Hierarchical Algorithms for Assessing Probabilistic Constraints on System Performance [*]

G. DE VECIANA, M. JACOME AND JIAN-HUEI GUO

Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas 78712
Tel: (512) 471-2051 Fax: (512) 471-5532
E-mail: {gustavo,jacome,jianhuei}@ece.utexas.edu

Abstract

We propose an algorithm for assessing probabilistic performance constraints for systems including components with uncertain delays. We make a case for designing systems based on a probabilistic relaxation of performance constraints, as this has the potential for resulting in lower silicon area and/or power consumption. We consider a concrete example, an MPEG decoder, for which we discuss modeling and assessment of probabilistic throughput constraints.

KEYWORDS: *Statistical design constraints, system-level specification and design aids, hierarchical design, probabilistic critical path detection, Hardware/Software co-design.*

1 Introduction

This paper discusses models and algorithms to support statistical relaxations of worst case constraints on system performance. Consider, for example, a system which is designed to meet a delay constraint d and suppose the critical path's delay D^p is in fact random. A design based on a *worst case analysis* would ensure that $\mathbb{P}(D^p > d) = 0$. In our view, for a number of application domains, such designs may be unnecessarily conservative. For example, suppose the design constraint d can be relaxed in the sense that it can be violated but only rarely, say $\mathbb{P}(D^p \geq d) \leq 10^{-6}$. Such a relaxation of design constraints will in turn allow for a larger set of acceptable design solutions with hopefully less demanding performance requirements and/or power consumption. Note that even when performance constraints are truly worst case, in the sense that the system malfunctions if they are not met, it is reasonable, and possibly beneficial, to still relax the performance constraints – say, to the same level of certainty as the probability of failure of the system's components. The examples in Section 4 suggest that one might expect to benefit significantly from a probabilistic relaxation of worst case constraints for systems comprising a large number of non-deterministic components.

[*]The work of G. de Veciana is supported by an NSF Career Grant NCR-9624230. The work of M. Jacome is supported by by an NSF Career Grant MIP-9624321.

Uncertainty in the performance of a system's components may have a variety of origins. For example, for high-level system representations, such as those used in system level and hardware/software codesign, uncertainty may be due to a looping index whose exact initial value is unknown, *e.g.,* data dependent [1, 4, 7, 6, 9, 15, 16]. A number of current system-level models use hierarchy and aggregation as a means of controlling complexity [4, 15]. If such approaches are to succeed, and one is to reason efficiently about the required performance of such systems during design space exploration, it will become increasingly critical to capture the performance variability of aggregated system elements.

We propose to use probability distributions to model uncertainty. The distributions may either be derived from statistical models for the underlying source of variability, estimated based on experimental data, or gathered through simulation/profiling. Realizing that characterizing distributions is in itself a challenging and expensive task, in Section 3.1 we propose a crude model based on knowing the mean and upper and lower bounds on delay. This simple characterization of non-determinism is shown to be conservative for assessing system performance and eliminates (in some cases) the need for obtaining detailed statistical information on component delays.

In Section 2 we formulate a *probabilistic critical path* problem, and propose an approximate algorithm for assessing probabilistic constraints on systems represented by directed acyclic graphs (DAGs) with random edge weights. Therein we discuss related work. In Section 3 we discuss modeling distributions as well as a set of transformations/reductions of typical high-level system models to obtain DAGs which are in turn amenable to analysis with the proposed algorithm. Synthetic examples exhibiting significant differences between worst case and probabilistic requirements are presented in Section 4. In Section 5 we discuss a concrete example, showing how our approach might be applied to modeling and assessing throughput constraints for an MPEG decoder. We conclude with a discussion of research/implementation directions we are currently pursuing.

2 Algorithm for Assessing Probabilistic Constraints

Consider a weighted directed acyclic graph $G(V,E)$. Suppose a source node $s \in V$ is selected and let P_s denote the set of paths starting at s. Naturally, a path p is an ordered set of adjacent edges in the graph, *i.e.,* $p \subset E$. The standard *critical path* problem assumes that edges have fixed weights, and determines the longest path, in terms of cumulative weight, in P_s. Algorithms to solve this problem are well known and have a runtime complexity of $\Theta(|V|+|E|)$. Suppose random weights (delays) D_e, with arbitrary distributions, are associated with edges $e \in E$ of the graph. We pose an analogous *probabilistic critical path* problem as follows: given a delay constraint d, identify which path is *most likely* to violate the constraint and what is its probability of failure.

Assuming edge delays are mutually independent, a path's delay, denoted by $D^p = \sum_{e \in p} D_e$, has a distribution given by the convolution of its edge's delay distributions. The probability that a path p fails to meet the delay constraint d is denoted by

$$\pi_d(p) = \mathbb{P}(D^p \geq d).$$

We state the probabilistic critical path problem as follows:

Problem 1 *Find $p^* \in P_s$, not necessarily unique, such that $\pi_d(p^*) \geq \pi_d(p)$ for all $p \in P_s$, and determine the probability $\pi_d(p^*)$ that the constraint will be violated.*

In general this problem is difficult to solve, primarily because the edge weights of a path are not additive (due to the convolution) and thus cannot be decomposed along the path as is usual when using dynamic programming approaches. In fact, by adapting a Lemma 2 in [5], one can show that Problem 1 is NP-hard, suggesting that one should seek good heuristics.

A word of warning is in order. There are two reasons why the probabilistic critical path problem should not be interpreted as the equivalent of the standard critical path problem when the weights are random. First, the problem is predicated on specifying a constraint d with respect to which a probabilistic critical path is identified. Second, and more subtly, we compare the performance of *individual* paths with each other, rather than assessing the maximum of the delays across all paths. Whereas in the case with deterministic weights these two problems are equivalent, in a graph with random weights they certainly are not. The discussions on modeling in §3 and examples in §4 further elucidate this point.

2.1 Previous Work

To our knowledge there is no previous work addressing the above problem. Our work was inspired from recent work in network routing considering the uncertainty in the delay or bandwidth availability at remote links [5]. The flavor of the approach, in particular our use of the Chernoff bound to perform constraint analysis for a simple example, can be found in [6][page 112]. However, both our formulation of the probabilistic critical path problem as well as the proposed systematic algorithm are new. We also note that there are efficient algorithms for determining the most reliable path through a network with unreliable links, a problem which arises in voice recognition and Viterbi decoding applications. However such problems are significantly easier (can be reduced to a the traditional shortest paths problems) than the problem we address here.

2.2 Approximate Algorithm

We propose an algorithm to solve the probabilistic critical path problem based on an approximate formulation as a convex optimization problem. The edge weight distributions D_e on the DAG are represented via a parametric weight $\Lambda_e(\theta) = \log \mathbb{E} \exp[\theta D_e]$ for $\theta \geq 0$, *i.e.*, the moment generating function of the delay distribution on the edge. This results in a collection of DAGs with deterministic weights $\Lambda_e(\theta)$ parameterized by θ. Our algorithm solves an optimization problem over this set of parameterized DAGs by using the standard critical path algorithm. The derivation of the algorithm has been relegated to Appendix A.

Initialization: check that the problem is "well posed" by verifying that:

1. the constraint d exceeds the critical path delay for the graph where the weights are given by the mean edge delays;
2. and, the constraint d is bounded by the critical path delay for the graph with weights given by the maximum (possibly infinite) delay on each edge.

Optimization: determine the maximum $f^*(d)$ and the optimizers $\hat{\theta}$ and $\hat{p}$ for the following optimization problem

$$f^*(d) = \sup_{\theta \geq 0}(\theta d - \max_{p \in P_s} \sum_{e \in p} \Lambda_e(\theta)) = \hat{\theta} d - \sum_{e \in \hat{p}} \Lambda_e(\hat{\theta}). \tag{1}$$

Note that evaluating $\max_{p \in P_s} \sum_{e \in p} \Lambda_e(\theta)$ for some θ requires determining the critical path in a graph with edge weights $\Lambda_e(\theta)$.

Result: a *guaranteed* upper bound on the probability of failure, $\pi_d(p^*) \leq \exp[-f^*(d)]$, and a candidate path $\hat{p}$ most likely to violate the constraint.

2.3 Algorithm Complexity and Other Remarks

Standard line search methods, see *e.g.*, [12], can be used to determine the supremum, which will be achieved in the cases of interest, in (1). However some care should taken in selecting an efficient algorithm since each evaluation of $f(\theta) = \max_{p \in P_s} \Lambda^p(\theta)$ requires solving a standard critical path problem incurring a runtime cost $\Theta(|V| + |E|)$. Note that, whatever optimization algorithm is selected, any stopping criterion will yield an *upper* bound on the failure probability.

If the edges on the graph have distributions selected from a finite set C then $\sum_{e \in p} \Lambda_e(\theta) = \sum_{c \in C} n(p,c) \Lambda_c(\theta)$ where $n(p,c)$ is the number of elements of type c on path p. Such a representation may be appropriate and improve the algorithm's efficiency in some cases, particularly when considering a further optimization over *sets of possible implementations*, *i.e.*, various parameters c.

Once $\hat{p}$ is obtained one might attempt to accurately compute the probability of failure for the path by directly performing the convolution of its edge's delay distributions. For long paths this might be a prohibitively expensive operation. Alternatively, if the path achieving the maximum has a large number, *n*, of random edges with distributions selected from C, as described above, then one can use the Bahadhur-Rao estimate to improve upon the Chernoff estimate, see (4) in Appendix A. Moreover it may also be of interest to examine the *sensitivity* of the probability of failure to the constraint d. Techniques for performing these tasks are further discussed in the Appendix B.

3 Modeling Issues

3.1 Modeling edge delay distributions

In practice it may be difficult to characterize the edge delay distributions, or equivalently the corresponding functions $\Lambda_e(\theta) = \log \mathbb{E} \exp[\theta D_e]$. Fact 3.1 below shows that one can find a simple uniform bound on $\Lambda_e(\theta) \le \overline{\Lambda}_e(\theta)$, given a *minimal amount* of information on the distribution of D_e. This in turn allows us to replace the weights on such edges by an upper bound. By considering (5) it should be clear that the proposed algorithm would once again give a conservative estimate for the probability of failure.

Fact 3.1 (See *e.g.*, [13, 8]) *Suppose that bounds, $l_e \le D_e \le u_e$, are known for the edge (or path) delay, as well as an upper bound m_e on the average delay $\mathbb{E}D_e \le m_e \le u_e$. Let $\overline{D}_e$ be a Bernoulli random variable on $\{l_e, u_e\}$ with mean m_e, i.e.,*

$$\mathbb{P}(\overline{D}_e = u_e) = \frac{m_e - l_e}{u_e - le}, \text{ and } \mathbb{P}(\overline{D}_e = l_e) = 1 - \frac{m_e - l_e}{u_e - l_e},$$

then $\forall \theta$ we have that $\Lambda_e(\theta) \le \overline{\Lambda}_e(\theta) = \log \mathbb{E} \exp[\theta \overline{D}_e]$.

3.2 Hierarchical representations and reductions to directed acyclic graphs

In general a hierarchical high-level system model comprises a variety of nodes and arc's representing computations and flow of control constructs, including branching, looping, and possible synchronization requirements, see *e.g.*, [9, 7, 4, 1, 15]. Rather than formally defining such a framework we will exhibit some cases that arise and the manner in which they are reduced to a corresponding DAG. Below we show how the delay weights associated with traversing the nodes in Fig. 1 can be reduced to *equivalent path weights*. Generalizations of these cases to more than two non-intersecting (independent) sub-paths should be clear from the discussions below.

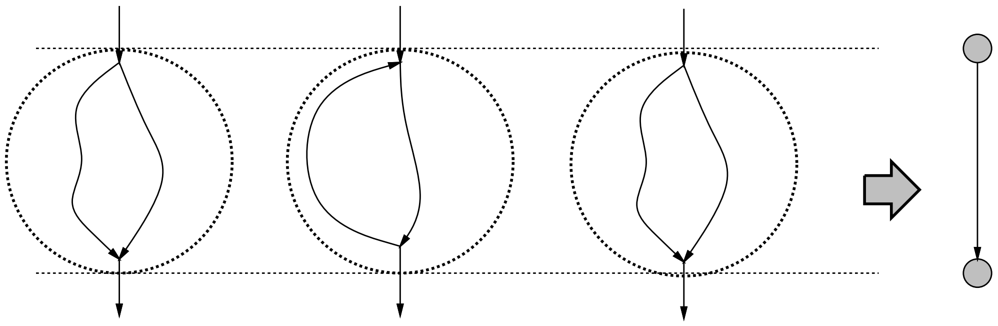

Figure 1: Probabilistic branching, looping, and synchronization reductions.

3.2.1 Reducing nodes with probabilistic branches.

Consider the left node in Fig. 1. Within the node, a branch is modeled probabilistically, in the sense that one of the two sub-paths, p_1 or p_2, is selected at random. Suppose the branching probability is γ then the weight for the sub-path p through this node is given by

$$\Lambda^p(\theta) = \gamma\Lambda^{p_1}(\theta) + (1-\gamma)\Lambda^{p_2}(\theta).$$

Note that if a branch is not modeled probabilistically (due to lack of information) then both paths would be kept in the eventual DAG.

3.2.2 Reducing nodes with iterations or feedback loops.

Consider the middle node in Fig. 1. It represents a node in which there is an uncertain number of iterations through path p_1, which can be modeled via a loop index random variable, say $N > 0$. Let $D(n)$, denote the delay for the n^{th} loop execution. Suppose these delays have a common distribution, $D(n) \sim D^{p_1}$, and are mutually independent and independent of the loop index N.

The delay D^p for a sub-path through this node is $D^p = \sum_{n=1}^{N} D(n)$, *i.e.*, a random sum of random variables. The weight for the node can be computed by conditioning on N to obtain

$$\Lambda^p(\theta) = \log\mathbb{E}[\exp\theta D^p] = \log\mathbb{E}[\mathbb{E}[\exp\theta\sum_{n=1}^{N} D(n)|N]] = \log\mathbb{E}[(\mathbb{E}[\exp\theta D^{p_1}])^N] = \log M_N(\mathbb{E}[\exp\theta D^{p_1}]),$$

where $M_N(z) = \mathbb{E}[z^N]$ is the probability generating function of the loop index's probability mass function.

For example, suppose the loop index, N, is modeled by a geometric distribution with parameter γ, *i.e.*, after completion of any iteration the probability of looping back is $1-\gamma$. In this case $M_N(z) = \frac{\gamma z}{1-z(1-\gamma)}$ and the overall weight for the sub-path through the looping node is

$$\Lambda^p(\theta) = \log\left[\frac{\gamma\exp[\Lambda^{p_1}(\theta)]}{1-\exp[\Lambda^{p_1}(\theta)](1-\gamma)}\right].$$

If the loop index is deterministic, *i.e.*, $\mathbb{P}(N=n) = 1$, then the corresponding DAG would unravel the loop, *i.e.*, the weight for a path through this node would be $\Lambda^p(\theta) = n \times \Lambda^{p_1}(\theta)$.

3.2.3 Reducing nodes with synchronization constraints.

In general synchronization is the most difficult abstraction to handle, particularly in a setup with random delays. Consider the rightmost example in Fig. 1, where two paths p_1 and p_2 must synchronize prior to leaving the node. The delay incurred in this node, D^p, is given by $D^p = \max[D^{p_1}, D^{p_2}]$, the maximum of the delay along the two paths. The weight for this node would be

$$\Lambda^p(\theta) = \log \mathbb{E} \exp[\theta D^p] = \log \mathbb{E} \exp[\theta \max[D^{p_1}, D^{p_2}]].$$

Unfortunately there is no general way to compute this metric, without explicitly computing the distributions for the maximum of the delays for the two paths.

Notice that whereas for graphs with deterministic delays we need only consider the worst case path to deal with synchronization, in the case of random path delays, both paths contribute to the characteristics of the synchronization time – it is this coupling that makes such nodes difficult to address. Below we consider several special cases and propose ad hoc approximations that deal with a limited amount of synchronization without requiring costly explicit computation of distributions.

Deterministic path combined with random delay path. Consider the case where only one of the paths, say p_1 has "randomness" while the second path p_2 has a constant delay d^{p_2}. Assuming the distribution of D^{p_1} is known one might consider explicitly computing the node's weight:

$$\begin{aligned}
\Lambda^p(\theta) &= \log \mathbb{E} \exp[\theta \max[D^{p_1}, d^{p_2}]] \\
&= \log\left[\exp[\theta d^{p_2}] \times \mathbb{P}(D^{p_1} \le d^{p_2}) + \mathbb{E}[\exp[\theta D^{p_1}] | D^{p_1} > d^{p_2}] \times \mathbb{P}(D^{p_1} > d^{p_2})\right] && (2) \\
&\le \log\left[\exp[\theta d^{p_2}] \times \mathbb{P}(D^{p_1} \le d^{p_2}) + \exp[\Lambda^{p_2}(\theta)]\right]. && (3)
\end{aligned}$$

Two examples where (2) might be used to compute the weight for a synchronization node follow: first, suppose $D^{p_1} \sim \text{uniform}[l, u]$ and the non-trivial case where $l < d^{p_2} < u$ then

$$\Lambda^p(\theta) = \log\left[\exp[\theta d^{p_2}] \frac{d^{p_2} - l}{u - l} + \frac{\exp[\theta u] - \exp[\theta d^{p_2}]}{u - l}\right];$$

second, suppose $D^{p_1} \sim \text{exponential}(\lambda)$ then

$$\Lambda^p(\theta) = \log\left[\exp[\theta d^{p_2}](1 - \exp[-\lambda d^{p_2}]) + \frac{\lambda \exp[(\theta - \lambda) d^{p_2}]}{\theta(\lambda - \theta)}\right].$$

The upper bound (3) could be used to simplify computations in other cases.

Paths with delays on bounded intervals and known means. Suppose the delays on p_1 and p_2 have upper and lower bounds and known means, *i.e.,* $l^{p_i} \le D^{p_i} \le u^{p_i}$ with $\mathbb{E} D^{p_i} = d^{p_i}$ for $i = 1, 2$. In this case the synchronization time satisfies the following inequalities:

$$l^p = \max[l^{p_1}, l^{p_2}] \le D^p = \max[D^{p_1}, D^{p_2}] \le \max[u^{p_1}, u^{p_2}] = u^p \quad \text{and} \quad \mathbb{E} D^p = d^p \le d^{p_1} + d^{p_2}.$$

Based on Fact 3.1 a conservative approximation for the weight of D^p is that of a Bernoulli random variable $\bar{D}^p$ with

$$\mathbb{P}(\bar{D}^p = u^p) = \frac{\min[d^{p_1} + d^{p_2}, u^p]}{u^p} = \alpha, \quad \text{and} \quad \mathbb{P}(\bar{D}^p = l^p) = 1 - \alpha.$$

Prob. of failure	$m = 1/2$ $n = 10$	$n = 20$	$n = 80$	$m = 1/4$ $n = 10$	$n = 20$	$n = 80$
Exact	1.07×10^{-2}	2.01×10^{-4}	2.69×10^{-14}	2.96×10^{-5}	1.61×10^{-9}	1.35×10^{-34}
Chernoff	2.52×10^{-2}	6.35×10^{-4}	1.63×10^{-13}	7.38×10^{-5}	5.45×10^{-9}	8.81×10^{-34}

Table 1: Probabilities of constraint violation and Chernoff approximations.

More explicitly we have that

$$\Lambda^p(\theta) \leq \bar{\Lambda}^p(\theta) = \log \mathbb{E} \exp[\theta \bar{D}^p] = \log \{\exp[\theta l^p](1 + \alpha(\exp[\theta(u^p - l^p)] - 1))\}.$$

By using the delay metric $\bar{\Lambda}^p(\theta)$ for this node we can proceed safely knowing we will still obtain an upper bound on performance. Note that if $d^{p_1} + d^{p_2} \geq u^p$ then this reduces to using the worst case upper bound on synchronization. However when $d^{p_1} + d^{p_2} < u^p$ we can still glean some information on the probabilistic behavior of that node.

Last resort conservative bound. If the paths in the node are "short" relative to the critical path of the graph then a simple upper bound can be devised by noting that, $\max[D^{p_1}, D^{p_2}] \leq D^{p_1} + D^{p_2}$, so it follows that

$$\Lambda^p(\theta) \leq \log \mathbb{E} \exp[\theta D^{p_1}] + \log \mathbb{E} \exp[\theta D^{p_2}] = \Lambda^{p_1}(\theta) + \Lambda^{p_2}(\theta).$$

This is likely to be conservative in the probabilistic sense, yet may still be reasonable when compared to the results obtained using the worst case edge delay.

4 Synthetic Examples: Why use probabilistic v.s. worst case critical paths?

For simplicity let us assume that all edge delays are independent and identically distributed Bernoulli random variables with mean m, *i.e.,* $\mathbb{P}(D_e = 1) = m$ and $\mathbb{P}(D_e = 0) = 1 - m$. Suppose there is a single path through the DAG representing a system and it has n edges so $D^p = \sum_{i=1}^n D_i$. Clearly the worst case critical path would have a length of n. A probabilistic analysis might consider the likelihood that the delay exceeds 90 % of the worst case delay, *i.e.,* $\mathbb{P}(D^p \geq 0.9n)$. Table 1 exhibits some results for this setup, where both the length n of the path and the mean m of the edge delays are varied.

When $m = 1/2$ and the path is relatively long, say $n = 20, 80$ the probability of failure are $O(10^{-4})$ and $O(10^{-14})$ respectively, possibly small enough to be neglected. Thus a delay constraint which is 90 % of the worst case, is very likely to be met. For $m = 1/4$ even a path with a moderate number of elements $n = 10$ has a small probability of failure $O(10^{-5})$ again showing that a probabilistic relaxation of the constraint is likely to be advantageous.

Based on this simple example it should be clear that as we consider increasingly large systems with many uncertain elements, the gains of a probabilistic relaxation of constraints will accrue. Moreover if the delay distributions are such that the average performance is significantly smaller than the worst case bounds, *e.g.,* 75 % smaller when $m = 1/4$, then probabilistic constraints are likely to allow a significant relaxation over the worst case critical path. The failure probabilities in Table 1 were computed exactly based on Bernoulli distributions and via the Chernoff bound used by our algorithm. Clearly the results compare favorably, and as expected the Chernoff bound gives an upper bound on the failure probability. In summary these examples show that if indeed there is sufficient uncertainty in the performance of elements on a reasonably large graph, the proposed method is likely to pay off handsomely if one can allow for a probabilistic relaxation of constraints.

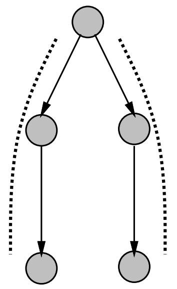

Figure 2: Probabilistic versus worst case critical paths.

In general the probabilistic and conventional critical paths need not coincide. Indeed, consider the graph in Fig. 2, where three edges have independent Bernoulli distributions on $\{0,1\}$ with means $1/2, 1/2, 1/8$ and the fourth is deterministic with mean 1.2. The worst case critical path is obviously p_2 with a delay of 2.2. Now, given a delay constraint $d = 1.5$ one can easily show that the probability of violation is largest on p_1, *i.e.,* $\mathbb{P}(D^{p_1} \geq 1.5) = 1/4 > 1/8 = \mathbb{P}(D^{p_2} \geq 1.5)$. This suggests that a designer optimizing a system based on worst case information *may* be addressing the wrong path, at least when a probabilistic relaxation of system constraints is possible.

5 Assessing Probabilistic Constraints for MPEG Video Decoders

In this section we illustrate the practical interest of the proposed algorithm for probabilistic constraint analysis by considering MPEG video decoders [10, 16, 2, 11]. The MPEG decoder was chosen because of the presence of non-deterministic (data dependent) delays in some of the key decoding sub-tasks. This example illustrates how the inherently variable nature of these tasks makes it interesting to assess probabilistic throughput constraints.

5.1 Background on MPEG-2

A video stream consists of a sequence of pictures or frames sampled at a given rate. Three basic types of pictures are defined: *intra-coded* pictures, which are coded without reference to other pictures; *forward/backward predictively coded* pictures, which can use motion prediction from a past/future picture; and *bidirectionally-predictively coded* pictures, which can use motion prediction from both past and future pictures. These are referred to as I, P and B pictures respectively.

Pictures are in turn subdivided into a number of *macroblocks* - a 16 by 16 pixel region. Depending on the picture type, a good match might be sought between its macroblocks and other pictures in the sequence, based on computing *motion vectors*. Thus a macroblock can be:

- *causal (forward coded):* defined from a previous picture, – allowed for macroblocks within P and B-pictures;
- *non-causal (backward coded):* defined from a future picture – allowed for macroblocks within B-pictures only;

- *interpolative (bidirectionally coded):* defined from a past and a future picture – allowed for macroblocks within B-pictures only.

Non-motion compensated macroblocks, are allowed for all types of pictures, and are said to be *intra-coded.*

As the MPEG-2 decoder reads the bitstream, it identifies the start and type of a coded picture, and then decodes each macroblock in the picture, as shown in Fig. 3.

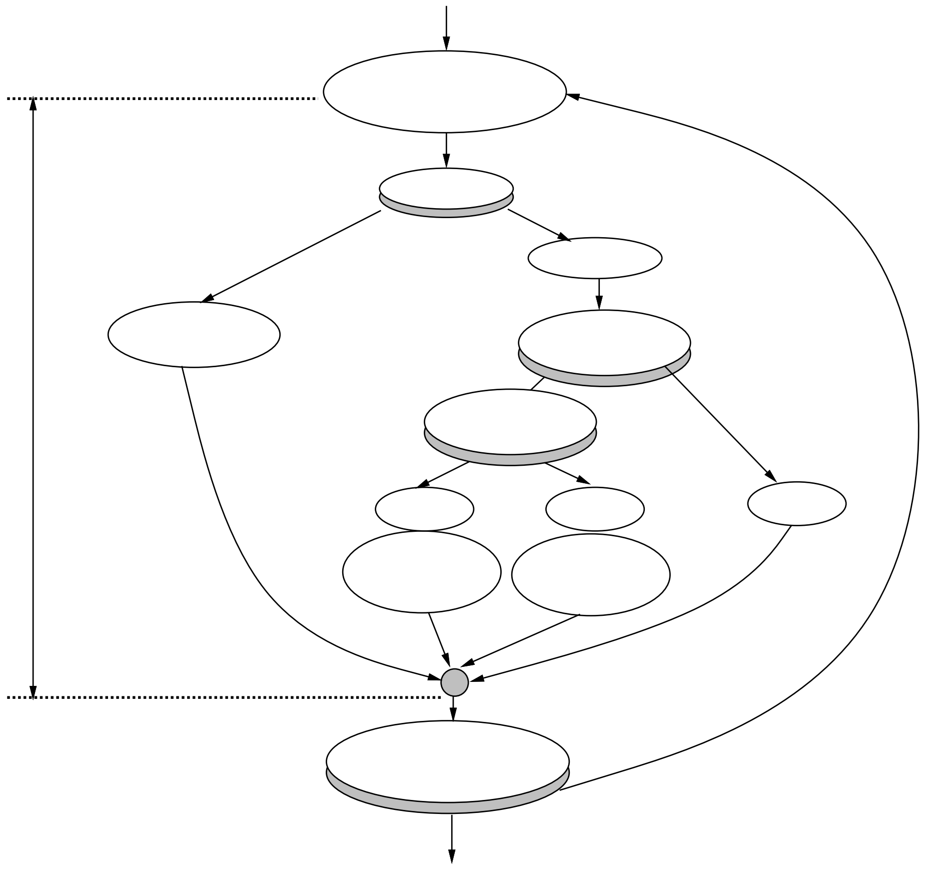

Figure 3: Macroblock decoding in an MPEG decoder.

		type of macroblock coding			
frame type	fraction	skip	intra-coded	forward/backward	bidirectional
I	1/15	0	1	0	0
P	4/15	0.0173	0.0658	0.9169	0
B	10/15	0.0848	0.0050	0.2226	0.6876

Table 2: Estimates of branching probabilities for MPEG macroblock decoding.

In Fig. 3, *N* represents the number of macroblocks in a picture – for the streams being considered a picture is comprised of 330 macroblocks. The shaded ellipses in Fig. 3 represent basic flow of control decisions taken during the decoding of each macroblock within a picture. Table 2 shows estimates of branching probabilities for these decision points. These estimates were generated by running a software decoder on a collection of MPEG video traces. The first two columns in Table 2 identify the type of picture (I, P, or B) and the percentage of occurrence of that particular type of picture in the fixed sequence of pictures considered for our MPEG-2 decoder. The third column in the table gives the branching probability for the first decision point in Fig. 3, i.e., the probability that a macroblock will be skipped within a P or a B picture (note that all macroblocks within an I pictures are intra-coded). The three last columns in Table 2 give the probability that a given macroblock will be intra-coded, forward/backward coded, or bidirectionally coded, for I, P and B pictures.

The performance of an MPEG-2 decoder is determined by the individual performance of five key modules: Variable Length Decoding (VLD), Inverse Quantization (IQ), Inverse DCT (IDCT), Pixel Interpolation (PI), and Pixel Add (PA) [10]. However not all the modules are executed for every macroblock. In particular, as shown in Fig. 3, none of the modules is executed for non-coded (or skipped) macroblocks, and the PI and PA Modules are not executed for intra-coded macroblocks. Moreover, the processing done by the PI and PA Modules for bidirectionally coded macroblocks is twice of that required by forward or backward coded macroblocks, since one additional reference macroblock needs to be considered in the first case. (This extra-processing is the reason for the separation of bidirectionally coded macroblocks from the two other types of motion compensated macroblocks in the control flow graph shown in Fig. 3.)

The algorithm-level descriptions of the MPEG-2 modules referred to above have been the focus of extensive studies on optimizations/transformations geared towards performance enhancement [10, 2]. In our example, we have adopted the set of highly optimized algorithmic descriptions derived by Lee and Kim [10]. In these behavioral descriptions, the VLC and IQ modules are merged in order to save on write/read cycles to memory.

5.2 Using Probabilistic Constraint Analysis to Guide the Design of an MPEG2 video Decoder

The objective in this example is to define/specify the RTL architecture (functional units and registers/memory) for the key MPEG-2 decoder modules referred to above, so as to derive a decoder supporting a throughput of 30 frames/sec (which translates into a 33.3 ms decoding time per picture).

Design Option 1 The modules' RTL descriptions of our initial design, referred to as Option 1, were directly derived from the modules' algorithmic descriptions given in [10]. The scheduling of operations within each module was strictly performed based on data dependencies, *i.e.,* the performance of such modules is never compromised by resource sharing. Memory blocks were assumed to be implemented by RAMs with a single read port (with two cycle read operations) and a single write port.

Table 3 shows the resulting execution delays (in # cycles) for the various MPEG-2 decoding modules. As mentioned previously, the execution delays of the PI and PA modules are given separately for bidirectionally coded and for forward/backward coded macroblocks.

A crucial observation needs to be made with respect to the numbers shown in Table 3 for the VLD+IQ Module. The execution delay of that module for each macroblock depends on the number of non-zero DCTs per macroblock, and is thus data dependent. In [10], the average size of VLCs in typical MPEG-2 bitstream was reported to be about 4.5 bits which in turn translates to an average of 30 non-zero DCTs per macroblock, *i.e.,* an average of 484 cycles per macroblock for Option 1. We have used a crude model for the delay of the VLD+IQ module given by a Gaussian distribution with this mean (see Table 3) and a standard deviation of 20 % of the mean, to account for the variability in the stream.

	macroblock prediction type	**Option 1**	**Option 2**
VLD + IQ (Average)	any	484	436
IDCT	any	2,304	1,152
Pixel Interpolation	forward/backward	320	160
	bidirectional	640	320
Picture Add	forward/backward	512	256
	bidirectional	1024	512

Table 3: Time estimates (# cycles per macroblock) for MPEG modules.

In the upper part of Fig. 4, we show the decoding time distributions for I, P, and B pictures for design Option 1, derived using the execution delays per macroblock (in # cycles) given in Table 3, the branching probabilities given in Table 2, and the previously mentioned model for the VLC+IQ block. Table 4 shows the corresponding average and worst case decoding times (in # cycles) for the three types of pictures, and also the worst case and average decoding time considering all picture types (given on the last row of the table).

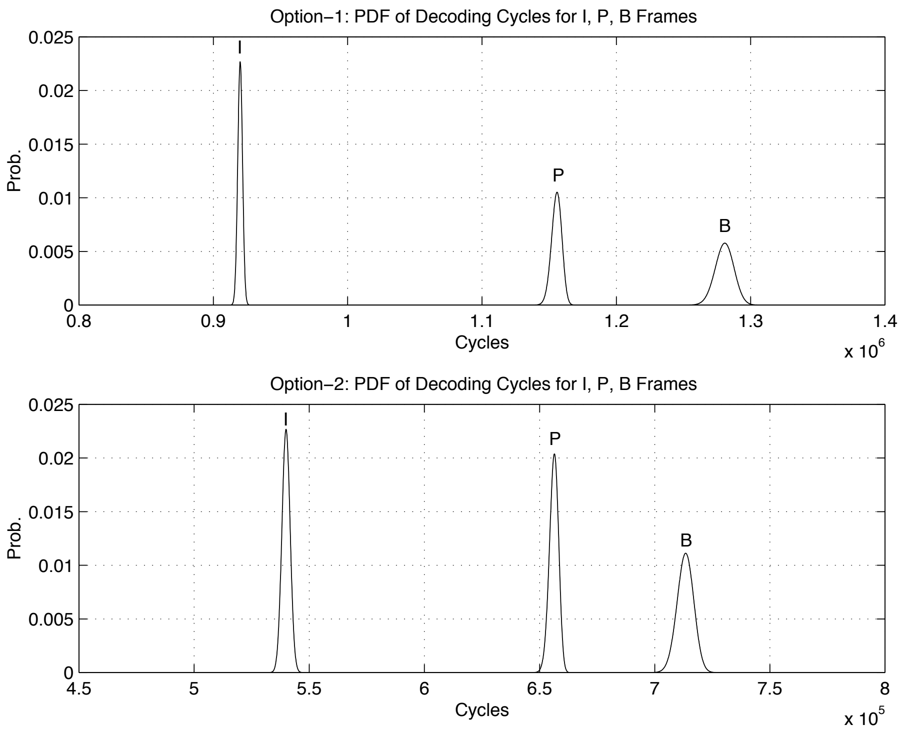

Figure 4: Decoding time distributions, for I,P and B Frames for Options 1 and 2

The maximum combinatorial delay for our module's RTL descriptions was determined to be 43 ns (for a 0.7 μm standard-cell library). So, for a 43 ns clock, our Option 1 design led to an average delay per picture of 52.6 ms (i.e., the decoder would only sustain a throughput 19 pictures per second), which is

	frame type	**Option 1**	**Option 2**
Average / Maximum ($\times 10^5$)	I	9.200 / 10.159	5.399 / 6.357
	P	11.559 / 12.500	6.564 / 7.506
	B	12.807 / 13.684	7.134 / 8.011
	all	12.234 / 13.684	6.866 / 8.011

Table 4: Average and worst case decoding times, in # cycles for I,P and B frames.

below the target of 33.3 ms per picture (i.e., the desired throughput of 30 pictures per second). Moreover, the resulting design exhibited a worst case picture decoding delay of 58.8 ms. Option 1 was thus clearly insufficient in terms of performance, and was dropped.

Design Option 2 A second implementation, which we will call Option 2, was then developed, taking advantage of the fact that the computations performed by the IDCT, the PI, and the PA Modules can be easily parallelized. A new design was developed that: (1) has two parallel IDCT units (i.e., can compute simultaneously two 8x8 2-D IDCTs, each of which is done as a loop whose body computes an 8-point IDCT); (2) has two parallel pixel interpolation and pixel add units[1]; and (3) uses RAMs with two parallel read ports (still with a two cycle read operation).

Table 3 shows the resulting execution delays (in # cycles) for the various decoder modules for Option 2. The bottom part of Fig. 4 shows the decoding time distributions for I, P, and B pictures for the new design. Table 4 shows the resulting average and worst case picture decoding delays for the three types of pictures, and also the worst case and average decoding delays for all picture types.

The maximum combinatorial delay was determined to be 46 ns, using the same standard-cell library, thus leading to an average decoding delay per picture of 31.5 ms, now below the target delay of 33.3 ms per picture, and to a worst case delay of 36.5 ms. Note, however, that the (relative) gap between the average and the worst case delays for Option 2 has increased significantly with respect to that for Option 1 (see last row of Table 4). Indeed, in the Option 2 design we have increased the decoder performance by introducing some parallelism in the IDCT, PI, and PA Modules. As a result, the relative percentage of time spent on the heavily data dependent VLD+IQ Module, with respect to the total decoding time, has increased significantly, leading to more significant delay variations across pictures.

It is in cases such as the above that the interest of the systematic algorithm for assessing probabilistic constraints proposed in this paper becomes obvious. Indeed, in order to adequately evaluate the suitability of the decoder design under discussion, a key piece of information (to be given to the designer) is the probability that the target delay of 33.3 ms will be exceeded by the particular decoder design. Note that, based on such a probability, and depending on the specific timing requirements of the application for which the MPEG-2 decoder is being developed, the outcome of the evaluation might be radically different. Specifically, the Option 2 design could be considered an adequate solution, could be an unnecessarily expensive solution (in terms of area and/or average power consumption), or could still require further performance improvements. Table 5 shows the probability of violating the decoding time constraint and the Chernoff bound, for I, P and B frames and overall obtained by our algorithm. The exact numbers are exhibited to show the quality of the approximations (upper bound) provided by the algorithm. Based on these numbers,

[1]Such parallelization is quite inexpensive in terms of silicon area, since the computations performed by those two modules are very simple.

the designer would proceed, either by performing yet another iteration at the RTL level, or by starting the physical design of the decoder.

frame type	**Exact**	**Chernoff**
I	$\mathbb{P}(Delay \geq 33.3ms) = 0$	N/A
P	$\mathbb{P}(Delay \geq 33.3ms) = 8.364x10^{-12}$	$\mathbb{P}(Delay \geq 33.3ms) \leq 1.162x10^{-10}$
B	$\mathbb{P}(Delay \geq 33.3ms) = 5.095x10^{-4}$	$\mathbb{P}(Delay \geq 33.3ms) \leq 4.103x10^{-3}$
all	$\mathbb{P}(Delay \geq 33.3ms) = 3.397x10^{-4}$	$\mathbb{P}(Delay \geq 33.3ms) \leq 2.735x10^{-3}$

Table 5: Probabilities of violating decoding time constraint for Option 2, given a 46 ns clock.

6 Conclusions

In this paper we have formulated a probabilistic critical path problem on a DAG with random weights and proposed a novel approximate algorithm for determining the likelihood that a constraint is satisfied. Through a discussion using synthetic and real examples, we have made a case for the importance/relevance of assessing probabilistic constraints on system performance, whenever the application domain is amenable to some level of constraint relaxation. Specifically, the ability to analyze the system model so as to derive less aggressive performance requirements on its various components has the potential to reduce the final cost and power consumption of the system.

Our algorithm is currently being implemented in Sky [15], a tool for assisting algorithm and architecture-level design space exploration during system level design. We also plan to implement the proposed algorithm on a reuse tool currently on its preliminary stage of development.

References

[1] In G. De Micheli and M. Sami, editors, *Hardware/Software Codesign*. Kluwer Academic Publishers, 1996.

[2] V. Bhaskaran, K. Konstantinides, R. Lee, and J. Beck. Algorithmic and architectural enhancements for real-time MPEG-1 decoding on a general purpose risc workstation. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(5):380–86, Oct. 1995.

[3] A. Dembo and O. Zeitouni. *Large Deviations Techniques and Applications*. Jones & Bartlett, Boston, 1992.

[4] D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design Of Embedded Systems*. PTR Prentice Hall, 1994.

[5] R. Guérin and A. Orda. QoS-based routing in networks with inaccurate information: theory and algorithms. *IBM Research Report 20515*, 1996.

[6] R. Gupta. *Co-synthsis of Hardware and Software for Digital Embedded Systems*. Kluwer Academic Publishers, 1995.

[7] R. Gupta and G. De Micheli. Hardware-software cosynthesis for digital systems. *IEEE Design & Test of Computers*, 10(3), 1993.

[8] I. Hsu and J. Walrand. Admission control for ATM networks. In *Proc. IMA Workshop on Stochastic Networks*, 1994.

[9] K. Kavi and B. Bukles. A formal definition of data flow graph models. *IEEE Transactions on Computers*, C-35(11), Nov 1986.

[10] W. Lee and Y. Kim. MPEG-2 video decoding on programmable processors: computational and architectural requirements. In *Proceedings of SPIE*, pages 265–87, 1995.

[11] N. Liu. MPEG decoder architecture for embedded applications. *IEEE Transactions on Consumer Electronics*, 42(4):1021–28, Nov. 1996.

[12] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 1989.

[13] D. Mitra and J. A. Morrison. Multiple time scale regulation and worst case processes for ATM network control. *Proc. of the 34th Conf. on Decision* & *Control*, pages pp. 353–357, 1995.

[14] M. Montgomery and G. de Veciana. On the relevance of time scales in performance oriented traffic modeling. In *Proc. IEEE INFOCOM*, volume 2, pages 513–20, 1996.

[15] H. Peixoto and M. Jacome. Algorithm and architecture level design space exploration using hierarchical data flows. In *Proc. of the 11th Intern. Conf. on Application-specific Systems, Architectures and Processors*, pages 271–82, July 1997.

[16] J. Wilberg, P. Ploger, and R. Camposano et al. Codesign of hardware, software, and algorithms- a case study. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 4, pages 552–55, 1996.

A Algorithm Derivation

The algorithm is based on maximizing an *upper bound* on the probability of failure over paths in P_s. For a fixed $\theta \in \mathbb{R}$, Chernoff's bound gives the following estimate on the probability that a path p, with delay D^p, fails to meet the constraint:

$$\pi_d(p) = \mathbb{P}(D^p \geq d) \leq \exp[-\theta d + \Lambda^p(\theta)],$$

where $\Lambda^p(\theta) = \log \mathbb{E} \exp[\theta D^p]$ is a convex function [3]. Since edge delays are assumed to be independent, it follows that

$$\Lambda^p(\theta) = \log \mathbb{E} \exp[\theta \sum_{e \in p} D_e] = \log \prod_{e \in p} \mathbb{E} \exp[\theta D_e] = \sum_{e \in p} \Lambda_e(\theta),$$

an *additive* metric along the path.

Now maximizing the bound over the paths in P_s we obtain

$$\pi_d(p^*) = \max_{p \in P_s} \pi_d(p) \leq \exp[-\theta d + \max_{p \in P_s} \Lambda^p(\theta)] = \exp[-\theta d + f(\theta)],$$

where $f(\theta) = \max_{p \in P_s} \Lambda^p(\theta)$ is convex since it is a maximum of convex functions. Furthermore, by minimizing over θ, we obtain the tightest such bound, *i.e.,*

$$\pi_d(p^*) \leq \inf_{\theta} \exp[-\theta d + f(\theta)] = \exp[-\sup_{\theta}(\theta d - f(\theta))] = \exp[-f^*(d)], \tag{4}$$

where $f^*(d) = \sup_\theta(\theta d - f(\theta))$, known as the convex dual of $f(\theta)$, is an (extended) convex function [3].

The proposed algorithm is based on determining $f^*(d)$, which in turn requires solving the following, possibly unbounded, convex optimization problem:

$$f^*(d) = \sup_{\theta}(\theta d - f(\theta)). \tag{5}$$

The initialization step ensures that the problem is "well posed" in the sense that, both, the optimization in (5) is bounded, and the delay constraint is meaningful. Standard arguments concerning convex dual functions of random variables lead to the two following requirements, which are used to initialize the algorithm and limit the search space to $\theta \geq 0$ [3][page 27]:

1. If d is greater than the critical path delay for the graph in which edge metrics are the average delays, $\mathbb{E}D_e$, then we need only to optimize over $\theta \geq 0$;

2. The optimization problem is bounded, as long as the delay constraint d can be achieved by some path in the network. In particular, if any edge on a path in P_s has a distribution with unbounded support $\mathbb{R}^+$, *e.g.,* exponential distributions, this will automatically be true. Prior to initiating the optimization one should ensure that d is less than the critical path delay on the graph with weights given by the maximum achievable[2] delay for each edge.

If these conditions are satisfied, (5) is bounded, the resulting maximizer $\hat{\theta}$ is unique, and there is an associated path $\hat{p}$, not necessarily unique, such that $f(\hat{\theta}) = \max_{p \in P_s} \Lambda^p(\hat{\theta}) = \Lambda^{\hat{p}}(\hat{\theta})$. As a result one obtains a guaranteed upper bound (4) and a candidate path $\hat{p}$ for the one most likely fail the constraint.

B Bahadur-Rao and Sensitivity Estimates

Consider the framework suggested in Section 2.3. Suppose that $\Lambda(\theta) = \sum_{c \in C} \frac{n(\hat{p},c)}{n} \Lambda_c(\theta)$ where $n = \sum_{c \in C} n(\hat{p}, c)$, and $\hat{\sigma}^2 = \frac{d^2 \Lambda(\hat{\theta})}{d^2 \theta}$. If the path $\hat{p}$ resulting from the optimization in (5), has a large number of elements n the Bahadhur-Rao estimate [3] gives the first approximation below

$$\mathbb{P}(\sum_{e \in \hat{p}} D_e > d) \approx \frac{1}{\sqrt{2\pi n \hat{\theta}^2 \hat{\sigma}^2}} \exp[-f^*(d)] \approx \frac{1}{\sqrt{4\pi f^*(d)}} \exp[-f^*(d)].$$

The second approximation is a heuristic proposed in [14] requiring no further computation beyond the original exponent obtained from the optimization. In either case the new estimates for $\pi_d(\hat{p})$ are likely to be more accurate, but are no longer guaranteed to be upper bounds for the probability of failure $\pi_d(p^*)$.

It is also of interest to asses changes in the probability of failure upon varying the delay constraint d. To this end one could construct a parametric fit, *e.g.*, quadratic, for the convex function $f^*(\cdot)$ based

[2]We can define this rigorously as, $d_e = \sup\{d | \mathbb{P}(D_e \geq d) > 0\}$.

on evaluating the function at several points. Notice that $f^*(\cdot)$ determines the probability of failure through the exponent, so function approximation errors would translate to even larger errors on the estimates of the failure probability. Nevertheless we believe this can be useful to quickly assess the *sensitivity* of the probability of failure to the system constraint d.