, W)$ 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 after the beginning of an episode.[1]

We consider only a special case of production system, in which production rules are matched in a fixed sequence. We call a sequence of production rules a *strategy*. Each rule i has associated

[1]Strictly speaking, we must also include some known percept that immediately precedes the actual deadline, at which point the agent "knows" that there is a fixed deadline at the next step and is thereby forced to act immediately. Without this, the agent is playing a much more dangerous game, walking blindfolded towards the utility cliff.

with it a match time t_i and a quality q_i, which corresponds to the utility of the rule's recommended action when the match succeeds.

We can consider two types of production rule: the *partial* rule, where the match may fail and no action is recommended, and the *complete* rule (or *decision rule*), where an action is recommended in all circumstances. Here, we concentrate on the latter type. 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.) Our architecture can accommodate rules up to some maximum size n; we denote a production system architecture with rule language $\mathcal{J}$ and maximum size n by $M_{\mathcal{J},n}$.

In executing a typical sequence[2] $R = \langle r_1 \ldots r_m \rangle$, 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 above. 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. The construction algorithm must find the strategy with highest expected utility.

3.3 The Construction algorithm

The job of the construction algorithm C is to observe a set $\mathbf{e}$ of training episodes in E, and to build 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,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 condition parts of the 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*.

3.4 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

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

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 $V(s_1 s_2) \leq V(s_2)$.*

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

[2] Where no confusion can arise, we will use subscripts on rules to indicate their position in the sequence being discussed.

3.5 Results for fixed deadline

The value of a strategy s for this case has the form:

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

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.

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 $V(s_1 s_2) \leq V(s_2)$.*

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., $t_i \leq T_d$.*

3.6 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:

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

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 geometric intuition is given by the notion of a *performance profile*:

Definition 8 *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.*

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

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

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, as shown in figure 1. 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.

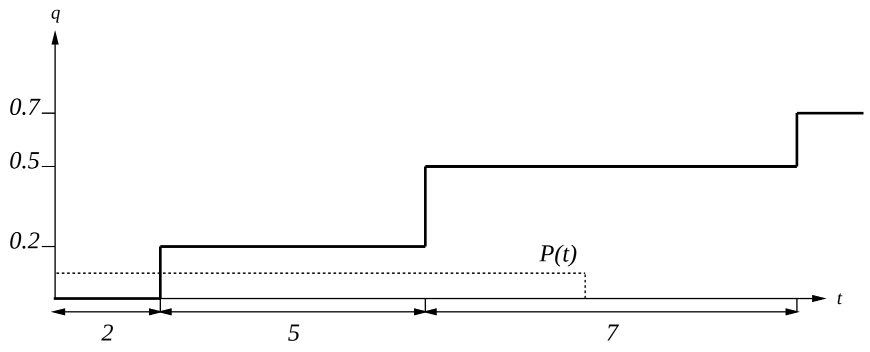

Figure 1: Performance profile for $r_1r_2r_3$, with p superimposed.

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

$$V(r_1r_2r_3) = [.2 - 0][1 - .2] + [.5 - .2][1 - .7] + [.7 - .5][1 - 1] = .25$$

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

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

This formula holds for any distribution P (proof omitted). Both formulations of the value function will be used. We now prove that a lower quality decision procedure cannot follow a higher quality one in an optimal strategy.

Lemma 4 *There exists an optimal sequence that is sorted in increasing order of q_i.*

The ordering constraint eliminates all but $\sum_{i=1}^{n} \binom{n}{i}$ 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:

Lemma 5 *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* $\mathbf{R}$.

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

3.7 A Dynamic Programming Algorithm

We devise a dynamic programming algorithm for obtaining an optimal sequence of decision rules that runs in pseudo-polynomial time. We assume that the time t_i associated with each rule is an integer (this is almost wlog, using standard rounding and scaling methods).

The dynamic programming formulation of the problem of constructing an optimal sequence uses the definition of $V(s)$ in equation (4), and assumes that optimal strategies are ordered by q_i.

The algorithm constructs the matrix $M(i,T)$, where i ranges over rules in $\mathbf{R}$ and is sorted in increasing order of quality, and T ranges from the start time 0 to the end time $L = \sum_{i \in \mathbf{R}} t_i$. Each entry $M(i,T)$ in the table stands for the best sequence that ends in rule i and at time T. The update rule is:

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

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* $\mathbf{R}$.

For the case of a uniform density function for the deadline, a similar DP algorithm constructs an optimal sequence in time $O(n^3)$.

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.*

3.8 Agnostic Learning of Decision Rules

Our learning algorithm $L_{\mathcal{J},k}$ searches for the best rule in $\mathcal{J}, k$: it must be *agnostic* [19] 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 [19]) 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. 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 $V(l_{\mathrm{opt}}) - V(l) < \epsilon + 2\epsilon_q$ 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.*

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 Summary

We have developed a scheme for scheduling computations in a bounded optimal manner in a variety of time-cost regimes. Whereas the construction is a theoretical one, it suggests that approximations cannot and should not be specified purely in terms of how far they are from an optimal solution calculated without computational constraints. The utility of a solution is to be judged in terms of the effect it has on the the actions performed by an agent noting that actions (and computations) have time value. Simon [38] urges inquiry into "the fascinating and important questions of what constitutes an efficient decision procedure, of how the structures of such procedures are related to the structures of particular kinds of decision-making environments." We share this fascination. The formulation of more realistic utility functions and methods for learning utilities of computational steps are important next steps we are engaged in. We confess, however, that the concept of bounded optimality may require several reformulations before it stabilizes into a "complexity theory" for AI. In the near future, we will need to improve our understanding of topics such as the rationality of composite systems, the learnability of first-order theories, and metalevel/object-level tradeoffs, so that we can analyse more realistic architectures than simple production systems.

5 Bibliography

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] K.R. Baker. Introduction to Sequencing and Scheduling. New York, Wiley, 1974.

[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] R.W. Conway. *Theory of scheduling.* Addison Wesley, Reading, Mass., 1967.

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

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

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

[9] M.S. Fox and S.F. Smith. ISIS-A knowledge-based system for factory scheduling. Expert System 1(1), 1984.

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

[11] Genesereth, M. R., and Nilsson, N. J. (1987) *Logical foundations of artificial intelligence.* Los Altos, CA: Morgan Kaufmann.

[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] Hansson, O., and Mayer, A. (1988) The optimality of satisficing solutions. *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, Minneapolis, MN.

[14] Hansson, O., and Mayer, A. (1989) Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Ontario.

[15] 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.

[16] Horvitz, E. J., Breese, J. S., and Henrion, M. (1988) Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, **2**, 247-302.

[17] 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.

[18] Kautz, H. (1989) Computers and thought lecture. *Eleventh International Joint Congress on Artificial Intelligence*, Detroit, MI.

[19] 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.

[20] Levesque, H. J. (1989) *Logic and the complexity of reasoning.* Technical report KRR-TR-89-2, Department of Computer Science, University of Toronto, Toronto, Ontario.

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

[22] S. Minton and A. Philips. Applying a Heuristic Repair Method to the HST Scheduling problem. Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, 1990, Morgan Kaufman.

[23] N. Muscetolla and S. Smith. Integrating Scheduling and Planning to Solve Space Mission problems. Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, 1990, Morgan Kaufman.

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

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

[26] P. Ow and S. Smith. Viewing Scheduling as an opportunistic problem-solving process. Annals of Operations Research 12:85-108, 1988.

[27] C.H. Papadimitriou and K. Steiglitz. Combinatorial Algorithms.

[28] Putnam, H. (1975) Probability and confirmation. In *Mathematics, Matter and Method.* Cambridge: Cambridge University Press.

[29] S. Russell, D. Subramanian. On Provably Rational Agents with Limited Performance Hardware. In Computational Learning and Cognition edited by E. Baum, SIAM Press, 1992.

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

[31] Russell, S. J., and Zilberstein, S (1991) In Proc. IJCAI-91, Detroit, MI.

[32] N. Sadeh and M. Fox. Focus of attention in an activity-based scheduler.

[33] Proceedings of the NASA Conference on Space telerobotics, 1989.

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

[35] Searle, J. (1990) *Desire, deliberation and action.* Unpublished manuscript.

[36] Simon, H. A. (1955) A behavioral model of rational choice. *Quarterly Journal of Economics* **69**, 99-118.

[37] Simon, H. A. (1958) Rational choice and the structure of the environment. In [39].

[38] Simon, H. A. (1976) On how to decide what to do. In [39].

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

[40] Simon, H. A. and Kadane, J. B. (1975) Optimal problem-solving search: all-or-none solutions. AI Journal (6).

[41] K. Sycara, S. Roth, N. Sadeh and M. Fox. Managing resource allocation in multi-agent time-constrained domains. Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, 1990, Morgan Kaufman.

[42] Tversky, A., and Kahneman, D. (1974) Judgement under uncertainty: Heuristics and biases. *Science* **185**, 1124-31.

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

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

[45] M. Zweben, M. Deale. Anytime Rescheduling. Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, 1990, Morgan Kaufman.

6 Appendix I : Proofs of Theorems

Lemma 7 *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),

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

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.

Theorem 6 *The optimal strategy for fixed time cost utility functions is the singleton $argmax_{i \in \mathbf{R}}[q_i - ct_i]$.*

Proof: By Lemma 7, 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 $\mathbf{R}$.

Lemma 8 *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),

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

Theorem 7 *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 6.

Lemma 9 *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:

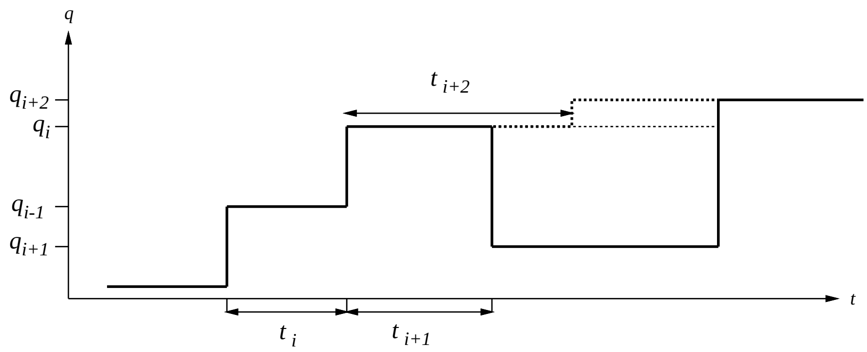

Figure 2: Proof for ordering by q_i; lower dotted line indicates original profile; upper dotted line indicates profile after removal of job $i+1$.

Lemma 10 *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).

Lemma 11 *There exists an optimal sequence that is sorted in increasing order of* q_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 $q_i > q_{i+1}$ (see figure 2). Removal of job $i+1$ yields a strategy s' such that $Q_{s'}(t) \geq Q_s(t)$, from lemma 9 and the fact that $t_{i+2} \leq t_{i+1} + t_{i+2}$. By lemma 10, 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.

Lemma 12 *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 $q_z \geq q_{s_m}$.

Lemma 13 *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 $t_i > t_{i+1}$. Removal of job i yields a strategy s' such that $Q_{s'}(t) \geq Q_s(t)$, from lemma 9. By lemma 10, 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.

When the deadline is drawn from a uniform 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. We begin by assuming that for any subset of the rules, each rule will have a non-zero probability of being matched, $P(\sum_{i \in R} t_i) < 1$. We will abandon this assumption later.

Recall that our objective is to find a sequence $s = (s_1, \ldots, s_m)$ of decision procedures drawn from the set $R = \{r_1, \ldots r_n\}$, which maximizes $V(s)$, where

$$V(s) = \sum_{i=1}^{m} \left[P\left(\sum_{j=1}^{i+1} t_{s_j}\right) - P\left(\sum_{j=1}^{i} t_{s_j}\right)\right] q_{s_i} + \left[1 - P\left(\sum_{j=1}^{m} t_{s_j}\right)\right] q_{s_m}$$

With a uniform distribution

$$P\left(\sum_{j=1}^{i} t_{s_j}\right) = \sum_{j=1}^{i} P(t_{s_j})$$

and we can simplify $V(s)$ as follows:

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

Essentially, 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. It can be shown that

$$V(s) = V(s') + (q_{s_m} - q_{s_{m-1}}) \left[1 - \sum_{i=1}^{m} P(t_{s_i})\right] \tag{5}$$

where s' is the sequence s without its last element s_m. With this recurrence we can use the dynamic programming scheme outlined before, whose complexity is linear in the sum of the times taken by the decision procedures $\sum_{i=1}^{n} t_{s_i}$. Here we would like to derive an algorithm that is polynomial in the number of procedures in the set R, and is not dependent on their running times. This requires us to rewrite the recurrence in equation 5 to eliminate the term involving the summation over times. All sequences considered here are sorted in increasing order of quality since we have already established that the optimal sequence has this property.

$$\begin{aligned} V(s) &= \sum_{i=1}^{m-1} P(t_{s_{i+1}}) q_{s_i} + q_{s_m}\left[1 - \sum_{i=1}^{m} P(t_{s_i})\right] \\ &= \left[\sum_{i=1}^{m-1} P(t_{s_{i+1}}) q_{s_i} - q_{s_m} \sum_{i=1}^{m} P(t_{s_i})\right] + q_{s_m} \\ &= V_{q_{s_m}}(s) + q_{s_m} \end{aligned}$$

where $V_q(s)$ is

$$V_q(s) = \sum_{i=1}^{m-1} P(t_{s_{i+1}}) q_{s_i} - q \sum_{i=1}^{m} P(t_{s_i}) \tag{6}$$

$V_q(s)$ can be understood as the value of working on $s = s_1 \ldots s_m$ instead of using a solution of quality q. We will call this the "partial value of s given q." It will usually be negative since q will typically be the value of the best rule. Equation 7 is the recurrence on $V_q(s)$ where s' is the sequence s with its last element removed.

$$V_q(s) = V_q(s') + P(t_{s_m})[q_{s_{m-1}} - q] \tag{7}$$

Now we can rewrite $V(s)$ as

$$\begin{aligned} V(s) &= V_{q_{s_m}}(s) + q_{s_m} \\ &= V_{q_{s_m}}(s') + P(t_{s_m})[q_{s_{m-1}} - q_{s_m}] + q_{s_m} \\ &= V_{q_{s_m}}(s') + P(t_{s_m})[q_{s_{m-1}}] + [1 - P(t_{s_m})] q_{s_m} \end{aligned}$$

Now we are ready to construct the dynamic programming algorithm to maximize $V(s)$. We let the state function $M_q(i)$ stand for the highest partial value given q of any sorted rule sequence ending in rule i:

$$M_q(i) = \max\{V_q(s) : s = s's_i\}$$

In order to get a recursive formula for $M_q(i)$, consider the sequence $s = s's_i = s''s_js_i$ ending in rule i, with rule j preceding i. We have, from equation 7,

$$M_q(i) = \max_{j<i}[M_q(j) + P(t_i)(q_j - q)]$$

with boundary condition

$$M_q(1) = -qt_{s_1}$$

For any given q, $M_q(i)$ can therefore be tabulated for $i = 1 \ldots n$ in time $O(n^2)$.

We can reconstruct the optimal sequence of decision procedures from the table of maximal partial values M as follows. Let $o = o_1 \ldots o_m$ be an optimal sequence – it is therefore sorted in increasing order of quality. Then using equation 8 we can express the value of o in terms of M_q by noting that since o is optimal, the final job o_m must be preceded by the highest-partial-value sequence ending in some penultimate job:

$$\begin{aligned} V(o) &= \max_{i \neq o_m}[M_q(i) + P(t_{o_m})q_i + (1 - P(t_{o_m}))q_{o_m}] \\ &= \max_{i \neq o_m}[M_{q_{o_m}}(i) + P(t_{o_m})q_i] + q_{o_m}[1 - P(t_{o_m})] \end{aligned}$$

Lemma 14 *The optimal sequence of decision procedures for a uniformly distributed stochastic deadline with $P(\sum_{i=1}^{n} t_{s_i}) < 1$ can be determined in $O(n^2)$ time where n is the number of decision procedures in R, and t_i is the running time of the i^{th} procedure.*

Proof: We know from Corollary 1 that the optimal rule sequence will end with rule n so we can compute M_{q_n} and apply equation 8 to find the optimal sequence.

Theorem 8 *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.*

Proof: In the simpler case where each rule has a non-zero probability of matching, q_n is the last rule in the optimal sequence. In the general case, one of the n jobs will be the last job that has a non-zero probability of completing. Any subsequent jobs can be ignored, since they do not contribute to the value of the sequence. We therefore need only calculate $M_(q_i)(i)$ for each i to determine the optimal sequence for each possible last job.