This is a repository copy of *Functional Uncertainty in Real-Time Safety-Critical Systems*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/187712/

Version: Accepted Version

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Functional Uncertainty in Real-Time Safety-Critical Systems

Sanjoy Baruah
Washington University
USA
baruah@wustl.edu

Alan Burns
The University of York
UK
alan.burns@york.ac.uk

David Griffin
The University of York
UK
david.griffin@york.ac.uk

ABSTRACT

Safety-critical cyber-physical systems increasingly use components that are unable to provide deterministic guarantees of the correctness of their functional outputs; rather, they characterize each outcome of a computation with an associated "uncertainty" regarding its correctness. The problem of assuring correctness in such systems is considered. A model is proposed in which components are characterized by bounds on the degree of uncertainty under both worst-case and typical circumstances; the objective is to assure safety under all circumstances while optimizing for performance for typical circumstances. A problem of selecting components for execution in order to obtain a result of a certain minimum uncertainty as soon as possible, while guaranteeing to do so within a specified deadline, is considered. An optimal semi-adaptive algorithm for solving this problem is derived. The scalability of this algorithm is investigated via simulation experiments comparing this semi-adaptive scheme with a purely static approach.

1 INTRODUCTION

Many safety-critical cyber-physical systems (CPS's) are required to have their safety properties verified (in some domains, certified) prior to deployment. However there is an increasing use in these CPS's of "**Autonomous Agents**" (AAs)[1] developed using techniques that are currently not widely accepted in safety-critical systems implementation. The presence of such "non-pedigreed"[7, 8] components means that approaches that have traditionally been used for the purposes of performing safety assurance in safety-critical systems are not directly applicable to the verification of these CPS's.

It is widely acknowledged that *predictability* of run-time behavior [33] is very important for the purposes of assuring safety in safety-critical systems. Although most non-trivial safety-critical systems inevitably encounter some uncertainty

---

[1]The term Autonomous Agents is used here as an abstraction that includes, but is not restricted to, components based on popular machine learning approaches (including Learning-Enabled Components as defined in [26]).

during run-time, safety-critical systems designers have developed a range of advanced and sophisticated techniques for dealing with inherent run-time unpredictability with regards to extra-functional properties such as timing (the duration required to complete execution) or energy consumption. In addition to the problem of such extra-functional unpredictability, however, safety-critical *systems that use AAs tend to also not be predictable from the functional perspective*: the precise "worth" or accuracy of a computation performed by an AA is often not easily estimated beforehand; this is particularly true if the AA incorporates deep learning or similar AI-based techniques. The problems arising from such lack of functional determinism in AI-based techniques such as deep learning has been recognized within the machine learning community. It is thus typically required that an AA, for example an autonomous advisor or a classifier, must not only produce output but it must also produce a measure of the uncertainty (or confidence) that is associated with that output. E.g., Kendall et al. [20] state "*Uncertainty should be a natural part of any predictive system's output. Knowing the confidence with which we can trust the semantic segmentation output is important for decision making. For instance, a system on an autonomous vehicle may segment an object as a pedestrian. But it is desirable to know the model uncertainty…*" A number of techniques have been developed for obtaining an estimate of uncertainty, including Bayesian Neural Nets [12], probabilistic model checking [9] and Deep Ensembles [22]. However in the safety-critical domain, where very low levels of uncertainty are required, it is unlikely that a single AA even if presented with familiar input will be able to provide an output with sufficiently high confidence. And with unfamiliar input it is very likely that the outputs of multiple alternative AAs will need to be fused together in order to drive down the aggregate uncertainty to acceptable levels: as noted by Guo et al. [14] in the context of autonomous vehicles, "*If the detection network is not able to confidently predict the presence or absence of immediate obstructions, the car should rely more on the output of other sensors …*". It follows that in the safety-critical real-time domain, with the current level of uncertainly that AAs, particularly learning-based ones, are able to provide, a number of diverse components will be needed to drive down uncertainty levels low enough to satisfy safety requirements. In this research we consider how a family of such components should be scheduled so that a specified deadline for producing the output is satisfied and the required level of confidence achieved (i.e. the level of uncertainly is guaranteed to fall below a pre-defined threshold).

A motivating use-case. We propose an approach for dealing with functional run-time uncertainty for a particular form of computation involving AAs, that is motivated by the following use-case drawn from a related project on navigation

in autonomous mobile systems (AMS's) such as vehicles and robots:

- A moving AMS repeatedly checks that its designated path is free of hazards. Each check must be completed by a deadline.
- For each check, the AMS may apply one or several of a range of available AAs, some of which are learning-enabled, that take input from different sources (e.g., cameras, lasers, Li-DAR, radars, microphones, other vehicles, and even satellite feeds) to evaluate the designated path. An AA, when applied, outputs its assessment –either HAZARD or CLEAR– of the path ahead:
- If the assessment is HAZARD, then the AMS immediately takes safety action and comes to a stand-still as rapidly as it safely can.
- If the assessment is CLEAR then an *uncertainty estimate*, q ($0 \leq q \leq 1$), is also output. The effectiveness of each AA is dependent on the current conditions of the environment: road markings, proximity of other vehicles (AMSs and human controlled), obstacles close to the designated path, weather, ambient light etc.; the value of q reflects the confidence one should place upon the CLEAR designation from this AA in view of current conditions.
- The AMS applies AAs one at a time until the cumulative uncertainty falls below a pre-defined level $\mathbb{Q}$. When this happens the AMS concludes that its designated path is currently a safe one, and proceeds with additional optimizations (e.g., activating actuators that ensure a smoother –less jerky– ride); the sooner this happens, the "better" the performance of the AMS (and hence the sooner the cumulative uncertainty falls below $\mathbb{Q}$ the better). However if collective uncertainty does not fall below $\mathbb{Q}$ by the pre-determined deadline then the AMS flags a potential *safety violation* and hands control over to an emergency handler (which may, e.g., bring it to a stand-still as rapidly as safely feasible). This outcome is considered a failure for the autonomous advisory system.

Thus, any one of three possible outcomes: (i) HAZARD designation; (ii) CLEAR designation at an uncertainty level $\leq \mathbb{Q}$; or (iii) failure (an inability to achieve uncertainty level $\mathbb{Q}$ in the CLEAR designation); can happen each time the AMS performs its hazard check. We are tasked with determining the order in which the AAs should be applied in order to avoid the third outcome altogether; subject to this constraint, we would like to obtain the second outcome ASAP.

The problem considered. Generalizing from the use-case above, we assume that we have a number of components, each an abstract representation of some AA, available to us. The i'th component C_i takes some *duration* to execute, and we obtain an output with an estimate of the *uncertainty* associated with that output. We make the simplifying assumption that the execution duration of C_i does not vary much on different executions, and is *a priori* known.[2] However the exact level of uncertainty that is returned is unknown prior to actually executing the component, and will in general differ upon different executions of C_i. We assume that a pair of upper bounds are *a priori* known for this actual uncertainty level:

- a *worst-case* bound q_i, denoting that the component will produce output with uncertainty $\leq q_i$ under all circumstances; and
- a *typical-case* [31] bound q_i^T, denoting that the component will attain an uncertainty level $\leq q_i^T$ under all "typical" circumstances – i.e., all circumstances except perhaps some highly pathological ones that are extremely unlikely to occur in practice. Note $q_i \geq q_i^T$.

Given a collection of such components, a target uncertainty level $\mathbb{Q}$ that must be attained, and a deadline D, we seek to determine the order in which the components should be executed in order to ensure that the target uncertainty level is obtained by the deadline, provided each component obtains a level $\leq$ its worst-case bound when executed, and minimize the duration taken to obtain the target level if each component obtains a level $\leq$ its typical-case bound when executed. That is, we seek to *optimize for performance assuming typical circumstances, whilst avoiding failure under all circumstances*, typical or not. In this paper we develop an *optimal* semi-adaptive algorithm for solving this problem; we will see that the problem is inherently intractable and hence our algorithm will, in general, have running time exponential in the number of components n. We anticipate that n will not be too large in actual systems (this is certainly true for our motivating use-case), and we have performed extensive experimental evaluation to understand the scalability of our optimal algorithm as the number of available components n increases. Note that we assume that the components are executed one at a time, rather than all in parallel. This has two advantages. First, it saves resources by not executing components that turn out to not be needed. Many AMSs (for example self-driving cars) are mobile and thus not tethered to an energy source; hence engineering solutions that minimise resource usage are highly desirable. Typical behaviour may require only 50% of those utilised in the worst case (see Section 6). As the worst case is expected to be a very rare occurrence, to execute the larger set of all components is clearly wasteful. The other advantage comes from allowing components to benefit from the fact that other components have already executed. This may allow the delivered uncertainty level to be significantly better than the typical as well as the worst-case estimates. An example of this relationship are the components that make use of LiDAR and monocular RGB (Red, Green, Blue camera); both of which can benefit from the prior execution of the other [37].

In the following descriptions we focus on the notion of uncertainty (q); an equivalent formulation is one that uses a measure of confidence p. Both of these metrics are usually expressed as probabilities, with $p = 1 - q$.

Organization. The reminder of this paper is organized as follows. We briefly discuss some related work in Section 2. In Section 3 we formally define the problem that we are studying here. We will see that solving this problem requires us to determine which component is *safe* to execute at some system state — in Section 4 we show how such safe choices can be identified. This is used in Section 5 to solve the general problem in a way that is efficient in terms of run-time complexity.

---

[2] This simplifying assumption removes timing uncertainty from the picture and so allows us to highlight the primary focus of this paper, which is that of dealing with the *functional* uncertainty that characterizes the run-time behavior of many AAs.

We have conducted extensive simulations upon synthetically-generated workloads in order to evaluate the performance of our algorithm: we report on these in Section 6. We conclude in Section 7 by placing this work within a larger perspective on the design and analysis of complex safety-critical cyber-physical systems.

2 RELATED WORK

The importance, and the enormous complexity, of obtaining assurance for safety-critical CPS's that incorporate machine learning has been widely recognized, and several large-scale initiatives aimed at solving this problem have been launched. Some examples include the *Assured Autonomy* Program [26] of the United States Defense Advanced Research Projects Agency (DARPA); the *Assuring Autonomy International Programme* [40] funded by the international insurance company Lloyd's of London at the University of York (UK); and the *Bounded Behavior Assurance* initiative [23], spearheaded by the major US defense contractor Northrop Grumman Corporation.

Our work here builds on some recent research [3] which proposed a component model in that each component is assumed to take a certain duration to execute, and to return a value that can only be determined once the component has completed execution, but for which worst-case and typical-case bounds are *a priori* known. However, [3] deals with a model of additive value that is accumulated by executing multiple components, whereas we are concerned here with minimizing uncertainty/confidence, which is inherently multiplicative rather than additive. Furthermore, [3] applies its model to multi-stage computations where there is a choice of components for each stage, and the issue is one of choosing a component for execution at each stage. In contrast, the problem we consider here has the added dimension that the order of execution of components is not fixed but must be determined by the resource allocation and scheduling mechanism.

In Section 3, we will define *static* and *semi-adaptive* variants of our problem depending on how much of the selection of components is determined prior to run-time; we note that a similar distinction between static and adaptive strategies was made in the context of a graph routing problem in [2, 4].

We emphasize that while there is a tremendous amount of ongoing research in the design of AAs for provable correctness (e.g., [15, 17, 32, 35, 36]) and, more specifically, the tailoring (via focusing on predictability, architectures, scheduling and WCET estimation) of the application and training of DNNs (Deep Neural Nets) in real-time systems (e.g., [11, 16, 18, 19, 24, 25, 27, 38, 39, 42]), we look upon such work as orthogonal to the research reported here: we adopt a "black-box" approach to AAs, whereby they are modelled entirely by their execution durations and the self-reported uncertainty of the results they compute, and the *a priori* estimates of these parameters.

A common form of AA is a classifier [1, 28, 29, 41]. Here an input is identified as being a member of one (or more) of a set of classes; each estimation having an associated probability. So, for example, the probability of input i being a member of $S1$ is $p1$; of $S2$, $p2$ etc.. If none of the membership probabilities

is greater than some threshold then the input must be deemed to be unclassified. This formulation is used by Wang et al. [35] to explore the use of IDK classifiers [21, 34]: if a "base"classifier is unable to produce an output with sufficient confidence the IDK classifier declares "I Don't Know". We have recently [5, 6] begun the consideration of IDK classifiers from a real-time perspective; the framework developed in the current paper is a further generalisation. A number of components are available, their order of execution must be determined and the level of uncertainty is equivalent, in our use case, to not knowing that the route is clear. Our scheme is also applicable to a broader set of machine-learning algorithms.

3 MODEL AND PROBLEM STATEMENT

In this section we provide a formal definition of the problem that was illustrated via a motivating use-case in Section 1 above. We consider various aspects of this problem: the manner in which outcomes of executing different components are combined together, as well as restrictions that are placed upon the component-selection strategies (including whether each component may be executed multiple times or not). We will see that different combinations of factors from these different aspects give rise to different specific problems.

We suppose that n different components $C_1, C_2, \ldots, C_n$, are available. Component C_i is characterized by the 3-tuple (d_i, q_i, q_i^T), where

- d_i denotes its execution duration (assumed to be fixed – see footnote 2);
- q_i is an *a priori* upper bound on the (unknown) actual uncertainty that is guaranteed to be obtained by executing C_i under all circumstances; and
- q_i^T is an *a priori* upper bound on the actual uncertainty that is guaranteed to be obtained by executing C_i under all *typical* [31] (i.e., non-pathological) circumstances.

A target acceptable uncertainty $\mathbb{Q}$ that must be obtained, and a deadline D within which this must happen, are also specified. That is, an ***instance*** of our problem is specified as follows:

$$\left\langle \left\{ C_i = (d_i, q_i, q_i^T) \right\}_{i=1}^{n}, \mathbb{Q}, D \right\rangle \tag{1}$$

During each execution of this instance, we are to execute components one at a time until the composite uncertainty becomes $\leq \mathbb{Q}$.

```
1   Q = 1 // Current net uncertainty; initialized prior to execution
2   repeat
3       choose a component, and execute it
4       Q = f(Q, q̂), where q̂ is the actual value obtained
        // Function f( , ) is described in the text
5   until (Q ≤ Q)
```

Figure 1: A Template of the Run-time Strategy

Correct and Typical Behaviors. As stated earlier, the exact uncertainty of the result obtained by executing a component is unknown prior to actually executing it, and the same component may exhibit different uncertainties upon different executions. The instance is said to exhibit *correct behavior* during an execution if each executed component C_i exhibits

an uncertainty no greater than its value parameter q_i; it is additionally said to exhibit *typical behavior* if each executed component exhibits an uncertainty no greater than its typical-value parameter q_i^T.

Scheduling goal. Components are chosen for execution as shown in Fig. 1. In so doing, our safety constraint is that the **repeat-until** loop must be exited within an interval of duration D upon all correct behaviors of the instance. Subject to satisfying this safety constraint, our optimization objective is to minimize the duration of execution of the loop during all typical behaviors of the instance.

Composition of uncertainties. In Line 4 of the pseudo-code of Fig. 1 above, we update the uncertainty Q based upon the actual uncertainty returned upon executing the last component that was executed, and the uncertainty that had accumulated prior to then. This is represented in the pseudo-code as $f(Q, \hat{q})$, where $\hat{q}$ denotes the value obtained by actually executing a component. Although the exact form of this function f will, in general, depend upon application characteristics, for the sake of concreteness we assume in the remainder of this presentation that this composition is *multiplicative*: $f(Q, \hat{q}) = Q \times \hat{q}$. In addition to being applicable to such multiplicative composition of uncertainties, our results hold for any system in which the uncertainties exhibited by executing components are drawn from a *totally ordered Abelian group*.[3] Consider, e.g., our motivating use-case from Section 1, and suppose that the confidence levels —the p parameters— represent probabilities of success and that the different components are independent. Let p_i be the actual confidence that is output upon executing the i'th component to be executed. After k components have been executed, the probability that their outputs are all incorrect is $\Pi_{i=1}^{k}(1 - p_i)$; hence, the probability of the output – CLEAR – being correct is $\left(1 - \left(\Pi_{i=1}^{k}(1 - p_i)\right)\right)$. It may be verified that we can have Q equal this expression after k iterations if we initialize Q to zero in Line 1, and update it as follows in Line 4: $Q = 1 - (1 - Q) \times (1 - p_i)$.

EXAMPLE 1. Throughout this manuscript we will use the example instance depicted in Figure 2 as an illustrative example.

There are three components in this instance, with execution durations 2, 3, and 4 respectively, guaranteeing to return results that have uncertainty no greater than 10^{-3}, 10^{-4}, and 10^{-5} respectively in all correct behaviors, and no greater than 10^{-4}, 10^{-5}, and 10^{-6} respectively in all typical behaviors. □

Component re-use. Application characteristics will determine whether each individual component may be executed

---

[3]See, any algebra text or, e.g., https://encyclopediaofmath.org/wiki/Totally_ordered_group (accessed February 17th, 2021) for a precise definition.

C_i	d_i	q_i	q_i^T
C_1	2	10^{-3}	10^{-4}
C_2	3	10^{-4}	10^{-5}
C_3	4	10^{-5}	10^{-6}

Figure 2: An example collection of components $\{C_1, C_2, C_3\}$

at most once, or multiple times, during a single execution of the instance. Our application domains of interest generally do not allow for re-use; accordingly we will restrict our attention here to the case where such re-use is forbidden. (We point out that forbidding re-use is the more difficult problem from an algorithmic perspective in the sense that relatively straightforward dynamic programming strategies can be applied to solve the variants with re-use permitted.)

Static and adaptive strategies. We distinguish between static and adaptive run-time strategies depending upon the manner in which the choice of components is made in Line 3 of Figure 1. A *static* strategy determines the order of execution prior to run-time, before the first one is executed; in contrast, only the first component is chosen prior to run-time by an *adaptive* strategy, and the actual uncertainty with which the result was returned upon executing a component is taken into account in choosing the next component to execute.

Not surprisingly, adaptive strategies are generally able to guarantee superior performance (i.e., shorter execution durations for typical behaviors); we illustrate below on our running example instance from Figure 2:

EXAMPLE 2. Consider the example collection of three components shown in Figure 2, with deadline $D = 8$ and target value $\mathbb{Q} = 10^{-9}$. Since $d_1 + d_2 + d_3 = 9$ while the available duration is 8, all three components cannot be executed.

A static schedule can only guarantee an overall uncertainty of $10^{-4} \times 10^{-5} = \mathbf{10^{-9}}$, by choosing components C_2 and C_3. Under a typical behavior in which each component obtains a value equal to its q_i^T parameter, we would need to execute both components in order to drive the uncertainty down to $\leq \mathbb{Q}$ (i.e., 10^{-9}); hence the schedule duration in a typical behavior is $d_2 + d_3 = 7$.

Now consider the following adaptive strategy:

    execute C_3
    if actual uncertainty returned by this execution is $\leq 10^{-6}$
        then execute C_1
        (for a guaranteed uncertainty $\leq 10^{-6} \times q_i = 10^{-6} \times 10^{-3} = \mathbf{10^{-9}}$)
        else execute C_2
        (for a guaranteed uncertainty $\leq q_3 \times q_i = 10^{-5} \times 10^{-4} = \mathbf{10^{-9}}$)

This strategy is safe since the maximum duration is $d_3 + \max(d_1, d_2) = 4 + \max(2, 3) = 7$. In a typical behavior, note that C_3 would obtain a value $\leq 10^{-6}$ and hence C_1 would be executed next, for a duration of $d_3 + d_1 = \mathbf{6}$. □

The following lemma asserts that adaptive strategies may outperform static ones by an arbitrarily large degree:

LEMMA 1. *Static strategies may have arbitrarily poor performance when compared to adaptive ones.*

Proof. Consider an instance comprising the following three components:

C_i	d_i	q_i	q_i^T
C_1	$D - 1$	10^{-4}	10^{-4}
C_2	1	10^{-2}	10^{-4}
C_3	1	10^{-2}	10^{-4}

with deadline D, and a target acceptable uncertainty $\mathbb{Q} = 10^{-6}$.

An adaptive schedule may execute one of C_2 or C_3. If the uncertainty in the result of this execution $\leq 10^{-4}$, then the other one of these is executed; else, C_1 is executed. Since q_2^T

and q_3^T are both 10^{-4}, the schedule duration in the typical-case is $1 + 1 = 2$.

A static schedule, on the other hand must include C_1 and one of C_2 or C_3 in order to guarantee an overall uncertainty $\mathbb{Q} = 10^{-6}$ under all correct behaviors. Hence even in typical-case behaviors its schedule duration is as large as D. The ratio of the typical-case schedule durations achievable by static versus adaptive strategies is thus $D/2$, which $\rightarrow \infty$ as $D \rightarrow \infty$. □

By demonstrating that adaptive strategies can perform arbitrarily better than static ones, Lemma 1 above argues in favor of considering the use of adaptive strategies rather than static ones. The remainder of this paper is devoted to the exploration of such an adaptive strategy: in Section 4 below we describe some pre-processing that enables us to very efficiently identify which components it is *safe* to execute at any given system state, and in Section 5 we describe how to optimize over all such safe components in order to choose one that minimizes typical response time (i.e., the duration needed to reduce the uncertainty to below the specified amount under all typical circumstances).

4 IDENTIFYING SAFE CHOICES

As discussed in Section 3 above, during run-time our algorithm repeatedly (Line 3, Figure 1) selects a component from amongst the ones that are eligible for execution in order to minimize the duration by which the overall uncertainty falls below the acceptable level, $\mathbb{Q}$, in all typical instance behaviors. Since the selection must also meet the safety constraint that the duration will not exceed the specified deadline in all correct (even if not typical) behaviors, the selected component must be one that is able to guarantee safety even if it (and some or all subsequent components that are executed) return[s] results with uncertainties that are as large as their respective maximum values – their q_i parameters. In this section we describe some pre-processing that is done by our algorithm in order that the set of such components eligible for selection may be identified efficiently. This algorithm is used as a subroutine in Section 5 to design an algorithm that optimizes performance while respecting this safety constraint.

For a given set S of components, a target uncertainty q, and a duration d, the *set of safe choices*, denoted $\underline{\text{safe}(S, d, q)}$, is the set of all the components that it is "safe" to execute in the sense that if it is executed an overall uncertainty $\leq q$ can be guaranteed within a duration d under all correct (but not necessarily typical) behaviors. Observe that determining these sets of safe choices is computationally intractable: it can be shown by a simple polynomial-time reduction to the PRODUCT KNAPSACK problem [10, 30] that it is an NP-hard problem to even determine whether such a set is empty or not, and we should therefore not expect to be able to solve it in polynomial time.[4]

Let S denote any subset of the set of components, and let d be any non-negative integer. Let $M(S, d)$ denote *the minimum*

---

[4]Note that it therefore follows that our overall problem is computationally intractable: given the set of all components, it is NP-hard to determine whether the safety constraint can be satisfied.

SafeSequence(S, d)

// Returns the sequence of components $\in S$ whose execution guarantees an uncertainty $M(S, d)$ within an interval of duration d.

1 **if** $(M(S, d) \geq 1.0)$ **return** $\langle \ \rangle$ // The empty sequence
2 $seq = \langle \ \rangle$ // Inititalise to the empty sequence
3 **repeat**
4   Suppose that the component $M_C(S, d)$ is C_k
5   append C_k to seq
6      $S = S \setminus \{C_k\}$
7      $d = d - d_k$
8 **until** $(M(S, d) \geq 1.0)$
9 **return** seq

Figure 3: Identifying the sequence of components in S whose execution guarantees a value $M(S, d)$ within duration d.

uncertainty that can be guaranteed over an interval of duration d, given the set S of components, in all correct behaviors of the instance. Below we write a recurrence relation defining values of $M(S, d)$ in terms of $M(S', d')$ where $S' \subsetneq S$ and $d' < d$. We also define an auxiliary function $M_C(S, d)$, denoting *the first component we should execute* in order to achieve this minimum uncertainty; in Fig. 3 we will use $M_C(S, d)$ to reconstruct the subset of components that guarantees this smallest uncertainty.

Base Case: S is a singleton set.

$$M(\{C_i\}, d) = \begin{cases} 1.0, & d < d_i \\ q_i, & d \geq d_i \end{cases} \qquad (2)$$

and

$$M_C(\{C_i\}, d) = \begin{cases} -, & d < d_i \\ C_i, & d \geq d_i \end{cases} \qquad (3)$$

Recursive case:
$M(S, d) = 1.0$ and $M_C(S, d) = -$ if $\left(d < \min_{C_i \in S}\{d_i\}\right)$; otherwise,

$$M(S, d) = \min_{(C_i \in S) \wedge (d \geq d_i)} \left\{ q_i \times M\left(S \setminus \{C_i\}, d - d_i\right) \right\} \quad (4)$$

$$M_C(S, d) = \text{The } C_i \text{ that minimizes the RHS above} \quad (5)$$

EXAMPLE 3. The values of $M(S, d)$ and $M_C(S, d)$ for our three-component running example (Figure 2) are provided in tabular form in Figure 4, for all non-empty subsets $S \subseteq \{C_1, C_2, C_3\}$ and all $d \leq d_1 + d_2 + d_3 = 9$. Each row corresponds to a non-empty subset of the set of components, and each column to a value of d. Each entry in this table is in the form of an ordered pair: the first element in the ordered pair in the d'th column of the row labeled S is $M(S, d)$, and the second element is the index of the component $M_C(S, d)$. Hence for example the entry in the last column of the first row states that an uncertainty as low as 10^{-12} can be guaranteed by the entire set of three components over an interval of duration 9, and that executing component C_3 first would enable us to achieve this guarantee. In a similar vein, we see that the set of components $\{C_1, C_2\}$, over an interval of duration 7, can

$S(\downarrow) / d(\rightarrow)$	0 – 1	2	3	4	5	6	7	8	9
$\{C_1, C_2, C_3\}$	(1, -)	$(10^{-3}, 1)$	$(10^{-4}, 2)$	$(10^{-5}, 3)$	$(10^{-7}, 2)$	$(10^{-8}, 2)$	$(10^{-9}, 3)$	$(10^{-9}, 3)$	$(10^{-12}, 3)$
$\{C_2, C_3\}$	(1, -)	(1, -)	$(10^{-4}, 2)$	$(10^{-5}, 3)$	$(10^{-5}, 3)$	$(10^{-5}, 3)$	$(10^{-9}, 3)$	$(10^{-9}, 3)$	$(10^{-9}, 3)$
$\{C_1, C_3\}$	(1, -)	$(10^{-3}, 1)$	$(10^{-3}, 1)$	$(10^{-5}, 3)$	$(10^{-5}, 3)$	$(10^{-8}, 3)$	$10^{-8}, 3)$	$(10^{-8}, 3)$	$(10^{-8}, 3)$
$\{C_1, C_2\}$	(1, -)	$(10^{-3}, 1)$	$(10^{-4}, 2)$	$(10^{-4}, 2)$	$(10^{-7}, 2)$	$(10^{-7}, 2)$	$(10^{-7}, 2)$	$(10^{-7}, 2)$	$(10^{-7}, 2)$
$\{C_3\}$	(1, -)	(1, -)	(1, -)	$(10^{-5}, 3)$	$(10^{-5}, 3)$	$(10^{-5}, 3)$	$(10^{-5}, 3)$	$(10^{-5}, 3)$	$(10^{-5}, 3)$
$\{C_2\}$	(1, -)	(1, -)	(1, -)	$(10^{-4}, 2)$	$(10^{-4}, 2)$	$(10^{-4}, 2)$	$(10^{-4}, 2)$	$(10^{-4}, 2)$	$(10^{-4}, 2)$
$\{C_1\}$	(1, -)	$(10^{-3}, 1)$	$(10^{-3}, 1)$	$(10^{-3}, 1)$	$(10^{-3}, 1)$	$(10^{-3}, 1)$	$(10^{-3}, 1)$	$(10^{-3}, 1)$	$(10^{-3}, 1)$

Figure 4: The functions $M(S, d)$ and $M_C(S, d)$, described in Section 4, for the example instance of Figure 2. Each row corresponds to a non-empty subset of components; each column to a possible value of d. The ordered pair in the row labeled S and column labeled d denotes $\big(M(s, d), M_C(S, d)\big)$. (The first two columns have been compressed into a single column, since they are identical across the two columns for all rows.)

guarantee an uncertainty as low as 10^{-7}, and that executing component C_2 would enable us to achieve this guarantee.[5] □

Computational Complexity. Expressions 2–5 are in the form of standard recurrences; for a given instance $\langle\{C_1, C_2, \ldots, C_n\}, \mathbb{Q}, D\rangle$, a table of the form depicted in Figure 4 can be generated in $\Theta\left(2^n \times D\right)$ time via a standard bottom-up dynamic programming implementation of these recurrences. We believe it reasonable to expect the value of D to not be particularly large in real-time systems; the dependence on 2^n indicates that the running time of the algorithm scales exponentially with the number of available components. Our experimental evaluations (Section 6) indicate that this is unlikely to prove a limitation in practice: even upon modestly-equipped general-purpose laptop computers, we are comfortably able to handle up to 30 or so components in very reasonable amounts of time.

Reconstructing the sequence of components: The procedure SafeSequence(S, d) listed in Figure 3 applies standard dynamic programming techniques to reconstruct the sequence of components that guarantee a value $M(S, d)$ for a set of components S and duration d, in time linear in the number of components in the sequence; its use is illustrated in Example 4 below.

Example 4. Suppose that we have components $\{C_1, C_2\}$ available, and an interval of duration 4. Procedure SafeSequence (Figure 3) is called with $S = \{C_1, C_2\}$ and $d = 4$. The column labeled 4 of the row labeled $\{C_1, C_2\}$ in Figure 4 contains the entry $(10^{-4}, 2)$; the second item in this ordered pair denotes that $M_C(\{C_1, C_2\}, 4) = C_2$. Procedure SafeSequence therefore appends C_2 to the initially empty seq, and repeats the loop with $S = \{C_1\}$ and $d = 4 - d_2 = 4 - 3 = 1$. Since $M(\{C_1\}, 1)$ equals one (the first column –the one labeled "0 – 1"– of the last row), SafeSequence exits the loop, returning the sequence $\langle C_2 \rangle$ of components.

As another example, suppose that all three components are available, and the interval duration is 8. Procedure SafeSequence (Figure 3) is called with $S = \{C_1, C_2, C_3\}$ and $d = 8$. The second-last column –the one labeled 8– of the first row in Figure 4 contains the entry $(10^{-9}, 3)$, indicating that $M_C(\{C_1, C_2, C_3\}, 8) = C_3$. The procedure SafeSequence therefore appends C_3 to the initially empty seq, and repeats the

loop with $S = \{C_1, C_2\}$ and $d = 8 - d_3 = 4$. Since the entry in the column labeled 4 of the row corresponding to the subset $\{C_1, C_2\}$ in Figure 4 contains the entry $(10^{-4}, 2)$, procedure SafeSequence concludes that $M_C(\{C_1, C_2\}, 4) = C_2$ and appends C_2 to seq. Now $S = \{C_1\}$ and $d = 4 - d_2 = 1$; since $M(\{C_1\}, 1)$ equals zero (the first column –the one labeled "0 – 1" of the last row), SafeSequence exits the loop and returns the sequence $\langle C_3, C_2 \rangle$ of components. It may be verified that their cumulative duration is ≤ 8 ($d_3 + d_2 = 7$), and their cumulative worst-case value-guarantees is indeed 10^{-9} ($q_3 \times q_2 = 10^{-5} \times 10^{-4} = \mathbf{10^{-9}}$).

For a final example, let us compute SafeSequence$(C_2, 2)$. The entry in Figure 4 in the row labeled $\{C_2\}$ and column labeled 2 is $(1, -)$; i.e., $M(\{C_2\}, 2) = 1.0$ (in other words, we cannot reduce the uncertainty at all over an interval of size two with only component C_2 – not surprising, since $d_2 > 2$). Hence by Line 1 of the pseudo-code of Figure 3, the empty sequence $\langle\ \rangle$ is returned.   □

Computing safe(S, d, q). We now describe how safe(S, d, q), the set of components that may be executed without compromising safety is computed when given the components in S, an interval of duration d, and a target maximum uncertainty q. It is safe to execute component C_i when an uncertainty $\le q$ must be guaranteed over an interval of duration d, if and only if, in the event of this execution achieving an uncertainty that is as poor as its worst-case guarantee of q_i, it remains possible to achieve the remaining uncertainty needed, $(q \div q_i)$, within the remaining interval, of duration $(d - d_i)$, from the remaining components, i.e., $(S \setminus \{C_i\})$:

$$\text{safe}(S, d, q) = \bigcup_{C_i \in S} \left\{C_i \mid M\big((S \setminus \{C_i\}), (d - d_i)\big) \le (q \div q_i)\right\}$$
(6)

Once the values of $M(S, d)$ have been pre-computed for all S and d, observe that determining safe(S, d, q) is an efficient operation that can be performed in time linear in the number of components in S: simply check, for each $C_i \in S$, whether $M\big((S \setminus \{C_i\}), (d - d_i)\big) \le (q \div q_i)$.

Example 5. We illustrate the use of Expression 6 on the collection of components depicted in Figure 2, for a couple of example values of S, d, and q.

(1) safe$(\{C_1, C_2, C_3\}, 8, 10^{-9})$:
    For each C_i, we need to determine whether $M(\{C_1, C_2, C_3\} \setminus \{C_i\}, 8 - d_i,)$ is $\le 10^{-9} \div q_i$. For instance for $C_i \leftarrow C_1$, this

---

[5]We will see below, in Example 4, how this latter piece of information may be used to reconstruct the set of components that guarantees an uncertainty of 10^{-7} over an interval of duration 7.

checks whether

$$M(\{C_2, C_3\}, 8 - 2) \leq 10^{-9} \div 10^{-3}$$

$$\equiv \quad M(\{C_2, C_3\}, 6) \leq 10^{-6}$$

$$\equiv \quad 10^{-5} \leq 10^{-6} \text{ (From Figure 4 – 2nd row, column labeled 6)}$$

which is **false**. Hence $C_1 \notin \text{safe}(\{C_1, C_2, C_3\}, 8, 10^{-9})$. In a similar vein, we can verify

- C_2: Is $M(\{C_1, C_3\}, 8-3) \leq 10^{-9} \div 10^{-4}$? I.e., is $M(\{C_1, C_3\}, 5) \leq 10^{-5}$? – Yes
- C_3: Is $M(\{C_1, C_2\}, 8-4) \leq 10^{-9} \div 10^{-5}$? I.e., is $M(\{C_1, C_2\}, 4) \leq 10^{-4}$? – Yes

And hence $\text{safe}(\{C_1, C_2, C_3\}, 8, 10^{-9}) = \{C_2, C_3\}$

(2) Another example: $\text{safe}(\{C_1, C_2\}, 4, 10^{-3}) = \{C_1, C_2\}$ since
- C_1: Is $M(\{C_2\}, 4-2) \leq 10^{-3} \div 10^{-3}$? I.e., is $M(\{C_1\}, 2) \leq 1$? – Yes (trivially)
- C_2: Is $M(\{C_1\}, 4-3) \leq 10^{-3} \div 10^{-4}$? I.e., is $M(\{C_1\}, 1) \leq 10$?– Yes (again, trivially)

(3) A final example: $\text{safe}(\{C_1, C_3\}, 5, 10^{-4}) = \{C_3\}$ since
- C_1: Is $M(\{C_3\}, 5-2) \leq 10^{-4} \div 10^{-3}$? I.e., is $M(\{C_3\}, 3) \leq 10^{-1}$? – No
- C_3: Is $M(\{C_1\}, 5-4) \leq 10^{-4} \div 10^{-5}$? I.e., is $M(\{C_1\}, 1) \leq 10$?– Yes, trivially

□

5 AN OPTIMAL SEMI-ADAPTIVE STRATEGY

Recall that adaptive strategies choose components for execution one at a time during run-time: the actual uncertainties at which results are returned by executing components play a role in determining which component to execute next. We have seen (Lemma 1) that adaptivity is desirable since non-adaptive (i.e., static) strategies may perform arbitrarily poorly when compared to adaptive ones. However, one potential drawback of adaptive strategies is the computational complexity of each selection step during run-time; indeed, the following is a correct optimal adaptive strategy: (i) determine the first component to execute; (ii) execute this component and note the uncertainty under which it returns its result; and (iii) recurse on the remaining components for the target uncertainty divided by this obtained uncertainty, and a duration equal to the original duration minus the execution duration of the component that was executed. Notice that such a naïve approach would require us to solve the NP-hard problem identified in Section 3 each time a component is selected during run-time: we expect that this is not realistic for all but the smallest problem instances. In this section we discuss a somewhat restricted form of adaptivity called *semi-adaptivity* [2], for which the pre-processing is more efficient than for general adaptivity. A semi-adaptive strategy is a restricted form of an adaptive strategy, that is of the following kind:

- An *initial sequence* of components is determined prior to run-time. This initial sequence specifies the order in which components should be executed in all typical behaviors.

- An *alternative sequence* of components is defined, also prior to run-time, for each component specified in the initial sequence[6]. If a component in the initial sequence fails to return its result with an associated uncertainty no smaller than its typical uncertainty parameter (its q^T) upon being executed, the original sequence is abandoned henceforth and the alternative sequence defined for this particular component is executed instead. The alternative sequence is then followed; the original sequence is no longer utilised.

EXAMPLE 6. *The adaptive strategy discussed in Example 2 can in fact be looked upon as a semi-adaptive one: the initial sequence is $\langle C_3, C_1 \rangle$, and alternative sequence $\langle C_2 \rangle$ is associated with the first component of this original sequence.*

(This example also illustrates why there may sometimes be no need for specifying an alternative sequence for the last component of the initial sequence: even the worst-case minimum uncertainty for this last component suffices to satisfy the safety constraint.) □

Once the initial and alternative sequences have been determined prior to run-time, we point out that choosing the next component during run-time becomes a $\Theta(1)$ (i.e., *constant-time*) operation: there is a choice of at most two components to execute next depending upon the actual value obtained by the just-executed component.

We now derive an algorithm for synthesizing the initial sequence and the alternative sequences. Let $G(S, d, q)$ denote *the minimum duration over which we can guarantee to achieve an uncertainty no larger than q under typical circumstances, whilst simultaneously guaranteeing to achieve an uncertainty no larger than q over an interval of duration $d \geq 0$ under all circumstances*. Example 7 illustrates this notation.

EXAMPLE 7. *As stated in Example 6 above, the scheduling strategy described in Example 2 is a semi-adaptive one which guarantees an uncertainty no larger than 10^{-9} over an interval of duration 8 using the three components $\{C_1, C_2, C_3\}$ under all circumstances, and a duration 6 under all typical circumstances. It may additionally be verified that no other semi-adaptive strategy that guarantees an uncertainty no larger than 10^{-9} over an interval of duration 8 under all circumstances can guarantee a duration smaller than 6 under all typical circumstances. In the notation introduced above, this can be represented as*

$$G(\{C_1, C_2, C_3\}, 8, 10^{-9}) = 6$$

□

We will write a recurrence relation defining values of $G(S, d, q)$ in terms of $G(S', d', q')$ where $S' \subsetneq S$, $d' < d$, and $q' > q$. Analogously with the case of minimum uncertainty above (Equations 2-5), we will also define an auxiliary function $G_C(S, d)$ that we will use to reconstruct a sequence that achieves this minimum duration.

---

[6]Other than perhaps the last component in the initial sequence – the example below illustrates why this alternative sequence is not always needed for the last component.

<u>Base case</u>: S a singleton set.

$$G(\{C_i\}, d, q) = \begin{cases} 0 & \text{if } q \geq 1 \\ d_i, & \text{if } 1 > q \geq q_i \text{ and } d \geq d_i \\ \infty & \text{otherwise} \end{cases} \quad (7)$$

and

$$G_C(\{C_i\}, d, q) = \begin{cases} C_i, & \text{if } 1 > q \geq q_i \text{ and } d \geq d_i \\ -, & \text{otherwise} \end{cases} \quad (8)$$

<u>Recursive case</u>:

$$G(S, d, q) = \min_{C_i \in \text{safe}(S, d, q)} \left\{ d_i + G\left(S \setminus \{C_i\}, d - d_i, \min\left((q \div q_i^T), 1\right)\right) \right\} \quad (9)$$

$$G_C(S, d, q) = \text{ The } C_i \text{ that minimizes the RHS above} \quad (10)$$

EXAMPLE 8. We illustrate the computation of $G(S, d, q)$ and $G_C(S, d, q)$, by computing these functions for S comprising the three components of Figure 2, $d = 8$, and $q = 10^{-9}$. The recursion graph for this computation is depicted in Figure 5. The numbers **1–6** labeling the nodes represent a possible order in which the recursive calls are made, assuming a standard top-down recursive implementation.

Let us start with the initial call, at the root node. We have already seen (Example 5) that safe$(\{C_1, C_2, C_3\}, 8, 10^{-9}) = \{C_3, C_2\}$; hence, only two recursive calls are made with C_3 and C_2 being the potential components executed first. In Example 5 we have also seen (by explicitly computing them), that safe$(\{C_1, C_2\}, 4, 10^{-3}) = \{C_1, C_2\}$ and safe$(\{C_1, C_3\}, 5, 10^{-4}) = \{C_3\}$ — these explain the nodes that are explored in the recursion graph of Figure 5.

It is evident that the path in this graph that is highlighted in blue has the minimum duration ($d_3 + d_1 = 4 + 2 = 6$). This path bears witness to the fact that $G(\{C_1, C_2, C_3\}, 8, 10^{-9}) = 6$ (as stated in Example 7 above), and $G_C(\{C_1, C_2, C_3\}, 8, 10^{-9}) = C_3$. □

Synthesizing the Sequences. Having computed the functions $G()$ and $G_C()$ defined above, we can use them to generate the initial sequence, and the SAFESEQUENCE procedure of Figure 3 that was derived in Section 4 to generate the alternative sequences. The manner in which we do so is represented in pseudo-code form in Figure 6 — given an instance $\langle \{C_i = (d_i, q_i, q_i^T)\}_{i=1}^n, \mathbb{Q}, D \rangle$, a call to GENERATESEQUENCES$(\{C_1, C_2, \ldots, C_n\}, D, \mathbb{Q})$ returns the initial sequence, and sets the variable A_i to be the alternative sequence for the i'th component of the initial sequence, $i = 1, 2, \ldots$. The pseudo-code for this is pretty self-explanatory; we illustrate its use in Example 9 below.

EXAMPLE 9. Let us revisit the situation discussed in Example 8 above, after $G(\{C_1, C_2, C_3\}, 8, 10^{-9})$ and $G_C(\{C_1, C_2, C_3\}, 8, 10^{-9})$ have been computed — i.e., after all the recursive calls of the recursion graph depicted in Figure 5 have completed. We explain how the initial sequence and alternative sequences are computed for this example, by tracing the execution of the pseudo-code in Figure 6.

- Since $G_C(\{C_1, C_2, C_3\}, 8, 10^{-9}) = C_3$, the initial iteration of the **while** loop would set seq to $\langle C_3 \rangle$, and A_1 to

SAFESEQUENCE$(\{C_1, C_2\}, 4)$ — we have seen in Example 4 that this is the sequence $\langle C_2 \rangle$.
- The next iteration has $S = \{C_1, C_2\}$, $d = 8 - 4 = 4$ and $v = 10^{-9} \div 10^{-6} = 10^{-3}$. Notice that this sub-problem has been solved (represented by the node in Figure 5 labeled $(\{C_1, C_2\}, 4, 10^{-3})$).
- Since $G_C(\{C_1, C_2\}, 4, 10^{-3}) = C_1$, C_1 is appended to the end of seq, which now has value $\langle C_3, C_1 \rangle$. A_2 is set equal to SAFESEQUENCE$(\{C_2\}, 2)$. We saw in Example 4 that this is the empty sequence $\langle \rangle$, which is the value assigned to A_2. Now, Line 10 assign q a value $10^{-3} \div 10^{-4}$, or 10. Therefore the **while** loop is exited and we return from the call to GENERATESEQUENCES.

Summarising, the returned sequence is $\langle C_3, C_1 \rangle$, while the alternative sequences are $A_1 = \langle C_2 \rangle$ and $A_2 = \langle \rangle$. The reader may verify that this is indeed the semi-adaptive strategy we had presented (without derivation) in Example 2. □

6 EVALUATION

In this section we conduct simulation experiments upon randomly-generated synthetic workloads in order to experimentally evaluate our semi-adaptive strategy. In this evaluation we first examine how long it takes our algorithm to construct the initial and alternative sequences: these experiments reveal the *scalability* of our approach — how many components our algorithm is able to accommodate with acceptable pre-run-time overhead. We then compare the schedules generated at run-time by the semi-adaptive algorithm with those generated by a static approach. We are thus able to experimentally characterize the benefits of adaptivity.

Random sets of components, representing the AAs available to us, are obtained by sampling uncertainty values generated by the DRS algorithm [13]. DRS enables a given budget of worst case and typical estimated of uncertainty to be uniformly divided amongst the components, while maintaining the normal relationship between worst case and typical. In addition, bounds are provided to prevent any given component from having an uncertainty value which is too dominant. Execution times for components were generated from a uniform distribution. Hence for these experiments a set of n components is defined by the following:

(1) Array d = [randint(min_execution_time, max_execution_time) for _ in range(n)]
(2) Array X = drs(n, *total_worst_case_uncertainty*, upper_bound=1.0, lower_bound = worst_case_lower_bound)
(3) Array Y = drs(n, *total_worst_case_uncertainty*, upper_bounds=X, lower_bound = typ_case_lower_bound)
(4) $C_i = (d_i, X_i, Y_i)$)

where drs(n, u, ...) is the DRS function that returns an array of n floating point values that multiply[7] to u, with additional constraints as specified by the named parameters, and C is a component as defined by Equation (1).

---

[7]The normal DRS algorithm uses the sum; however, as the positive real numbers under addition are isomorphic to [0, 1) under multiplication, it is trivial to adapt DRS to a multiplicative domain.
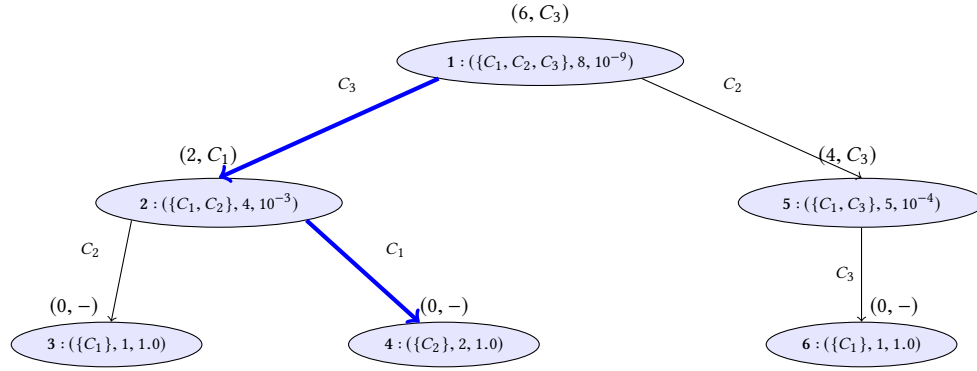
Figure 5: An example of recursive calls made during pre-processing by the Semi-Adaptive Strategy (discussed in Example 8). Each vertex is labeled with the three-tuple (S, d, q), denoting that an uncertainty $\leq q$ over an interval of duration d must be guaranteed using the set of components S. The values returned by each call are given as an ordered pair: the ordered pair over the vertex labeled (S, d, q) represents $(G(S, d, q), G_C(S, d, q))$.

GENERATESEQUENCES(S, d, q)

    // Returns the initial sequence, and sets A_i to be the
      // i'th alternative sequence
1   $seq = \langle\ \rangle$ **//** Initialise the initial sequence to be the
    // empty sequence
2   $i = 1$
3   **while** $(d > 0) \wedge (q < 1)$
4   Suppose that the component $G_C(S, d, q)$ is C_k
5   append C_k to seq
6      $A_i = $ SAFESEQUENCE$(S \setminus \{C_k\}, d - d_k)$
    // Pseudo-code in Figure 3
      // If C_k returns an uncertainty $> q_k^T$, then set A_i
        // to the schedule that guarantees the minimum
        // required uncertainty within the remaining
        // duration $(d - d_k)$, using the remaining
        // components $(S \setminus \{C_k\})$.
7      $i = i + 1$
8      $S = S \setminus \{C_k\}$
9      $d = d - d_k$
10    $q = q \div q_k^T$
11   return seq

Figure 6: Semi-adaptive Strategy: Generating initial and alternative sequences

Figure 7: Median Observed Running Time for Table-Generation

Once the components are specified, the deadline D is obtained by multiplying the total execution time by a constant that represents the proportion of components that can be considered for execution. Multiple values for this constant were tested and not found to have a significant effect on the results obtained, so only the results for 0.9 are presented. Finally, the target uncertainty bound $\mathbb{Q}$ is calculated by computing the best safe bound that the system is capable of. Due to the manner in which the target uncertainty bound is set, the impact of the majority of parameters is minimised.

Figure 7 shows how the running time of the algorithm scales with the number of components considered, with each data point representing the average of the generation times

of schedules for 250 randomly-generated instances. (These measurements were taken on an AMD EPYC 7501 running at 2.4GHz.) As can be seen, for the semi-adaptive algorithm this duration grows exponentially with respect to the number of components. The data in Figure 7 indicate that tables for up to 20 components can be constructed in approximately 15 minutes. In practice perhaps 6, or at most 10 components are likely to be employed in an AMS, in which case the running time for schedule generation is then less than a second.

Having thus established the viability of the semi-adaptive strategy, we next evaluate its efficacy by computing the response time (predicted execution time) of the typical sequence and comparing this to the response time if we were to have done static scheduling. Figure 8 shows these values for the number of components ranging from 3 to 20 (each point being the median value of 250 randomly produced examples). The

Sanjoy Baruah, Alan Burns, and David Griffin

Figure 8: Response Time for Static and Semi-adaptive Schedules

figure shows a clear reduction: for most numbers of components response times are less than half that required by even an optimal static scheme. This additionally implies that fewer than half the number of components are typically executed under semi-adaptive scheduling when compared to the static method.

The results presented in Figure 8 are for randomly generated parameters following uniform distributions — we were unable to find convincing arguments in, e.g., the machine learning literature, to support the use of other distributions for modeling AA components. However, it is reasonable to speculate that some AA components are likely to have more skewed behaviours: there may be AI-based algorithms that can only guarantee very poor outcomes (uncertainty close to 1.0) in the worst case, but perform much better –express lower uncertainty (i.e., greater certainty) that the outputs they generate are correct– under typical circumstances. It is expected that the availability of such components will significantly enhance the performance of semi-adaptive strategies vis-a-vis static ones since they would never form part of a static sequence but are likely to be chosen in semi-adaptive sequences.

Finally, whilst the above has illustrated the benefits that accrue when typical values (or better) are experienced it is theoretically possible for semi-adaptive scheduling to perform worse than static scheduling, provided that a sufficient number of components return their worst-case uncertainty value. An investigation was undertaken to find these situations in our analysis, and calculate an upper bound on the probability of the failure of a component to return its typical case uncertainty value. This was accomplished by enumerating all possible schedules and determining how many such failures were necessary to cause semi-adaptive scheduling to perform worse than static scheduling. Even in the worst case, it was found that provided the probability of a worst case uncertainty value being returned was less than 0.45, semi-adaptive scheduling would perform at least as well as static scheduling. As a probability of 0.45 would represent a common event, this suggests that for the hypothesis that a typical uncertainty value represents the worst case in all but very rare circumstances, semi-adaptive scheduling will perform better than static scheduling. It is possible to devise pathological cases which push the performance down further, but these seem to be extremely rare when using randomly generated parameters.

7 CONCLUSIONS

Autonomous Agents (AAs), including ones that are based upon deep learning and similar AI-based principles, are increasingly used in safety-critical CPS's; it is therefore imperative that the safety-critical systems research community devise techniques that enable the analysis of such systems to both assure safety (which is essential) and optimize performance (which is desirable, for cost and related reasons). This paper reports on some of our ongoing efforts in this direction. Building off recent work on *typical-case analysis* pioneered in [31] we have argued that safety-critical systems whose run-time behaviors incorporate a great deal of uncertainty should be designed to optimize for performance in the typical case (while guaranteeing safety in all cases, including atypical ones). We have developed a model for representing the run-time behaviour of some kinds of AAs that exhibit run-time functional uncertainty in a quantitative manner. We have applied this model to a problem that arises in an application relating to navigation in autonomous mobile systems, in which one or more of an available array of AAs are to be used, in any desired order, to perform a safety-assessment operation as soon as possible (for superior performance), but definitely within a specified deadline (for safety). We have identified a trade-off, formalized in the concepts of *static*, and *semi-adaptive* strategies, between run-time performance and the pre-run time computation that is needed, and have developed and experimentally evaluated algorithms for obtaining provably optimal semi-adaptive strategies.

Future work will look to move from a semi-adaptive to a more fully adaptive scheme. For such schemes the run-time overhead may be prohibitive; but this is yet to be fully investigated.

Acknowledgements

REFERENCES

[1] Aysegul Acar, Yakup Demir, and Cuneyt Guzelis. 2017. Object recognition and detection with deep learning for autonomous driving applications. *SIMULATION* 93, 9 (2017), 759–769.

[2] Kunal Agrawal and Sanjoy Baruah. 2019. Adaptive Real-Time Routing in Polynomial Time. In *Real-Time Systems Symposium (RTSS), 2019 IEEE*.

[3] Kunal Agrawal, Sanjoy Baruah, and Alan Burns. 2020. The Safe and Effective Use of Learning-Enabled Components in Safety-Critical Systems. In *2020 32nd Euromicro Conference on Real-Time Systems*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[4] Sanjoy Baruah. 2018. Rapid routing with guaranteed delay bounds. In *Real-Time Systems Symposium (RTSS), 2018 IEEE*.

[5] Sanjoy Baruah. 2021. Real-Time Scheduling of Multistage IDK-Cascades. In *2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC)*. 79–85. https://doi.org/10.1109/ISORC52013.2021.00021

[6] Sanjoy Baruah, Alan Burns, and Yue Wu. 2021. Optimal Synthesis of IDK-Cascades. In *Proceedings of the Twenty-Ninth International Conference on Real-Time and Network Systems (RTNS '21)*. ACM, New York, NY, USA.

[7] Peter Bishop, Robin Bloomfield, and Peter Adelard. 2002. Justifying the use of software of uncertain pedigree (SOUP) in safety related applications. In *Proceedings of the 5th International Symposium Programmable Electronic Systems in Safety Related Applications*.

[8] Robin E. Bloomfield. 2001. Methods for assessing the safety integrity of safety-related software of uncertain pedigree (SOUP). Prepared by Adelard for the Health and Safety Executive (HSE), UK.

[9] R. Calinescu, K. Johnson, and C. Paterson. 2016. FACT: A Probabilistic Model Checker for Formal Verification with Confidence Intervals. In *Tools and Algorithms for the Construction and Analysis of Systems*, Marsha Chechik and Jean-François Raskin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 540–546.

[10] Claudia D'Ambrosio, Fabio Furini, Michele Monaci, and Emiliano Traversi. 2018. On the Product Knapsack Problem. *Optimization Letters* 12, 4 (2018). https://doi.org/10.1007/s11590-017-1227-5

[11] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *Proc. ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 1–14.

[12] Y. Gal and Z. Ghahramani. 2016. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proc. of the 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 1050–1059.

[13] David Griffin, Iain Bate, and Robert I. Davis. 2020. Generating Utilization Vectors for the Systematic Evaluation of Schedulability Tests. In *IEEE Real-Time Systems Symposium, RTSS 2020, Houston, Texas, USA*. IEEE. http://eprints.whiterose.ac.uk/167646/

[14] C. Guo, G. Pleiss, Y. Sun, and K.Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proc. of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 1321–1330.

[15] David Harel, Assaf Marron, and Joseph Sifakis. 2019. Autonomics: In Search of a Foundation for Next Generation Autonomous Systems. arXiv:cs.SE/1911.07133

[16] S. Heo, S. Cho, Y. Kim, and H. Kim. 2020. Real-Time Object Detection System with Multi-Path Neural Networks. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 174–187.

[17] Hanzhang Hu, Debadeepta Dey, Martial Hebert, and J. Andrew Bagnell. 2019. Learning Anytime Predictions in Neural Networks via Adaptive Loss Balancing. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 3812–3821. https://doi.org/10.1609/aaai.v33i01.33013812

[18] Wonseok Jang, Hansaem Jeong, Kyungtae Kang, Nikil Dutt, and Jong-Chan Kim. 2020. R-TOD: Real-Time Object Detector with Minimized End-to-End Delay for Autonomous Driving. arXiv:cs.CV/2011.06372

[19] Weiwen Jiang, Edwin H.-M. Sha, Xinyi Zhang, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu. 2019. Achieving Super-Linear Speedup across Multi-FPGA for Real-Time DNN Inference. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 67 (2019), 23 pages.

[20] A. Kendall, V. Badrinarayanan, and R. Cipolla. 2016. Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding. arXiv:cs.CV/1511.02680

[21] F. Khani, M. Rinard, and P. Liang. 2016. Unanimous Prediction for 100% Precision with Application to Learning Semantic Mappings. In *Association for Computational Linguistics (ACL)*.

[22] B. Lakshminarayanan, A. Pritzel, and C. Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. arXiv:stat.ML/1612.01474

[23] J. Lee, A. Prajogi, E. Rafalovsky, and P. Sarathy. 2016. Assuring Behavior of Autonomous UxV Systems. In *S5: The Air Force Research Laboratory (AFRL) Safe and Secure Systems and Software Symposium*.

[24] S. Lee and S. Nirjon. 2020. SubFlow: A Dynamic Induced-Subgraph Strategy Toward Real-Time DNN Inference and Training. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 15–29.

[25] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. 2020. PCONV: The Missing but Desirable Sparsity in DNN Weight Pruning for Real-Time Execution on Mobile Devices. *Proc. of the AAAI Conference on Artificial Intelligence* 34, 04 (2020), 5117–5124.

[26] Dr. Sandeep Neema. [n.d.]. Assurance for Autonomous Systems is Hard. https://www.darpa.mil/attachments/AssuredAutonomyProposersDay_ProgramBrief.pdf. Accessed: 2019-03-07.

[27] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-Based Weight Pruning. In *Proc. oTwenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Association for Computing Machinery, 907–922.

[28] M.E. Paoletti, J.M. Haut, J. Plaza, and A. Plaza. 2019. Deep learning classifiers for hyperspectral imaging: A review. *ISPRS Journal of Photogrammetry and Remote Sensing* 158 (2019), 279–317.

[29] F. Pereira, T. Mitchell, and m. Botvinick. 2009. Machine learning classifiers and fMRI: – a tutorial overview. *NeuroImage* (2009), S199–S209.

[30] Ulrich Pferschy, Joachim Schauer, and Clemens Thielen. 2019. The Product Knapsack Problem: Approximation and Complexity. arXiv.

[31] Sophie Quinton, Matthias Hanke, and Rolf Ernst. 2012. Formal Analysis of Sporadic Overload in Real-time Systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '12)*. EDA Consortium, San Jose, CA, USA, 515–520. http://dl.acm.org/citation.cfm?id=2492708.2492836

[32] Joseph Sifakis. 2018. Autonomous Systems – An Architectural Characterization. arXiv:cs.SY/1811.10277

[33] John A. Stankovic and Krithi Ramamritham. 1990. What is Predictability for Real-time Systems? *Real-Time Syst.* 2, 4 (Oct. 1990), 247–254. https://doi.org/10.1007/BF01995673

[34] T. P. Trappenberg and A. D. Back. 2000. A classification scheme for applications with ambiguous data. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, Vol. 6. 296–301 vol.6.

[35] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, and Joseph E. Gonzalez. 2017. IDK Cascades: Fast Deep Learning by Learning not to Overthink. *CoRR* abs/1706.00885 (2017). arXiv:1706.00885 http://arxiv.org/abs/1706.00885

[36] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018. SkipNet: Learning Dynamic Routing in Convolutional Networks. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII (Lecture Notes in Computer Science)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.), Vol. 11217. Springer, 420–436. https://doi.org/10.1007/978-3-030-01261-8_25

[37] P. Wei, L. Cagle, T. Reza, J. Ball, and J. Gafford. 2018. LiDAR and Camera Detection Fusion in a Real-Time Industrial Multi-Sensor Collision Avoidance System. *Electronics* 7, 6 (2018).

[38] Y. Xiang and H. Kim. 2019. Pipelined Data-Parallel CPU/GPU Scheduling for Multi-DNN Real-Time Inference. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*. 392–405.

[39] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. Abdelzaher. 2020. Scheduling Real-time Deep Learning Services as Imprecise Computations. In *Proc. IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 1–10.

[40] York. 2021. Assuring autonomy international programme. https://www.york.ac.uk/assuring-autonomy/. Accessed: 2021-01-17.

[41] J. Zhang, F. Li, H. Wu, and F. Ye. 2019. Autonomous Model Update Scheme for Deep Learning Based Network Traffic Classifiers. In *Proc. IEEE Global Communications Conference (GLOBECOM)*. 1–6.

[42] H. Zhou, S. Bateni, and C. Liu. 2018. S3DNN: Supervised Streaming and Scheduling for GPU-Accelerated Real-Time DNN Workloads. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 190–201.