Optimal Sequencing of Contract Algorithms *

Shlomo Zilberstein [a] François Charpillet [b] Philippe Chassaing [c]

[a] *Department of Computer Science*
University of Massachusetts
Amherst, MA 01003, U.S.A.
E-mail: Shlomo@cs.umass.edu

[b] *LORIA-INRIA*
B.P. 239
54506 Vandoeuvre-lès-Nancy Cedex, France
E-mail: Francois.Charpillet@loria.fr

[c] *Institut de Mathématiques Elie Cartan*
Université Henri Poincaré Nancy I, B.P. 239
54506 Vandoeuvre-lès-Nancy Cedex, France
E-mail: Philippe.Chassaing@iecn.u-nancy.fr

We address the problem of building an interruptible real-time system using non-interruptible components. Some artificial intelligence techniques offer a tradeoff between computation time and quality of results, but their run-time must be determined when they are activated. These techniques, called *contract algorithms*, introduce a complex scheduling problem when there is uncertainty about the amount of time available for problem-solving. We show how to optimally sequence contract algorithms to create the best possible interruptible system with or without stochastic information about the deadline. These results extend the foundation of real-time problem-solving and provide useful guidance for embedding contract algorithms in applications.

Keywords: contract algorithms, flexible computation, real-time problem solving, mathematical foundation, meta-level control

* This work was supported in part by the National Science Foundation under grants No. IRI-9624992 and IRI-9907331, by an NSF/INRIA cooperative research grant, and by the LIRE Cooperative Program at INRIA.

1. Introduction

Since the mid 1980's, the artificial intelligence (AI) research community has produced a large body of work on incremental problem solving techniques such as *anytime algorithms* [4,18] and *flexible computation* [10,11]. Numerous such algorithms have been constructed for solving core AI problems such as heuristic search, constraint satisfaction, planning and scheduling, and diagnosis. The working notes of the 1996 AAAI Fall Symposium on Flexible Computation offer a good sample of such techniques and applications [14]. The resulting systems have proved useful in addressing the high computational complexity and the inherent uncertainty associated with AI problem-solving techniques.

Both anytime algorithms and flexible computation refer to problem-solving techniques that are interruptible, making it possible to stop deliberation and select action at any time. Solution quality typically improves as computation time increases. Therefore, these techniques facilitate the construction of systems that can operate with limited computational resources and react to dynamic changes in their environment. When combined with an appropriate meta-level control, anytime algorithms make it possible to build systems that optimize the amount of deliberation based on the actual progress they make and the urgency to take action [4,11].

Some anytime algorithms, however, are not interruptible. Such algorithms, called *contract algorithms* [18], require the amount of run-time to be determined prior to their activation. In other words, contract algorithms offer a tradeoff between computation time and quality of results, but they do not produce a constant stream of results. Once activated with a particular contract time, a contract algorithm may not produce any useful result before the end of the contract.

There are several reasons why certain problem-solving techniques are non-interruptible. One general class of contract algorithms uses the contract time to tune up the algorithm so as to produce a result within that time. The algorithm's run-time in this case is determined by a set of internal parameters. For example, game playing programs based on adversarial search can perform depth-bounded search using a heuristic evaluation function to estimate the "goodness" of non-terminal states. For any given time allocation, one can set up an appropriate depth limit so as to complete the search within the available time. Another example is using state-space abstraction to reduce the amount of time it takes to solve a problem. This technique can reduce the computational cost of finding

a path between two points on a grid or computing a policy given a Markov decision process. A more precise problem description leads to a more valuable result, but it also takes more time to compute. Once a new resolution is used, the algorithm starts solving the problem from scratch, leading to a non-interruptible approach. Additional examples of AI techniques that fall in this category are certain model-based diagnosis techniques [15], design-to-time scheduling [6], and some job scheduling techniques [5].

Another important class of contract algorithms includes systems composed of two or more anytime problem-solving techniques. Composition, even simple sequencing, destroys interruptibility. For example, a medical treatment system that is composed of an anytime diagnosis module and an anytime treatment planning module is not interruptible. Treating the resulting system as a contract algorithm is more appropriate. The problem of optimal allocation of a given contract time among the components of the composed system has been studied for a variety of program structures [20].

This paper addresses the problem of building an interruptible real-time system using contract algorithms. The key question is how to use prior knowledge about the performance of the algorithm and the expected deadline in order to produce the best possible result by the deadline. This is done by activating the contract algorithm multiple times with different contracts. When the system is interrupted, the best available solution is returned. Section 2 provides a formal definition of the contract sequencing problem. Section 3 describes an optimal solution to the problem when no information is available about the deadline. In Section 4, we show how to handle quality uncertainty and stochastic deadlines. Section 5 provides an optimal solution to the sequencing problem when a time-dependent utility function is given instead of a strict deadline. In Section 6, we discuss several lines of related work. The paper concludes with a summary of the contribution of this work and some open questions.

2. Using contract algorithms in interruptible domains

Suppose that a contract algorithm, $\mathcal{A}$, is used in an interruptible domain in which the amount of time available for problem solving is unknown in advance. At a particular time, the system receives a *deadline* signal indicating that the computation must be terminated and the best available result must be returned. Examples of interruptible domains include a diagnosis system in an intensive

care unit; a data visualization program that may be interrupted by its user; and a scheduling program that may need to return a new schedule once processors are ready to accept new tasks.

We use in this paper prior information about the performance of the contract algorithm. This information is typically based on an empirical evaluation of the algorithm with a large set of problem instances. We begin with a simple form of performance profile [4,3] and generalize it in Section 4.

Definition 1. A **performance profile**, $Q_{\mathcal{A}}(t)$, of a contract algorithm $\mathcal{A}$, denotes the output quality as a function of contract time t.

The performance profile is assumed to be a monotone increasing and continuous function of time.

$$t_1 < t_2 \Leftrightarrow Q_{\mathcal{A}}(t_1) < Q_{\mathcal{A}}(t_2)$$

Monotonicity is a standard property of anytime algorithms that can be guaranteed if the best result rather than the most recently generated one is returned. Strict monotonicity and continuity are assumed in order to simplify the analysis in Section 3.

If the amount of time available for computation is known in advance, the best strategy to maximize the quality of the result is to run the contract algorithm once giving it all the available time. What happens when there is no information (or only stochastic information) about the deadline? The contract algorithm should be activated with some contract time x_1. If the algorithm completes its execution before the deadline, it should be reactivated with a new contract x_2 and so on. Because of the monotonicity of $Q_{\mathcal{A}}$, it is never beneficial to use a short contract following a longer one. Therefore, we get the following sequence of contracts:

$$x_1 < x_2 < \cdots < x_i < x_{i+1} \cdots$$

Suppose that an interruptible anytime algorithm $\mathcal{B}$ is constructed using $\mathcal{A}$ with the sequence of contracts $X = (x_1, x_2, \ldots)$. Whenever $\mathcal{B}$ is interrupted, it should return the result obtained by the most recently completed contract. No solution is available before the termination of the first contract (which is arbitrarily small). Therefore, the performance profile of the interruptible algorithm, $\mathcal{B}$, is as follows.

$$Q_{\mathcal{B}}(t) = \begin{cases} 0 & \text{if } t < x_1 \\ Q_{\mathcal{A}}(x_1) & \text{if } x_1 \leq t < x_1 + x_2 \\ & \vdots \\ Q_{\mathcal{A}}(x_i) & \text{if } \sum_{j=1}^{i} x_j \leq t < \sum_{j=1}^{i+1} x_j \end{cases} \quad (1)$$

What is the sequence of contracts that produces the best anytime algorithm? To answer this question we must first define the notion of "best" sequence. If the deadline, d, is known in advance, the best quality $Q_{\mathcal{A}}(d)$ can be guaranteed. Suppose now that an interruptible algorithm is created by a sequence of contracts. We want the interruptible algorithm to guarantee the *same* quality as the contract algorithm, if it runs on a processor that is accelerated by a factor of $r \geq 1$. This definition follows the notion of *bounded optimality*[1] defined by Russell, Subramanian and Parr [16]. In fact, the results presented in Section 3 can be interpreted as the construction of a bounded-optimal interruptible algorithm from the contract one. Moreover, we prove that the minimal acceleration needed by *any* interruptible algorithm that matches the quality of $\mathcal{A}$ is 4. The acceleration ratio is defined as follows.

Definition 2. Let $\mathcal{A}$ be a contract algorithm and $\mathcal{B}$ an interruptible algorithm produced by the sequence of contracts $X = (x_1, x_2, ...)$, then the **acceleration ratio** of X, $r \geq 1$, is the smallest constant c for which:

$$\forall t \geq \frac{x_1}{c} : \quad Q_{\mathcal{B}}(ct) \geq Q_{\mathcal{A}}(t) \quad (2)$$

The acceleration ratio is the minimal acceleration that guarantees that the interruptible algorithm will always have a solution ready from the previous contract that is at least as good as the one produced by the contract algorithm when the time of interruption is known in advance. The condition $t \geq \frac{x_1}{c}$ is needed because no solution is available before the termination of the first contract. Note that the acceleration ratio does not imply that any application *must* utilize a

[1] The term bounded optimality has been used also by Horvitz [10] to refer to the optimization of computational utility given a set of assumptions about expected problems and constraints in reasoning resources. This definition is broader than the one used in this paper. Our notion of bounded optimality is more similar to the one used by Russell, Subramanian and Parr [16]. In particular, the latter definition focuses on optimization of program design as opposed to optimization of the actions performed by an agent. The optimal sequencing of contract algorithms is an example of program construction mechanism that fits that definition.

faster processor; it is only a performance measure of a given sequence of contracts. It is also important to note that the acceleration ratio is a worst-case measure. An acceleration of r is only needed when, in the worst case, the last contract algorithm is interrupted near its termination time. On average, however, a lower acceleration is needed. This has been confirmed empirically by Pos [15].

3. Optimal sequencing of contracts

Russell and Zilberstein have shown that a *particular* sequence of contracts requires an acceleration ratio of 4 [18,21], thus establishing an upper bound on the optimal solution. The sequence of contracts achieving this ration is a geometric series with run-time being doubled at each activation. This result is summarized by the following theorem.

Theorem 1. [Russell and Zilberstein, 1991] For any contract algorithm $\mathcal{A}$, an interruptible algorithm $\mathcal{B}$ can be constructed such that $Q_{\mathcal{B}}(4t) \geq Q_{\mathcal{A}}(t)$.

This section introduces a stronger result and a generalization of the above theorem. First, we prove that the acceleration ratio of 4 is the best possible over any sequence of contracts. Then, the result is generalized to the case of multiple problem instances.

3.1. Solving a single problem instance

We first show that the above sequence construction is in fact optimal in the sense that it requires the minimal acceleration ratio. In general, the acceleration ratio must hold in the worst possible case: when the interruptible algorithm is stopped just before the end of the current contract x_{i+1} rendering that contract useless. The best result produced by the previous run (with contract time x_i) is returned. This leads to the following property.

Lemma 1. Equation *(2)* is equivalent to

$$\forall i \geq 1: \quad Q_{\mathcal{A}}(x_i) \geq Q_{\mathcal{A}}(\sum_{j=1}^{i+1} x_j \ /c) \tag{3}$$

Proof: We first show that (2) $\Rightarrow$ (3). Consider some $i \geq 1$ and let

$$s(i) = \frac{1}{c} \sum_{j=1}^{i} x_j.$$

Let $t \rightarrow x_-$ denote the fact that t approaches x from below. Given that Condition (2) holds in particular for $t \rightarrow s(i+1)_-$, we get:

$$\lim_{t \rightarrow s(i+1)_-} Q_{\mathcal{B}}(ct) \geq \lim_{t \rightarrow s(i+1)_-} Q_{\mathcal{A}}(t)$$

From (1) we know that for any t, $s(i) \leq t < s(i+1)$:

$$Q_{\mathcal{B}}(ct) = Q_{\mathcal{A}}(x_i).$$

Therefore we obtain:

$$\forall i \geq 1: \quad Q_{\mathcal{A}}(x_i) \geq \lim_{t \rightarrow s(i+1)_-} Q_{\mathcal{A}}(t).$$

Finally, by the continuity of $Q_{\mathcal{A}}$:

$$\forall i \geq 1: \quad Q_{\mathcal{A}}(x_i) \geq Q_{\mathcal{A}}(s(i+1)),$$

that is:

$$\forall i \geq 1: \quad Q_{\mathcal{A}}(x_i) \geq Q_{\mathcal{A}}(\sum_{j=1}^{i+1} x_j \ /c).$$

The converse, (3) $\Rightarrow$ (2), is straightforward. Consider any t such that $s(i) \leq t < s(i+1)$. We have

$$\begin{aligned} Q_{\mathcal{B}}(ct) &= Q_{\mathcal{A}}(x_i) \\ &\geq Q_{\mathcal{A}}(\sum_{j=1}^{i+1} x_j \ /c) \\ &\geq Q_{\mathcal{A}}(t). \end{aligned}$$

■

To summarize, in order to show that a particular sequence of contracts is optimal, it is sufficient to show that its acceleration ratio is the smallest required to satisfy Equation (3).

Theorem 2. The minimal acceleration ratio needed to construct an interruptible algorithm from a given contract algorithm is $r = 4$.

Proof: From Lemma 1 we know that for any sequence of contracts, $X = (x_1, x_2, ...)$, r must satisfy:

$$\forall i \geq 1: \quad Q_{\mathcal{A}}(\frac{x_1 + x_2 + \cdots + x_{i+1}}{r}) \leq Q_{\mathcal{A}}(x_i)$$

From the strict monotonicity of $Q_{\mathcal{A}}$ we get:

$$\forall i \geq 1: \quad \sum_{j=1}^{i+1} x_j \leq r x_i$$

Setting $g_i = \sum_{j=1}^{i} x_j$, the previous equation can be rewritten:

$$\forall i \geq 2: \quad g_{i+1} \leq r(g_i - g_{i-1}). \tag{4}$$

The sequence $(g_1, g_2, ...)$ is an increasing sequence of positive numbers. Let ρ be the infimum[2] of the sequence.

$$\rho = \inf\{\frac{g_2}{g_1}, \frac{g_3}{g_2}, ..., \frac{g_{i+1}}{g_i}, ...\}$$

It is clear that $\rho \geq 1$. From Equation 4 we obtain

$$\forall i \geq 2: \quad g_i \geq g_{i-1} + \frac{g_{i+1}}{r},$$

and thus, for any $i \geq 2$ we have

$$\frac{g_i}{g_{i-1}} \;\geq\; 1 + \frac{g_{i+1}}{r g_{i-1}} \;\geq\; 1 + \frac{\rho^2}{r}.$$

Because $\{\forall i \; x_i \geq a\} \Rightarrow \{\inf_i x_i \geq a\}$, we conclude that

$$\rho \geq 1 + \frac{\rho^2}{r}.$$

Equivalently, because $\rho > 1$, we get

$$r \geq \frac{\rho^2}{\rho - 1}.$$

The function $f(\rho) = \frac{\rho^2}{\rho - 1}$ has a minimum of $f(2) = 4$ on the interval $(1, \; +\infty)$. Therefore, $r \geq 4$. ■

Therefore, the minimal acceleration ratio needed to construct an interruptible algorithm from a given contract algorithm is 4. This factor can be reduced by using additional processors. (In the multi-processor case, all the processors are accelerated by the same factor r.) In general, the minimal acceleration ratio

[2] The infimum of a set S is the greatest lower bound of S.

goes down as the number of processors grows. The acceleration ratio approaches 1 as the number of processors approaches infinity because with unlimited number of processors we can have a dedicated processor executing a single contract for any possible deadline.

The more practical question of finding good schedules for a small number of processors has been studied recently by Bernstein and Zilberstein [2]. The result is summarized by the following theorem.

Theorem 3. [Bernstein and Zilberstein, 2000] An upper bound on the best acceleration ratio that can be achieved with k processors is:

$$r = \frac{(k+1)^{\frac{k+1}{k}}}{k}$$

In the case of two processors, for example, the best known acceleration ratio is $3\sqrt{3}/2 \approx 2.598$. A lower bound of 2 can be established by showing that any lower acceleration ratio violates Theorem 2. This leaves open the general problem of optimal sequencing of a contract algorithm using $k > 2$ processors.

3.2. Solving multiple problem instances

We now generalize Theorem 2 to the case in which a contract algorithm with a performance profile $Q_{\mathcal{A}}$ is used to solve m independent problem instances. The goal is to maximize the *minimal* solution quality over all problem instances. When the deadline d is known in advance, the obvious best strategy is to divide all the available time equally among the problem instances guaranteeing a minimal quality of $Q_{\mathcal{A}}(\frac{d}{m})$. The question is how to construct the best interruptible anytime algorithm when the deadline is unknown. As with a single task, we want to find the minimal acceleration needed to match the performance of the contract algorithm with a known deadline.

There are many situations in which multiple instances of a problem must be solved in real-time. Examples include search problems with multiple possible starting states; code optimization of several tasks to be executed on a parallel machine; and compression of a file divided into m blocks to be transmitted through m equivalent communication channels. In all these examples, the objective is to maximize the minimal quality over all problem instances by the deadline.

Another application relates to the continual computation model developed by Horvitz [13]. In this model, an interactive system uses idle time in order to

solve possible future problems so that those results will be ready when needed. This approach can be used to solve anticipated problems during idle time of an interactive system. When the user interrupts the computation by making a specific choice, the system can respond using the best available solution for that choice. When a set of problem instances are equally likely to be selected, the goal of continual computation is to maximize the minimal quality of the solutions available at the deadline. This is exactly the problem addressed in this section.

Lemma 2. There exists an optimal sequence of contracts that (1) always improves the solution of a problem with the lowest current solution quality and (2) solves the problems in their original order in a round robin fashion.

Proof: The first part is true because any optimal sequence of contracts that violates this rule can be easily transformed into a sequence that always improves the lowest available solution quality. Any contract that does not improve the minimal solution quality, does not improve the overall quality and has no immediate effect on the performance profile. Therefore, the first contract that does improve the minimal solution quality can be placed ahead of it. This leads to a desired non-decreasing sequence of contracts. The second part of the lemma is an immediate result of the first part and the symmetry among the initial set of problem instances. Due to symmetry, one can choose arbitrarily among the problem instances that share the lowest solution quality. In particular, the original order can be used. ■

Let $X = (x_1, x_2, ...)$ be a sequence of contracts that meets the requirements of Lemma 2. Then, the performance profile of the interruptible algorithm solving the m problem instances is as follows.

$$Q_{\mathcal{B}}(t) = \begin{cases} 0 & \text{if } t < x_1 + x_2 + ... + x_m \\ Q_{\mathcal{A}}(x_1) & \text{if } \sum_{j=1}^{m} x_j \leq t < \sum_{j=1}^{m+1} x_j \\ \vdots & \\ Q_{\mathcal{A}}(x_i) & \text{if } \sum_{j=1}^{i+m-1} x_j \leq t < \sum_{j=1}^{i+m} x_j \end{cases}$$

This is due to the fact that the first (and shortest) among the last m contracts defines the overall quality.

We want to find the minimal acceleration ratio c such that $Q_{\mathcal{B}}(ct)$ delivers a result at least as good as the one returned by $\mathcal{A}$ when the deadline is known (i.e., $Q_{\mathcal{A}}(\frac{t}{m})$).

Lemma 3. For any $c \geq 1$ the condition:

$$\forall t \geq \frac{x_1 + ... + x_m}{c} : \quad Q_{\mathcal{B}}(ct) \geq Q_{\mathcal{A}}(\frac{t}{m})$$

holds if and only if:

$$\forall i \geq 1 : \quad Q_{\mathcal{A}}(x_i) \geq Q_{\mathcal{A}}(\sum_{j=1}^{i+m} x_j \ /cm)$$

This lemma is a straightforward generalization of Lemma 1. The complete proof is omitted.

Theorem 4. The minimal acceleration ratio needed to construct an interruptible algorithm to solve m problem instances with a given contract algorithm is $r = (\frac{m+1}{m})^{m+1}$.

The complete proof, similar to that of Theorem 2, is given in Appendix A. Note that the ratio $(\frac{m+1}{m})^{m+1}$ is obtained by a sequence of contracts defined by a geometric series with run-times being multiplied by a factor of $\frac{m+1}{m}$. In other words:

$$X = (1, \frac{m+1}{m}, ..., (\frac{m+1}{m})^i, ...).$$

Therefore, the construction described by Russell and Zilberstein [18] is the solution for the special case in which $m = 1$.

4. Handling quality uncertainty and stochastic deadlines

In this section we generalize the problem of sequencing contract algorithms to situations in which some information about the deadline is available. It is not surprising that there is no closed-form solution to the sequencing problem in this case. Instead, the optimal sequence of contracts can be determined using dynamic programming. To be able to apply dynamic programming, the problem is formalized using a discrete representation of time (as opposed to the continuous representation used in the previous sections).

4.1. Stochastic deadlines

A stochastic deadline is specified by a prior probability distribution that the deadline will occur at time t measured relative to the activation time of the contract algorithm.

Definition 3. A **stochastic deadline** is a probability distribution, $P_{\mathcal{D}}(t)$, of the deadline over time.

The value of a result of quality q is defined by the following utility function.

Definition 4. A **utility function**, $U(q)$, is the value of a solution of quality q produced by the contract algorithm.

We assume that the utility function is non-decreasing. Therefore any strategy to optimize utility is based on optimizing quality. The contract sequencing problem in this case is defined as follows. Given a contract algorithm, a stochastic deadline and an arbitrary utility function, find an optimal policy to activate a sequence of contracts. An optimal policy is one that maximizes the expected utility of the available solution at the deadline. This is one difference between the goal in the previous section of optimizing *worst-case* quality. Note that the technique developed in this section can be used to optimize the worst-case quality (rather than expected utility). However, optimizing expected utility seems to be a more appropriate objective when a stochastic deadline is given.

We approach the problem by defining a corresponding Markov decision process (MDP) and finding an optimal policy by solving it. The states of the MDP are (q,t) indicating the availability of solution of quality q at time t. In addition, there is a terminal state associated with each solution quality q, indicating reaching the deadline with that quality. The latter states indicate termination of the computation with quality q, while the states (q,t) indicate an intermediate state of the computation. The action or decision taken at each non-terminal state is to activate the contract algorithm with a new contract time τ.

Definition 5. Let $P_{\mathcal{D}}(m|n)$ be the probability distribution of the deadline occurring at time m $(m > n)$ given that it has not occurred so far and the current time is n.

Note that $P_{\mathcal{D}}(m|n)$ can be easily computed from the prior probability $P_{\mathcal{D}}(t)$ as follows.

$$P_{\mathcal{D}}(m|n) = P_{\mathcal{D}}(m)/(1 - \sum_{i=1}^{n} P_{\mathcal{D}}(i)) \tag{5}$$

This can be easily obtained by observing that n stands for the proposition that the deadline did not occur at time points $\{1, ..., n\}$ and by applying Bayes' rule.

Definition 6. The **completion probability** of a contract τ starting in state (q, t) is

$$c(t, \tau) = \sum_{i \geq t+\tau} P_{\mathcal{D}}(i|t)$$

The completion probability is the probability that the contract algorithm will terminate before the deadline occurs.

We now define the following value function over states:

$$V(q, t) = \max_{\tau} \{c(t, \tau)\ V(Q_{\mathcal{A}}(\tau), t + \tau)\ +\ (1 - c(t, \tau))\ U(q)\} \tag{6}$$

The value of a contract τ at state (q, t) depends on whether the deadline occurs during the contract or not. If the contract algorithm completes its execution (with probability $c(t, \tau)$), then the value depends on the new quality and time. If the deadline occurs, a final state is reached with a reward that depends on the quality of the existing solution. The value function is defined based on the best action τ in each state.

Theorem 5. The contracts that maximize the value function defined in *Equation (6)* provide an optimal solution to the contract sequencing problem for a given stochastic deadline.

Proof: Because of the one-to-one correspondence between the sequencing problem and the MDP and because the stochastic performance profile and the deadline satisfy the Markov assumption, it is obvious that an optimal policy for the MDP provides an optimal solution to the sequencing problem. ■

Many existing algorithms can be used to solve Equation (6). Because this is a finite-horizon MDP with no cycles (time always moves forward), dynamic programming can be used to determine the best action for each state in one

sweep of the state space. The policy can be constructed off-line once for a given deadline distribution, offering efficient reactive meta-level control.

One concern about these meta-level control policies is the size of the state space which is affected by the choice of time units and quality units. Contract time, in particular, may range over a large interval. The size of the policy, however, can be reduced by selecting coarse units. The effect of unit size on the effectiveness of meta-level control policies has been studied recently for the progressive processing model [23]. Although these base-level deliberation modules are somewhat different, the results are very encouraging. They show that by varying the unit size one can achieve a dramatic reduction in policy construction time with only a small relative error. For example, for a particular class of progressive processing tasks, construction time was reduced from more than 88 minutes to less than 1 second, introducing a value error of less than 2%.

4.2. Handling quality uncertainty

Until now, we assumed that the quality produced by the anytime algorithm is deterministic for any given contract. In practice, there may be some uncertainty about the quality. Hence we generalize the performance profile as follows.

Definition 7. A **stochastic performance profile**, $P_{\mathcal{A}}(q|t)$ of a contract algorithm, $\mathcal{A}$, denotes the probability of getting solution of quality q with contract time t.

We assume that the quality of the result is stochastic, but observable. That is, once the contract algorithm completes its execution, the *actual* quality of the solution produced by the algorithm can be determined. The quality of a plan, for example, is observable if it is measured by the sum of the costs of the operators. However, when solution quality is defined as the *approximation ratio* with respect to the *optimal* solution it may not be observable. Hansen and Zilberstein [8] have studied situations in which solution quality is partially observable. Their approach to the problem could be used to augment the framework presented in this paper.

The case of a stochastic performance profile can be solved using a similar approach by simply redefining the value function. The outcomes of actions (contracts) in this case depend on the stochastic performance profile and the likelihood of reaching the deadline, both of which satisfy the Markov assump-

tion (i.e., they are independent of previous states, given the current state of the computation). The value of a state is the expected value of the best contract τ defined as follows.

$$V(q,t) = \max_{\tau} \left\{ c(t,\tau) \sum_{q'} P_{\mathcal{A}}(q'|\tau) V(\max(q,q'), t+\tau) \; + \; (1 - c(t,\tau)) \; U(q) \right\} \tag{7}$$

In this case, it is necessary to average over all possible outcomes of a contract. Note that if the new quality is lower than the current solution quality, the better solution is used rather than the most recent one. As in the deterministic case, the resulting policy is an optimal solution to the contract sequencing problem.

5. Optimal sequencing with no strict deadline

In this section we consider the case in which there is no strict deadline. Instead, the value of the solution decreases over time as specified by a time-dependent utility function [4,3].

Definition 8. A **time-dependent utility function**, $U(q,t)$, is the value of a solution of quality q produced by the contract algorithm and returned at time t.

The contract sequencing problem in this case is defined as follows. Given a contract algorithm, its stochastic performance profile and an arbitrary time-dependent utility function, find the best policy to activate a sequence of contracts. As in the previous section, an optimal policy maximizes the expected utility of the returned solution. Note that in this case the meta-level control problem is not only to determining the next contract, but also when to stop the computation and return the current best result.

Similar to the previous section, we approach the problem by defining a corresponding MDP with states representing the current quality and time. The action taken at each state, however, is either to terminate the computation and return the current solution of quality q or activate the algorithm with a particular contract τ. We define the following value function over states.

$$V(q,t) = \max_{d} \begin{cases} \sum_{q'} P_{\mathcal{A}}(q'|\tau) V(\max(q,q'), t+\tau) \\ \quad \text{if } d\text{=continue with contract } \tau \\ U(q,t) \\ \quad \text{if } d\text{=terminate the computation} \end{cases} \tag{8}$$

When a new contract is activated, the value is defined by the distribution of output quality and the contract time. If the computation is terminated, the value is the utility of returning a result of quality q at time t.

Theorem 6. The policy that maximizes the value function defined in *Equation (8)* provides an optimal solution to the contract sequencing problem for a given time-dependent utility function.

Proof: Again, this is an immediate result of the one-to-one correspondence between the optimal sequencing problem and the above MDP and because the stochastic performance profile satisfies the Markov assumption. ■

Using dynamic programming, the optimal policy can be constructed off-line once for a given time-dependent utility function, offering an efficient, reactive meta-level control.

6. Related work

Meta-level control of computation has been studies extensively over the past decade. Horvitz has studied a wide range of meta-reasoning techniques for such applications as sorting, theorem proving, traveling salesman problems, computer graphics, and most notably flexible diagnosis algorithms [10–12]. Horvitz has identified several classes of functions to describe the utility of computation, including *urgency*, *deadline*, and *urgency-deadline* [11]. Urgency refers to utility functions that assign a cost to the delay in action. The deadline pattern refers to cases in which results are not valuable at all if the delay is greater than a certain constant. The urgency-deadline requires consideration of both the cost and availability of time. Given such utility functions, Horvitz describes a variety of meta-reasoning strategies for *interruptible* problem-solving techniques. Dean and Boddy have developed a stopping criterion for an anytime planning algorithm [4,3]. The model uses a time-dependent utility function, similar to what

we use in Section 5. Russell and Wefald have developed a general framework for meta-reasoning and applied it to control search in game playing programs [17]. Hansen and Zilberstein have developed a comprehensive solution to the problem of monitoring and control of interruptible anytime algorithms [8,9]. The solution covers several probabilistic representations of performance profiles and utility functions. An important part of the solution is the ability to factor the cost of monitoring the computation and/or the state of the environment. Sandholm and Lesser have presented an algorithm for determining the optimal stopping policy for incomplete decision algorithms given a cap on the number of steps [19]. The technique is based on developing a decision tree that includes revised likelihood of reaching a decision within a certain number of steps, given that no decision is currently available. The work also shows how to derive a cap on the number of steps when it is not given as part of the problem definition.

What is common to these *deliberation scheduling* techniques is a formal approach to meta-reasoning based on probabilistic information about alternative computations and the use of decision-theory to maximize the expected value of computation. The value of computation is an extension of the value of information, taking into account the cost of computation. The optimality of the meta-level control depends on assumptions that sometimes limit the sequences of computations considered. For example, the myopic meta-level control assumption selects the best single computational step as long as it has positive value and refrains from the complex evaluation of long sequences. In this paper, we use a relatively simple model of computation (based on reactivating a contract algorithm with a particular allocation), allowing us to solve the global optimization problem.

Our analytical solution to the contract sequencing problem in Section 3 was inspired by the work of Baeza-Yates, Culberson, and Rawlins on searching an unbounded region for an object [1]. The problem involves a "robot" whose goal is to find a distinguished object in the plane. For example, the object may be a point that in n steps away from the robot on a straight line, but the robot does not know the direction or the distance to the point. It is assumed that the robot can move one step at a time and that it cannot tell the distance to the point, but it can recognize it when it reaches the destination. The goal is to find an optimal search strategy that minimizes the distance traveled by the robot before it reaches the destination. Apparently, this problem is similar to the problem of sequencing a single contract algorithm. The analogy between the two problems

is not obvious, but once formalized, they present similar mathematical questions. In particular, Lemmas 1 and 3 in this paper provide a foundation to solving the problem of searching in the plane. Furthermore, the analytical work reported here offers a dramatic simplification of the proofs presented in [1].

The contract sequencing policies developed in Sections 4 and 5 follow several other recent applications of dynamic programming to construct monitoring policies for managing computations. Such control policies improve over the previously used myopic approaches to control computations. The latter select the next *single* computational step with the highest value as long as its comprehensive value is positive. One of the early examples of using non-myopic control policies has been reported by Einav and Fehling [7]. They have developed an algorithm to construct generate-and-test policies to solve a problem with several given non-interruptible methods. Unlike contract algorithms, these methods do not offer a tradeoff between computation time and quality. Another early example is the technique developed by Russell, Subramanian, and Parr for sequencing a set of rules given a stochastic deadline [16]. Their dynamic programming approach provides a globally optimal schedule. Hansen and Zilberstein have developed similar policies to control interruptible anytime algorithms [8]. More recently, the approach has been applied successfully to control a complex *progressive processing* task structure [22]. These examples demonstrate that MDPs can be used effectively to handle uncertainty in computation and develop optimal meta-level control policies.

7. Conclusion

We have analyzed the problem of optimal sequencing of contract algorithms. The problem arises when there is uncertainty about the amount of time available for problem-solving with contract algorithms. When no prior information is available about the deadline, the best sequence can match the performance of the contract algorithm when it runs on a processor that is 4 times faster. No slower acceleration can guarantee that performance. This result is generalized to the case in which an interruptible system must solve multiple problem instances. When stochastic information is available about the deadline and about the performance of the contract algorithm, the optimal sequence of contracts can be constructed using dynamic programming. Finally, we solve the case in which no deadline is specified but the utility function is time-dependent.

There are several obvious extensions of this work that have not been discussed in the paper. The sequencing policies developed is Sections 4 and 5 are extremely general. They can be applied to different variations of the problem we consider. For example, there may be multiple problem instances to be solved, the utility function over the set of problems may be arbitrary (rather than minimal quality), and there may be a given prior distribution over the problem instances (indicating the likelihood a particular solution may be needed at the deadline).

Generalizing the analytical results of Section 3 is a harder task. When multiple problem instances are solved, it is interesting to find an optimal sequencing strategy to maximize the *average* quality rather than minimal quality. This turned out to be a much harder problem that remains open. Once average quality replaces the minimal quality, the equations we get are much harder to solve in closed form.

To summarize, we show how to optimally sequence contract algorithms so as to maximize their utility in interruptible domains. These results provide useful guidance for embedding contract algorithms in a wide range of practical applications. In addition, they show that a large body of work on planning under uncertainty (using Markov decision processes) is applicable to meta-level control of computation.

Acknowledgments

We thank Dan Bernstein, Anne Boyer, Dan Dooley, Eric Hansen, Tuomas Sandholm, René Schott and the anonymous reviewers for feedback and fruitful discussions of earlier versions of this manuscript.

Appendix A: Proof of Theorem 4

From Lemma 3, for any sequence of contracts $X = (x_1, x_2, ...)$, r must satisfy:

$$\forall i \geq 1: \quad Q_{\mathcal{A}}(\frac{x_1 + x_2 + ... + x_{i+m}}{m}) \leq Q_{\mathcal{A}}(rx_i)$$

From the strict monotonicity of $Q_{\mathcal{A}}$ we get:

$$\forall i \geq 1: \quad \sum_{j=1}^{i+m} x_j \leq mrx_i.$$

Setting $g_i = \sum_{j=1}^{i} x_j$, we can rewrite the previous equation as:

$$\forall i \geq 2: \quad g_{i+m} \leq mr(g_i - g_{i-1}). \tag{9}$$

We know that the sequence $(g_1, g_2, ...)$ is an increasing sequence of positive numbers. As before, let ρ be defined as:

$$\rho = \inf\{\frac{g_2}{g_1}, \frac{g_3}{g_2}, ..., \frac{g_{i+1}}{g_i}, ...\}.$$

It is clear that $\rho \geq 1$. From Equation 9 we obtain

$$\forall i \geq 2: \quad g_i \geq g_{i-1} + \frac{g_{i+m}}{mr}$$

or

$$\begin{aligned}
\frac{g_i}{g_{i-1}} &\geq 1 + \frac{1}{mr}\frac{g_{i+m}}{g_{i-1}} \\
&= 1 + \frac{1}{mr}\frac{g_i}{g_{i-1}}\frac{g_{i+1}}{g_i}\cdots\frac{g_{i+m}}{g_{i+m-1}} \\
&\geq 1 + \frac{\rho^{m+1}}{mr}.
\end{aligned}$$

Finally, we deduce that:

$$\rho \geq 1 + \frac{\rho^{m+1}}{mr}.$$

Equivalently, because $\rho > 1$, we get:

$$r \geq \frac{\rho^{m+1}}{m(\rho - 1)}$$

The function $\rho \longrightarrow \frac{\rho^{m+1}}{\rho-1}$ reaches its minimum on the interval $(1, +\infty)$ when $\rho = \frac{m+1}{m}$, therefore $r \geq (\frac{m+1}{m})^{m+1}$.

The ratio $(\frac{m+1}{m})^{m+1}$ can be obtained by a sequence of contracts defined by a geometric series with run-times being multiplied by a factor of $\frac{m+1}{m}$. In other words:

$$X = (1, \frac{m+1}{m}, ..., (\frac{m+1}{m})^i, ...).$$

Thus the best possible acceleration ration is $r = (\frac{m+1}{m})^{m+1}$. ■

References

[1] Ricardo Baeza-Yates, Joseph Culberson, and Gregory Rawlins. Searching in the plane. *Information and Computation*, 106:234–252, 1993.

[2] Daniel S. Bernstein and Shlomo Zilberstein. Optimal sequencing of contract algorithms using multiple processors. CS Technical Report 00-62, Computer Science Department, University of Massachusetts, Amherst, 2000.

[3] Mark Boddy and Thomas Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285.

[4] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. *Seventh National Conference on Artificial Intelligence*, 49–54, 1988.

[5] Jean-Michel Gallone and François Charpillet. Real-time scheduling with Neurosched. *13th International Conference on Tools with Artificial Intelligence*, 478–479, 1997.

[6] Alan Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1491–1502, 1993.

[7] David Einav and Michael R. Fehling. Computationally-optimal real-resource strategies. *IEEE International Conference on Systems, Man and Cybernetics*, 581–586, 1990.

[8] Eric A. Hansen and Shlomo Zilberstein. Monitoring the progress of anytime problem-solving. *Thirteenth National Conference on Artificial Intelligence*, 1229–1234, 1996.

[9] Eric A. Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. To appear in *Artificial Intelligence*, 2001.

[10] Eric Horvitz. Reasoning about Beliefs and Actions under Computational Resource Constraints. *Workshop on Uncertainty in Artificial Intelligence*, Seattle, WA, 1987.

[11] Eric Horvitz. Reasoning under varying and uncertain resource constraints. *Seventh National Conference on Artificial Intelligence*, 111–116, 1988.

[12] Eric Horvitz. *Computation and Action Under Bounded Resources.* Ph.D. dissertation, Stanford University, California, 1990.

[13] Eric Horvitz. Models of continual computation. *Fourteenth National Conference on Artificial Intelligence*, 286–293, 1997.

[14] Eric Horvitz and Shlomo Zilberstein, eds. Fall Symposium on Flexible Computation in Intelligent Systems. Technical Report FS-96-06, AAAI: Menlo Park, CA, 1996.

[15] Anita Pos. *Time-Constrained Model-Based Diagnosis.* Master Thesis, Department of Computer Science, University of Twente, The Netherlands, 1993.

[16] Stuart J. Russell, Devika Subramanian and Ron Parr. Provably bounded optimal agents. *Thirteenth International Joint Conference on Artificial Intelligence*, 338–344, 1993.

[17] Stuart J. Russell and Eric H. Wefald. Principles of metareasoning. *Artificial Intelligence*, 49:361–395, 1991.

[18] Stuart J. Russell and Shlomo Zilberstein. Composing real-time systems. *Twelfth International Joint Conference on Artificial Intelligence*, 212–217, 1991.

[19] Tuomas W. Sandholm and Victor R. Lesser. Utility-based termination of anytime algorithms. *ECAI Workshop on Decision Theory for DAI Applications*, 88–99, 1994.

[20] Shlomo Zilberstein. Optimizing decision quality with contract algorithms. *Fourteenth International Joint Conference on Artificial Intelligence*, 1576–1582, 1995.

[21] Shlomo Zilberstein and Stuart J. Russell. Optimal composition of real-time systems. *Artificial Intelligence* 82(1-2):181–213, 1996.

[22] Shlomo Zilberstein and Abdel-Illah Mouaddib. Reactive control of dynamic progressive

processing. *Sixteenth International Joint Conference on Artificial Intelligence*, 1268–1273, 1999.

[23] Shlomo Zilberstein, Abdel-Illah Mouaddib, and Andrew Arnt. Dynamic scheduling of progressive processing plans. *ECAI 2000 Workshop on Planning, Scheduling and Design*, 139–145, 2000.