Optimal Priority Assignment Algorithms for Probabilistic Real-Time Systems

Dorin Maxim*, Olivier Buffet*, Luca Santinelli*, Liliana Cucu-Grosjean* and Robert I. Davis♯
* INRIA, Nancy Grand-Est, France
♯ University of York, Real-Time Systems Research Group, United Kingdom
firstname.lastname@inria.fr, rob.davis@cs.york.ac.uk

Abstract

In this paper we investigate the problem of optimal priority assignment in fixed priority pre-emptive single processor systems where tasks have probabilistic execution times. We identify three sub-problems which optimise different metrics related to the probability of deadline failures. For each sub-problem we propose an algorithm that is proved optimal. The first two algorithms are inspired by Audsley's algorithm which is a greedy (lowest priority first) approach that is optimal in the case of tasks with deterministic execution times. Since we prove that such a greedy approach is not optimal for the third sub-problem, we propose a tree search algorithm in this case.

1 Introduction

Across a wide range of industries in the real-time embedded systems domain, including aerospace, automotive electronics, and telecommunications, there is a strong demand for new functionality that can only be met by using advanced high performance microprocessors. Building real-time systems with reliable timing behaviour on such platforms represents a considerable challenge. The wide variability of execution times due to aggressive hardware acceleration features like cache, and deep memory hierarchies means that deterministic approaches to worst-case execution time analysis and worst-case response time analysis can indicate that timing constraints will not be met, when in practice the probability of a deadline miss actually occurring in the lifetime of the system is vanishingly small. In this way, deterministic analysis can lead to significant overprovision in the system architecture, effectively placing an unnecessarily low limit on the amount of new functionality that can be included in a given system.

An alternative approach is to use probabilistic analysis. System reliability is typically expressed in terms of probabilities for hardware failures, memory failures, software faults, etc. This approach also extends to the time domain. For example, the reliability requirements placed on the timing behaviour of a sub-system implemented on an advanced high performance microprocessor might indicate that the timing failure rate must be less than 10^{-9} per hour of operation. Probabilistic analysis techniques that seek to meet this requirement, rather than attempting to provide an absolute guarantee, have the potential to outperform deterministic techniques. In this case, the use of deterministic techniques will inevitably lead to overprovision, and may require the use of advanced hardware features to be limited so that meaningful results can be obtained. Turning off such features has a significant impact on average case performance, degrading quality of service and/or increasing power consumption.

A key problem that needs to be addressed by probabilistic analysis is calculation of the timing failure rate. This can then be compared with reliability requirements to determine if the system is acceptable. With this ultimate goal in mind, this paper examines three simple sub-problems (see Section 2) relating to optimal priority assignment for fixed priority pre-emptive scheduling of uniprocessor real-time systems.

Related work. The problem and solutions we propose in this paper belong to the realm of probabilistic analysis. Recently, the real-time community has been using terms like *stochastic analysis* regardless of the approach (probabilistic or statistical) [15, 21, 4]. In this paper we use the notion of *probabilistic* analysis to indicate that the approach is based on the theory of probability. Moreover, by *probabilistic real-time system* we mean a real-time system with at least one parameter described by a random variable. Therefore probabilistic analysis consists of the temporal analysis of systems that have at least one parameter described by a random variable. Such analyses have been proposed for calculating the response time of tasks [19, 9, 1, 12, 10] under a known scheduling policy.

In this paper we are interested in priority assignment policies for fixed priority pre-emptive scheduling of single processor systems with probabilistic execution times. In particular, we aim to derive priority assignment policies that are *optimal* in the sense that they optimise some metric related to the probability of deadline failures.

In the deterministic case, a priority assignment policy P is referred to as optimal with respect to a given task model, if and only if the following holds: P is optimal if there are no tasksets that are compliant with the task model that are schedulable using another priority assignment policy, that

are not also schedulable using policy P. [5, 6].

In the deterministic case, solutions to the problem of finding an optimal priority assignment are well known. In 1973, Liu and Layland [14] showed that Rate Monotonic (RM) priority ordering is the optimal fixed priority assignment policy for implicit-deadline tasksets. In 1982, Leung and Whitehead [13] showed that Deadline Monotonic (DM) priority ordering is optimal for constrained-deadline tasksets. In 1990, Lehoczky [11] showed that DM is not optimal for tasksets with arbitrary deadlines; however, an optimal priority ordering for such tasksets can be determined, in at most n(n +1) / 2 task schedulability tests, using Audsley's optimal priority assignment algorithm [2, 3].

In this paper, we focus on the problem of finding an optimal priority assignment for real-time systems with probabilistic execution times, scheduled pre-emptively on one processor according to fixed priorities. Initial versions of this problem were presented in [17, 16]. To the best of our knowledge this paper presents the first solution to the problem.

Paper organization. The remainder of the paper is structured as follows. Section 2 presents the task model, terminology and notation used in the rest of the paper. Section 3 defines the three problems we consider, and Section 4 provides preliminary results which are used in their solution. Solutions to each problem are given in Sections 5, 6 and 7. Finally, Section 8 concludes with a brief summary and discussion of possible future work.

2 Task Model, Terminology, and Notation

In this paper, we consider a set Γ of n synchronous constrained deadline periodic tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is characterized by three parameters $(\mathcal{C}_i, T_i, D_i)$ where T_i is the period, D_i the relative deadline, and $\mathcal{C}_i$ the worst-case execution time described by a random variable[1]. We assume a constrained-deadline task model such that $D_i \leq T_i$.

The worst-case execution time of task τ_i is assumed to have a known probability function (PF) $f_{\mathcal{C}_i}(\cdot)$ with $f_{\mathcal{C}_i}(c) = P(\mathcal{C}_i = c)$ giving the probability that τ_i has a computation time equal to c. The values of the worst-case execution time of τ_i are assumed to belong to the interval $[C_i^{\min}, C_i^{\max}]$.

The worst-case execution time $\mathcal{C}_i$ can be written as follows:

$$\mathcal{C}_i = \begin{pmatrix} C_i^0 = C_i^{\min} & C_i^1 & \cdots & C_i^{k_i} = C_i^{\max} \\ f_{\mathcal{C}_i}(C_i^{\min}) & f_{\mathcal{C}_i}(C_i^1) & \cdots & f_{\mathcal{C}_i}(C_i^{\max}) \end{pmatrix}, \tag{1}$$

where $\sum_{j=0}^{k_i} f_{\mathcal{C}_i}(C_i^j) = 1$.

For example for a task τ_i we might have a worst-case execution time $\mathcal{C}_i = \begin{pmatrix} 2 & 3 & 25 \\ 0.5 & 0.45 & 0.05 \end{pmatrix}$; thus $f_{C_i}(2) = 0.5$, $f_{C_i}(3) = 0.45$ and $f_{C_i}(25) = 0.05$.

[1] In this paper we will use a calligraphic typeface to denote random variables.

Each task τ_i generates an infinite number of successive jobs $\tau_{i,j}$, with $j = 1, \ldots, \infty$. All jobs are assumed to be independent of other jobs of the same task and those of other tasks, hence the execution time of a job does not depend on, and is not correlated with, the execution time of any previous job.

The set of tasks is scheduled according to a fixed-priority policy, i.e., all jobs of the same task have the same priority. Let $HP(i)$ to denote the set of tasks with priorities higher than that of task τ_i and $LP(i)$ denote the set of tasks with priorities lower than that of τ_i.

Further, we use Φ to denote a (partial or total) priority assignment defined by a list of tasks ordered from the *lowest* to the *highest* priority. $priority(i)$ is used to denote the priority level of task τ_i. All tasks are assumed to have unique priorities.

The response time of a job is the elapsed time between its release and its completion.Since we consider jobs with probabilistic execution times, the response time of a job is also described by a random variable. Let $\mathcal{R}_{i,j}(\Phi)$ be the response time of job $\tau_{i,j}, \forall i, j$ for a given priority assignment Φ.

In the case of fixed-priority scheduling, the main result providing the calculation of $\mathcal{R}_{i,j}(\Phi), (\forall i, j, \Phi)$ is given by Diaz et al. [8], where the PF of the response time of the j-th job of task τ_i is given by

$$f_{R_{i,j}(\Phi)} = f^{[0,\lambda_{i,j}]}_{R_{i,j}(\Phi)} + (f^{(\lambda_{i,j},\infty)}_{R_{i,j}(\Phi)} \otimes f_{C_i}), \tag{2}$$

where $\lambda_{i,j}$ is the release time of $\tau_{i,j}$. A solution for Equation (2) can be obtained recursively.

Equation (2) can be reformulated as follows:

$$\mathcal{R}_{i,j}(\Phi) = \mathcal{B}_i(\lambda_{i,j}, \Phi) \otimes \mathcal{I}_i(\lambda_{i,j}, \Phi) \otimes \mathcal{C}_i, \tag{3}$$

where $\mathcal{B}_i(\lambda_{i,j}, \Phi)$ is the accumulated *backlog* of higher priority tasks released before $\lambda_{i,j}$ and still active (not completed yet) at $\lambda_{i,j}$. $\mathcal{I}_i(\lambda_{i,j}, \Phi)$ is the sum of the execution times of higher priority tasks arriving after $\lambda_{i,j}$.

For a task τ_i, a time interval $[a, b]$ and a priority assignment Φ, the response time $\mathcal{R}_i^{[a,b]}(\Phi)$ of τ_i is obtained from the calculation of its PF by averaging the job response times over the interval $[a, b]$:

$$f_{\mathcal{R}_i^{[a,b]}(\Phi)} = \frac{1}{n_{[a,b]}} \sum_{j=1}^{n_{[a,b]}} f_{\mathcal{R}_{i,j}(\Phi)}, \tag{4}$$

with $n_{[a,b]} = \lceil \frac{b-a}{T_i} \rceil$ giving the number of jobs belonging to τ_i released in the interval.

3 Problems

In this paper we consider the problem of finding a priority assignment such that the associated schedule meets a requirement specifying the maximum acceptable probability of timing (i.e. deadline) failure within a given time interval. We refer to a priority ordering that meets such

a requirement as a *feasible* priority assignment. We note that this use of the term feasible is an extension of its normal use in the deterministic case, where a feasible priority ordering is one in which the associated schedule has zero probability of timing failure.

Before defining the sub-problems we deal with, we first introduce some additional notation and definitions.

Definition 1 (Job deadline miss)**.** *For a job $\tau_{i,j}$ and a priority assignment Φ, the deadline miss probability* $\text{DMP}_{i,j}$ *is the probability that the j-th job of task τ_i misses its deadline:*

$$\text{DMP}_{i,j}(\Phi) = P(\mathcal{R}_{i,j}(\Phi) > D_i). \tag{5}$$

Definition 2 (Task deadline miss ratio)**.** *For a task τ_i, a time interval $[a, b]$ and a priority assignment Φ, the task deadline miss ratio is computed as follows*

$$\begin{aligned}\text{DMR}_i(a, b, \Phi) &= \frac{P(\mathcal{R}_i^{[a,b]}(\Phi) > D_i)}{n_{[a,b]}} \\ &= \frac{1}{n_{[a,b]}} \sum_{j=1}^{n_{[a,b]}} \text{DMP}_{i,j}(\Phi),\end{aligned} \tag{6}$$

where $n_{[a,b]} = \lceil \frac{b-a}{T_i} \rceil$ is the number of jobs of task τ_i released during the interval $[a, b]$.

In this paper we consider that at the end of each hyperperiod any incomplete job is aborted. This is a form of reset prior to synchronously releasing all of the tasks at the start of the next hyperperiod. In future work we aim to relax this assumption and move towards a more general case.

The consequence of resetting the system after each hyperperiod is that we can focus our analysis on the first hyperperiod $[0, H]$ (with $H = \text{lcm}\{T_1, T_2, \cdots, T_n\}$) which is a feasibility interval. During the first hyperperiod of a schedule, i.e., $a = 0$ and $b = H$, the task deadline miss ratio is denoted by $\text{DMR}_i(\Phi)$ for a priority assignment Φ and a task τ_i. Unless specified otherwise, in the following we consider the interval $[a, b] = [0, H]$.

The DMP gives the probability that a job misses its deadline, whereas the DMR gives the average probability over the hyperperiod that a job of the task will miss its deadline, and hence the average probability over the entire lifetime of the system, that a job of the task will miss its deadline.

For each task τ_i we consider a specified parameter $p_i \in [0, 1]$, referred to as the maximum permitted deadline miss ratio.

We define three new problems:

1. Basic Priority Assignment Problem (**BPAP**). This problem involves finding a priority assignment such that the DMR of every task does not exceed the threshold specified, i.e. $\text{DMR}_i(\Phi) \leq p_i$. Hence, we search for a feasible priority assignment Φ^* such that $\text{DMR}_i(\Phi^*) \leq p_i, \forall i$.

2. Minimization of the Maximum Priority Assignment Problem (**MPAP**). This problem involves finding a priority assignment that minimizes the maximum deadline miss ratio of any task. Hence, we search for a priority assignment Φ^* such that $\max_i\{\text{DMR}_i(a, b, \Phi^*)\} = \min_\Phi\{\max_i \text{DMR}_i(a, b, \Phi)\}$.

3. Average Priority Assignment Problem (**APAP**). This problem involves finding a priority assignment that minimizes the sum of the deadline miss ratios for all tasks. Hence, we search for a feasible priority assignment Φ^* such that $\sum_i \text{DMR}_i(a, b, \Phi^*) = \min_\Phi\{\sum_i \text{DMR}_i(a, b, \Phi)\}$.

In the following we use the relationship $\succeq$ as it is defined in [8], summarized as follows. A random variable $\mathcal{X}$ is *greater than or equal to* another random variable $\mathcal{Y}$, noted $\mathcal{X} \succeq \mathcal{Y}$ if and only if $\forall V\ P(\mathcal{Y} \leq V) \leq P(\mathcal{X} \leq V)$. The relationship $\succeq$ between two probabilistic response times defines a probabilistic order among probabilistic variables and allows us to compare random variables.

To motivate our work we prove in Section 3.1 that Rate Monotonic (RM) priority ordering is not optimal for **BPAP**.

3.1 Non-optimality of Rate Monotonic priority ordering

Rate monotonic priority assignment [14] is optimal for the deterministic problem of scheduling implicit deadline tasksets according to a fixed-priority policy. We therefore analyzed the applicability and optimality of RM to BPAP. Below, we present a counter example, a case where there exists a feasible priority assignment for BPAP, but RM priority assignment is not feasible.

Let $\Gamma = \{\tau_1, \tau_2\}$ be a taskset such that each task is characterized by the parameters $(\mathcal{C}, T, D, p)$. We have τ_1 defined by $(\begin{pmatrix} 2 & 3 \\ 0.5 & 0.5 \end{pmatrix}, 4, 4, 0.5)$ and τ_2 defined by $(\begin{pmatrix} 2 & 3 \\ 0.5 & 0.5 \end{pmatrix}, 8, 8, 0.1)$.

According to RM priority assignment, τ_1 has the higher priority and τ_2 the lower priority. In this case the response time of task τ_1 is $\mathcal{R}_1 = \begin{pmatrix} 2 & 3 \\ 0.5 & 0.5 \end{pmatrix}$ and the response time of task τ_2 is $\mathcal{R}_2 = \begin{pmatrix} 4 & 7 & 8 & D_2^+ \\ 0.25 & 0.25 & 0.375 & 0.125 \end{pmatrix}$

Table 1 summarizes the computation of the response time of task τ_2. The higher priority task, τ_1, has two jobs released in the hyperperiod, while the lower priority task, τ_2, has only one job released in the hyperperiod.

If jobs $\tau_{2,1}$ and $\tau_{1,1}$ have execution times equal to 2, then job $\tau_{2,1}$ is not pre-empted by the second job of τ_1 and it has a response time equal to 4. In the other three cases job $\tau_{2,1}$ is pre-empted and its response time is further increased. In these cases, only if all three jobs have execution times equal to 3 does job $\tau_{2,1}$ miss its deadline,

		$\mathcal{T}_{1,1}$		$\mathcal{T}_{1,2}$			
		2	3	2		3	
		0.5	0.5	0.5		0.5	
$\mathcal{T}_{2,1}$	2	4	5	7		8	
	0.5	0.25	0.25	0.125		0.125	
	3	5	6	7	8	8	D_2^+
	0.5	0.25	0.25	0.125	0.125	0.125	0.125

Table 1. The response time of τ_2 when τ_2 is the lowest priority task

with a probability of 0.125. We indicate by D_2^+ that the response time values are larger than the deadline.

Given that the deadline miss ratio of task τ_2 is $\text{DMR}_2 = 0.125$, which is greater than the maximum permitted value, $p_2 = 0.1$, then RM priority assignment is not feasible for this taskset with respect to the specified constraints.

Next we consider the alternative priority ordering where task τ_2 has a higher priority than task τ_1. For this priority ordering, the response time of task τ_2 is given by $\mathcal{R}_2 = \begin{pmatrix} 2 & 3 \\ 0.5 & 0.5 \end{pmatrix}$.

Task τ_1 has two jobs released in the hyperperiod $[0, 8)$. The response time of its first job is $\mathcal{R}_{1,1} = \begin{pmatrix} 4 & 5 & 6 \\ 0.25 & 0.5 & 0.25 \end{pmatrix}$, giving a deadline miss probability of $DMP_{1,1} = 0.75$.

Its second job has a response time of $\mathcal{R}_{1,2} = \begin{pmatrix} 2 & 3 & 4 & D_1^+ \\ 0.125 & 0.375 & 0.375 & 0.125 \end{pmatrix}$, giving a deadline miss probability of 0.125.

Combining the response times of the two jobs we get a response time for task τ_1 of $\mathcal{R}_1 = \begin{pmatrix} 2 & 3 & 4 & D_1^+ \\ 0.0625 & 0.1875 & 0.3125 & 0.4375 \end{pmatrix}$ and hence a deadline miss ratio of $\text{DMR}_1 = 0.4375$ which is less than the maximum permitted value of $p_1 = 0.5$. Thus the priority assignment is a feasible one, with all tasks having deadline miss ratios that are less than their maximum permitted values.

As RM priority assignment is not feasible for this taskset with respect to the specified constraints, and there exists an alternative priority ordering that is feasible, then RM is not an optimal priority assignment policy for the Basic Priority Assignment Problem (BPAP).

4 Preliminary results

Before providing solutions to the problems described in Section 2 we first state some preliminary theorems and results.

4.1 Order of higher priority tasks

Theorem 1 (Order of higher priority tasks)**.** *Let $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n constrained-deadline periodic tasks with probabilistic execution times scheduled pre-emptively according to a fixed-priority algorithm on a single processor. If a task τ_i has its priority known and given, then the priority order of the higher priority tasks (belonging to $HP(i)$) does not impact the value of $\text{DMP}_{i,j}(\Phi)$ for any job of τ_i or the value of $\text{DMR}_i(a, b, \Phi), \forall a, b$. Stated otherwise, if membership of the sets $HP(i)$ and $LP(i)$ are unchanged, then the response time $\mathcal{R}_{i,j}$ of any job of τ_i is unchanged and the response time $\mathcal{R}_i^{[a,b]}$ of task τ_i is unchanged whatever the priority order of tasks within $HP(i)$ and within $LP(i)$.*

Proof. Concerning the lower priority tasks, with pre-emptive scheduling, the tasks in $LP(i)$ do not influence the response time of task τ_i (or of that of any job $\tau_{i,j}$ of task τ_i), whatever their priority order.

Let us consider now the case of higher priority tasks. Let $\tau_{i,j}$ be the job of the task τ_i that is released at time t. According to Equation (3) the response time $\mathcal{R}_{i,j}$ of $\tau_{i,j}$ is obtained by adding to the existing backlog $\mathcal{B}_i$ its own execution time and the sum of execution times of all higher priority jobs that arrive after its release.

Since the backlog $\mathcal{B}_i$ at time t represents the execution times of the higher priority tasks that have been released before time t (including those that have been released at time t) and that have not been completed yet, its value does not depend on the priority order of the higher priority tasks.

In a time interval of length t, any task τ_k can have only $\lceil t/T_k \rceil$ releases which are relevant to the calculation of the response time of task τ_i, whatever its priority. This is true for all the tasks that have higher priority than the task for which we are computing the response time. Furthermore, the same reasoning is valid for the number of pre-emptions, i.e., the number of pre-emptions from a task with priority higher than τ_i is the same, regardless of whether that task has the highest priority or any priority higher than that of τ_i. Moreover, since the summation of two random variables is commutative, it follows that the response time of task τ_i is the same whatever the priorities we assign to the rest of the tasks in $HP(i)$. □

4.2 Monotonicity of the response time

Theorem 2 (Monotonicity of the response time)**.** *Let $\Gamma = \{\tau_1, \ldots, \tau_n\}$ be a set of constrained-deadline tasks with probabilistic execution times, scheduled pre-emptively according to a fixed priority algorithm. Recall that Φ is a priority assignment that may be represented by a list of*

tasks in sequence from lowest to highest priority. Let Φ_1 and Φ_2 be two priority assignments such that, to get from one assignment to the other, one only has to change the position of one unique task τ_i in the list, leaving the relative order of all other tasks unchanged. If the priority of τ_i is lower in Φ_1 than in Φ_2, then the response time of any of its jobs is such that $\mathcal{R}_{i,j}(\Phi_1) \succeq \mathcal{R}_{i,j}(\Phi_2)$. Consequently, the task response time $\mathcal{R}_i^{[a,b]}(\Phi_1) \succeq \mathcal{R}_i^{[a,b]}(\Phi_2)$.

Proof. Follows from the fact that the backlog and the interference increase with decreasing priority of the task, and consequently increasing priority of other tasks with respect to τ_i.

More precisely for each job in the interval $[a, b)$, we compare the response time distributions and the way they are derived in the two cases. With priority assignment Φ_1 (where task τ_i is at lower priority) τ_i has backlog and interference that is no smaller than with priority assignment Φ_2. This is because all of the tasks that are at higher priority than task τ_i in Φ_2 are also at higher priority than τ_i in Φ_1. Hence, from Equation (3) the response time of any job of task τ_i cannot increase when the priority assignment is changed from Φ_1 to Φ_2. Further, from Equation (4) neither can the response time of task τ_i □

The response time of a task increases with respect to the priority of the task. The lower the priority level, the more the interference from higher priority tasks, and so the larger the response time. The monotonicity of the task response time leads to a similar conclusion about the deadline miss probability (DMP).

Corollary 1 (Monotonicity of DMP and DMR)**.** *Let $\Gamma = \{\tau_1, \ldots, \tau_n\}$ be a set of constrained-deadline tasks with probabilistic execution times, scheduled pre-emptively according to a fixed priority algorithm. Let Φ_1 and Φ_2 be two priority assignments such that, to get from one assignment to the other, one only has to change the position of one unique task τ_i in the list (priority sequence), leaving the relative order of all other tasks unchanged. If the priority of τ_i is lower in Φ_1 than in Φ_2, then $\text{DMP}_{i,j}(\Phi_1) \geq \text{DMP}_{i,j}(\Phi_2)$ and $\text{DMR}_i(a, b, \Phi_1) \geq \text{DMR}_i(a, b, \Phi_2)$.*

Proof. Under the condition stated in Theorem 2 the response time of the generic task τ_i cannot decrease with a decrease in its relative priority. Under the same condition, since $\text{DMP}_{i,j}(\Phi) = P(\mathcal{R}_{i,j}(\Phi) > D_i)$ (Equation (5)) and $\text{DMR}_i(a, b, \Phi) = \frac{P(\mathcal{R}_i^{[a,b]}(\Phi) > D_i)}{n_{[a,b]}}$ (Equation (6)), the job deadline miss probability and the task deadline miss ratio cannot decrease with a decrease in the task's relative priority. □

5 Solution for BPAP

In this section we present an optimal algorithm that solves the Basic Priority Assignment Problem (BPAP) as defined in Section 2. First we provide a definition of what we mean by an optimal priority assignment in this case.

Definition 3 (Optimal algorithms for BPAP)**.** *Let Γ be a set of constrained-deadline tasks with probabilistic execution times and each task characterized by paramters (C_i, T_i, D_i, p_i). An algorithm is optimal with respect to BPAP if, for any arbitrary taskset Γ, it always finds a feasible priority assignment whenever such an assignment exists.*

5.1 The intuitive (but incorrect) solution for BPAP

At first glance, one might think that an optimal priority assignment for the Basic Priority Assignment Problem could be obtained by assigning priorities to tasks in increasing order of their maximum permitted deadline miss ratios, (i.e. the lower the maximum permitted deadline miss ratio, the higher the priority).

Below, we provide a counter example showing that this priority assignment policy is not optimal for BPAP.

Consider a taskset $\Gamma = \{\tau_1, \tau_2\}$ such that τ_1 is defined by $(\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0.2 & 0.3 & 0.3 & 0.2 \end{pmatrix}, 5, 5, 0.4)$ and τ_2 by $(\begin{pmatrix} 4 \\ 1 \end{pmatrix}, 10, 10, 0.2)$.

Assuming that τ_2, which has the smaller permitted deadline miss ratio, is assigned the higher priority, then we have the following response times for the two jobs of task τ_1:

$$\mathcal{R}_{1,1} = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 0.2 & 0.3 & 0.3 & 0.2 \end{pmatrix}$$

$$\mathcal{R}_{1,2} = \begin{pmatrix} 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 0.04 & 0.12 & 0.21 & 0.26 & 0.21 & 0.12 & 0.04 \end{pmatrix}.$$

The deadline miss ratio of task τ_1 is given by $\text{DMR}_1 = \frac{\text{DMP}_{1,1}+\text{DMP}1,2}{2} = \frac{0.8+0.16}{2} = \frac{0.96}{2} = 0.48$ which is higher than its maximum permitted deadline miss ratio of 0.4.

Alternatively, if τ_1 is assigned the higher priority, then τ_1 would have $\text{DMR}1 = 0$, and τ_2 would have a response time of $\mathcal{R}_2 = \begin{pmatrix} 5 & 7 & 8 & 9 & 10 & 11 & 12 \\ 0.2 & 0.06 & 0.15 & 0.22 & 0.21 & 0.12 & 0.04 \end{pmatrix}$ and $\text{DMR}2 = 0.16$ which is less then its maximum allowed DMR of 0.2.

This example shows that there exists tasksets and constraints on their maximum permitted deadline miss ratios that are not feasible when priorities are assigned according to each task's maximum permitted deadline miss ratio, and yet are feasible with an alternative priority assignment. Hence assigning priorities based on the maximum permitted deadline miss ratio of each task is not an optimal solution for BPAP.

5.2 A first optimal solution for BPAP

Theorem 1 indicates that the order of higher priority tasks does not impact the response time of a task. This result suggests that a greedy approach similar to Audsley's optimal priority assignment algorithm [2, 3] can be used to solve BPAP.

Theorem 1 means that we can assign priorities to tasks in a greedy fashion, starting at the lowest priority level

and continuing up to the highest priority level. While assigning task τ_i to a particular priority level, we need not be concerned about the tasks that have already been assigned to lower priority levels (tasks in $LP(i)$) as their response times will be unchanged, and we do not need to know the priority assignment for the remaining tasks, since this does not impact the response time of task τ_i as all of those tasks will be in $HP(i)$ whatever their relative priority ordering.

Since the maximum permitted deadline miss ratio p_i for each task τ_i is specified as part of the problem description, we can use this information within the priority assignment algorithm to determine whether or not a task is feasible at a particular priority level or if it needs to occupy a higher one (see Algorithm 1).

To find a task to place at the next available priority level k, Algorithm 1 checks each unassigned task in turn to see if its maximum permitted deadline ratio would be exceeded if it were placed at priority level k, if not, then the task is assigned to that priority level, and the algorithm continues to the next higher priority. If it is exceeded, then that task needs to be placed at a higher priority level, and the algorithm continues to check other unassigned tasks to see if any of them can be assigned to priority level k.

Note, that when computing the deadline miss ratio for a task at priority level k we do not need to assign priorities to the unassigned tasks that will occupy the priority levels above priority k, any assignment can be assumed, since this will not affect the response time of the task placed at priority k.

Algorithm 1 is similar to Audsley's algorithm [2] and does not require backtracking. Indeed, once we have assigned a subset of tasks to lower priority levels where they do not exceed their maximum allowed deadline miss ratio p, and we are now seeking a task to place at priority level k, if none of the tasks left unassigned can be assigned to this priority level, then it would be useless to backtrack and try and move any of the unassigned tasks to a lower priority level, where it would have an even higher DMR. Since none of the tasks left unassigned can be placed at priority level k without exceeding their maximum permitted deadline miss ratios the tasks set is infeasible, and no priority assignment exists such that all of the tasks can meet the requirements placed on their deadline miss ratios.

Theorem 3. *Algorithm 1 is optimal with respect to BPAP.*

Proof. The proof follows directly from the description of Algorithm 1, Theorem 1 and Definition 3. □

Algorithm 1: Solution to BPAP: the *feasibility* function verifies that for $\forall \tau_i$, $DMR_i < p_i$

Input: $\Gamma = \{\tau_i, i \in 1..n\}$ /* source set of tasks */
Output: Φ /* destination sequence of tasks */

```
Φ ← ()
for l ∈ n..1 do
    assignment ← FALSE
    for τ_i ∈ Γ' do
        /* feasibility function such that the computed DMR_i < p_i */
        if feasible(τ_i, Φ) then
            Φ ← Φ.τ_i
            Γ' ← Γ'\{τ_i}
            assignment ← TRUE
            break
    if assignment = FALSE then
        /* could not find a task to put at this priority level */
        break
```

6 Solution for MPAP

6.1 The Lazy and Greedy Algorithm

For MPAP we propose a *Lazy and Greedy Algorithm* (LGA). It is inspired by Audsley's optimal priority assignment algorithm [2]. Indeed, Algorithm 2 incrementally builds a solution Φ as a sequence of tasks, starting with the lowest priority first, and adding to Φ at each iteration an unassigned task. This algorithm is:

- *lazy* because, if there exists an unassigned task τ which could be used at the current priority level l without degrading the worst DMR, then τ is assigned to this priority level, and
- *greedy* because, if no such non-degrading task exists, then it assigns to the current priority level the task that would have the smallest deadline miss ratio, without reasoning about what will happen at higher priority levels.

Algorithm 2's main loop goes through the priority levels in order: $l \in n..1$. At each iteration, it performs a for loop over the unassigned tasks to search for the one (τ_{best}) that has the best DMR (DMR_{best}) at the current priority level l (this is the greedy part). This requires computing, for each unassigned task τ_i, its potential DMR δ (if τ_i is not in Φ, $\text{DMR}_i(\Phi)$ is the DMR of τ_i if appended at the end of Φ) and comparing it to the current most promising task. Further, each unassigned task's DMR is compared with the worst DMR over all of the tasks already in Φ (referred to as DMR_{worst}) so that, whenever a task is found with a deadline miss ratio better than or equal to DMR_{worst}, the search is cancelled and this task is assigned (this is the lazy part).

6.2 Optimality of the Lazy and Greedy algorithm

Before proceeding to show that the Lazy and Greedy algorithm provides an optimal solution to MPAP, we must first define what we mean by optimality in this case.

Definition 4 (Optimal algorithms for MPAP)**.** *Let Γ be a set of constrained-deadline tasks with probabilistic execution times, where each task is characterized by the parameters $(\mathcal{C}_i, T_i, D_i)$. A priority assignment algorithm P is optimal with respect to MPAP if the priority ordering determined by algorithm P, for any arbitrary taskset Γ, has*

Algorithm 2: Lazy and Greedy Algorithm

Input: $\Gamma = \{\tau_i, i \in 1..n\}$ /* source set of tasks */
Output: Φ /* sequence of tasks */, DMR_{worst} /* worst DMR */

$\Phi \leftarrow ()$
$\text{DMR}_{worst} \leftarrow 0$
/* Loop over the priority levels (from lowest to highest) */
for $l \in n..1$ **do**
  /* Search among unassigned tasks */
  $(\tau_{best}, \text{DMR}_{best}) \leftarrow (0, +\infty)$
  for $\tau_i \in \Gamma$ **do**
    /* Compute DMR of current task τ_i */
    $\delta \leftarrow \text{DMR}_i(\Phi)$
    /* If this DMR is better than (or equal to) the current worst DMR in Φ, be lazy: pick this task and stop the search. */
    if $\delta \leq \text{DMR}_{worst}$ **then**
      $(\tau_{best}, \text{DMR}_{best}) \leftarrow (\tau_i, \delta)$
      break
    /* If this DMR improves on other unassigned tasks, remember this task. */
    if $\delta < \text{DMR}_{best}$ **then**
      $(\tau_{best}, \text{DMR}_{best}) \leftarrow (\tau_i, \delta)$
  /* The search is done. The task in τ_{best} can be assigned at the current priority level. */
  $\Gamma \leftarrow \Gamma \backslash \{\tau_{best}\}$
  $\Phi \leftarrow \Phi.\tau_{best}$
  /* Update the value of the worst DMR in Φ. */
  if $\text{DMR}_{worst} < \text{DMR}_{best}$ **then**
    $\text{DMR}_{worst} \leftarrow \text{DMR}_{best}$
return $(\Phi, \text{DMR}_{worst})$

a maximum deadline miss ratio for any task, which is no larger than that obtained with any other priority ordering.

Theorem 4. *The Lazy and Greedy Algorithm (LGA) is optimal with respect to MPAP.*

Proof. Let us consider a problem with n tasks $\tau_1, \ldots, \tau_n$. Let Φ^g be the solution returned by the Lazy and Greedy Algorithm (Φ^g is a permutation over $1..n$, and $\Phi^g(i)$ is the index of the i-th task by order of priority).

Let M be the set of positions in Φ^g corresponding to items with the maximum deadline miss ratio: $M = \arg\max_i \text{DMR}_{\Phi^g(i)}(\Phi^g) = \{m_1, \ldots, m_{|M|}\}$, as illustrated in Table 2. A first observation is that this set contains at least one item (but possibly more). Let us also assume that these indices are ordered as they appear in Φ^g, meaning that they verify: $m_1 > m_2 > \cdots > m_{|M|}$ (as the first tasks introduced have low priorities). Due to the algorithm, when the first of these tasks ($\tau_{\Phi^g(i)}$) is added to M ($m_1 = i$), each remaining task has a deadline miss ratio (DMR) at that priority level that is greater than or equal to that of $\tau_{\Phi^g(m_1)}$, but it will subsequently become equal or better, once the task is assigned a higher priority level. Note also that there is no *a priori* reason for these "worse tasks" to be contiguous.

Let Φ' be any other complete solution, and let us refer to:

	(low P.) ←			→ (high P.)
tasks	$\tau_{\Phi^g(4)}$	$\tau_{\Phi^g(3)}$	$\tau_{\Phi^g(2)}$	$\tau_{\Phi^g(1)}$
DMR	13	22	17	22
M	–	$m_1 = 3$	–	$m_2 = 1$

Table 2. Example fixed priority assignment with associated DMRs and M set

- $\tau_c = \tau_{\Phi^g(m_1)}$ as the *critical task* of Φ, i.e., the lowest priority task in Φ that has the maximum or worst DMR for any task in priority order Φ;
- DMR_c^g as the DMR of τ_c in Φ^g;
- $HP(\tau, \Phi)$ as the set of tasks with higher priorities than τ in Φ (Φ omitted when using the current permutation);
- $LP(\tau, \Phi)$ as the set of tasks with lower priorities than τ in Φ;
- τ_b as the task in $HP(\tau_c, \Phi^g)$ which has the lowest priority in Φ'; this will be our "bad" task in the proof;
- DMR_b' as the DMR of τ_b in Φ'.

To show that Φ' has a "worst DMR" that is no smaller than that of Φ^g, we now describe a transform that turns Φ^g into Φ' while also demonstrating that DMR_b' is greater than or equal to DMR_c^g. This transform is illustrated in Table 3, with numbers indicating tasks, and with tasks that will be moved underlined. The transformation proceeds as follows (starting from Φ^g):

1. lower τ_b to the position just above τ_c without altering the relative priority order of the other tasks; this only alters the response times, and thus the DMRs, of higher priority tasks, and can only improve these values;
2. swap the priorities of tasks τ_c and τ_b; task τ_b then has a DMR greater than or equal to DMR_c^g (otherwise the Lazy and Greedy Algorithm would have selected it—or another task—instead of τ_c);
3. permute the tasks in $LP(\tau_b)$ so that they are in the same relative priority order as in Φ'; this does not alter the response time, and thus the DMR, of task τ_b;
4. lower the priority of task τ_b to place it at its appropriate position among the tasks in $LP(\tau_b)$ as defined by the priority ordering Φ'; this cannot make τ_b's DMR better;
5. permute the tasks in $HP(\tau_b)$ to finally obtain Φ'; this last operation does not alter τ_b's DMR.

Following this process guarantees that, in Φ', at least one task (τ_b) has a DMR greater than or equal to DMR_c^g. This concludes the proof that the priority assignment found by the Lazy and Greedy Algorithm is optimal for MPAP □

	(low P.) ←			→ (high P.)			
$\Phi^g =$	6	5	4	**3**	2	*1*	
	6	5	4	**3**	*1*	2	(1. lower τ_b)
	6	5	4	*1*	**3**	2	(2. swap τ_c and τ_b)
	4	5	6	*1*	**3**	2	(3. permute $LP(\tau_b)$)
	4	*1*	5	6	**3**	2	(4. lower τ_b)
$\Phi' =$	4	*1*	**3**	5	6	2	(5. permute $HP(\tau_b)$)

Table 3. Example process leading from a permutation Φ^g to another one Φ' with τ_c=task 3 and τ_b=task *1*.

Complexity Let us consider the non-lazy version of LGA, i.e., not picking a task just because its DMR is better than the current worst DMR in Φ. Then, numbering iterations from n to 1 (reflecting the number of unassigned tasks), iteration i has to compute the DMRs of i tasks and choose the best of these i tasks to assign. Assuming that a DMR is computed with a fixed cost, this means that iteration i has complexity $O(i)$ and that the complete algorithm has a complexity of $O(n(n+1)/2)$ DMR computations.

In the normal (lazy) version of LGA, many searches for the best task to choose will be interrupted early, which may lead to significant speed-ups. In the best case, the algorithm has complexity $O(n)$ DMR computations.

7 Solutions for APAP

The max criterion used in MPAP can be seen as pessimistic as it optimizes the worst DMR among all tasks. A different objective is to optimize the *average* DMR over all tasks, which is equivalent to using a sum operator instead of max. The sum and max criteria can be written as:

$$g(\Phi) = \bigoplus_{\tau_i \in \Phi} \text{DMR}_i(\Phi), \tag{7}$$

where $\bigoplus$ represents either the sum ($\sum$) or the max operator.

This section investigates solutions to APAP (i.e. the optimization of the summation criterion). First we provide a definition of what we mean by an optimal solution in this case.

Definition 5 (Optimal algorithms for APAP)**.** *Let Γ be a set of constrained-deadline tasks with probabilistic execution times, where each task is characterized by parameters $(\mathcal{C}_i, T_i, D_i)$. A priority assignment algorithm P is optimal with respect to APAP if the priority ordering determined by algorithm P, for any arbitrary taskset Γ, has a value for the metric $g(\Phi)$ (as defined by Equation (7)), which is no larger than that obtained with any other priority ordering.*

7.1 When Being Greedy is not the Best Choice

A first question is whether the Lazy and Greedy Algorithm produces an optimal solution for APAP. As we will now see, a simple example demonstrates that this is not the case.

	$\tau_2 > \tau_1$	$\tau_1 > \tau_2$
DMR_1	0.5	0.85
DMR_2	0.6	0
sum	1.1	0.85

Table 4. Results for Example 1 depending on the task priorities.

Theorem 5. *The Lazy and Greedy Algorithm may return a solution with a non-optimal average* DMR.

The above theorem is proved by the following counter example, using a time interval $[a, b]$ corresponding to one hyperperiod.

Example 1 (counter-example for the sum criterion)**.** *Let $\tau = \{\tau_1, \tau_2\}$ with each task characterized by parameters $(\mathcal{C}_i, T_i, D_i)$. We consider τ_1 defined by $(\begin{pmatrix} 1 & 3 \\ 0.5 & 0.5 \end{pmatrix}, 4, 2)$ and τ_2 defined by $(\begin{pmatrix} 1 & 2 & 4 \\ 0.3 & 0.2 & 0.5 \end{pmatrix}, 4, 4)$.*

For this example, we can compute the response times, the DMRs, and the values of the sum criterion for the two possible Priority orderings:

- $\tau_2 > \tau_1$:

$$\mathcal{R}_1 = \begin{pmatrix} 1 & D_1^+ \\ 0.5 & 0.5 \end{pmatrix}$$

$$\mathcal{R}_2 = \begin{pmatrix} 2 & 3 & 4 & D_2^+ \\ 0.15 & 0.1 & 0.15 & 0.6 \end{pmatrix}$$

- $\tau_1 > \tau_2$:

$$\mathcal{R}_2 = \begin{pmatrix} 1 & 2 & 4 \\ 0.3 & 0.2 & 0.5 \end{pmatrix}$$

$$\mathcal{R}_1 = \begin{pmatrix} 2 & D_1^+ \\ 0.15 & 0.85 \end{pmatrix}$$

Table 4 gives the DMR of each task and the value of the sum criterion depending on the priority assignment.

Algorithm 2 would pick task τ_2 to be at the lower priority level, since with that priority it would have a deadline miss ratio of $\text{DMR}_2 = 0.6$, which is less than the deadline miss ratio of $DMR_1 = 0.85$ that task τ_1 would have if it were assigned that priority.

This priority assignment would be a good solution if we tried to minimize the maximum DMR, but when we look at the sum of the two deadline miss ratios, we see that, with task τ_1 at the higher priority and task τ_2 at the lower priority, the sum of the deadline miss ratios is 1.1. However, if instead we reverse the priority order, we obtain a sum of the deadline miss ratios of 0.85, which is less than the solution provided by Algorithm 2, thus proving that Algorithm 2 is not an optimal algorithm for APAP.

7.2 Tree Search algorithm

As the Lazy and Greedy Algorithm is not optimal, we need to resort to a different approach.

To optimize $g(\Phi) = \sum_i \text{DMR}_i(\Phi)$, a simple approach is to use a tree search algorithm enumerating all solutions. Among various possible tree search algorithms, we choose here Depth-First Search (DFS), which explores each branch as far as possible before backtracking. As in previous algorithms, we start with the lowest priority, extending the partial priority ordering Φ progressively as we go down the tree. The partial assignment Φ obtained in any node of the tree gives a lower bound for any complete assignment Φ' containing it ($\Phi \subseteq \Phi'$ and $|\Phi'| = n$):

$$\begin{aligned} g(\Phi) &= \sum_{\tau_i \in \Phi} \text{DMR}_i(\Phi) = \sum_{\tau_i \in \Phi} \text{DMR}_i(\Phi') \\ &\leq \sum_{\tau_i \in \Phi'} \text{DMR}_i(\Phi') = g(\Phi'). \end{aligned}$$

This allows for:

- incrementally refining the lower bound while going down the tree: $g(\Phi.\tau_i) = g(\Phi) + \text{DMR}_i(\Phi)$; and
- pruning branches when the current lower bound is higher than the best complete solution found so far (variable g^{best} initialized to $+\infty$).

The resulting process is detailed in Algorithm 3.

Because of the different criteria optimized in APAP, one cannot be as lazy as in LGA. Nevertheless, if a task is encountered with a DMR of zero, then the search loop can also be interrupted early.

Complexity As in any tree search algorithm, the worst-case scenario is when all candidate solutions have to be visited, which means a complexity of $O(n!)$ (again assuming a fixed cost for computing a task's response time and thus a DMR).

Improvement Other natural candidate algorithms are best-first search algorithms, e.g., A* or IDA* [18]. In all cases an important point would be to improve the lower bounds by using an admissible heuristic which can be obtained:

- by solving all problems with, e.g., 2 or 3 tasks; but the cost of these pre-computations could be detrimental;
- by using Algorithm 2, which will presumably return a first good candidate solution to complete the current partial priority assignment; or
- by using their current DMRs to decide in which order to try candidate tasks.

Note that there is no point trying to refine this search algorithm as long as one does not know how problems are distributed. Indeed, as stated by the no-free-lunch theorem(s) for search [20], there is no "best" overall algorithm. Given two algorithms A and B, one will be better than the other on a subset of problems.

One could also exploit the fact that, if Φ and Φ' are partial and contain the same tasks, then one only needs to develop the subtree for the one with the lowest value.

Finally, a reversed search (setting highest priority tasks first) would give a simple admissible—but not necessarily very informative—heuristic by computing the deadline miss ratio of all remaining tasks as if each were the next task.

Tree-search algorithms can be used with other criteria, for example to optimize the mean squared DMR with $g(\Phi) = \sum_{\tau_i \in \Phi} \text{DMR}_i^2(\Phi)$, which provides an intermediate objective between the worst and the average criteria.

Algorithm 3: Depth-First Search

```
f_best = +∞ /* best value so far (glob. var) */
Γ = {τ_i, i ∈ 1..n} /* source set of tasks */
(Φ, g) ← RECUR(Γ, (), n, 0)
return (Φ, g)

/* Function recursively completing the current solution Φ. */
RECUR(Γ, Φ, l, g) /* Note: g = g(Φ) */
/* If priority level 0 is attained, we have a complete solution. */
if l = 0 then
    /* Is this solution the new best solution? */
    if g < g^best then
        g^best ← g
    return (Φ, g)
/* Otherwise, if the current partial solution is worse than the best solution
so far, then backtrack. */
if g ≥ g^best then
    return (Φ, g)
/* Try each unassigned task τ_i at the current priority level. */
(Φ^min, g^min) ← ((), +∞)
for τ_i ∈ Γ do
    δ ← DMR_i(Φ)
    /* Get the best solution completing Φ.τ_i. */
    (Φ', g') ←RECUR(Γ\{τ_i}, Φ.τ_i, l − 1, g + δ)
    /* Memorize the best completed solution. */
    if g' < g^min then
        (Φ^min, g^min) ← (Φ', g')
    /* If task τ_i has a null DMR, then backtrack. */
    if δ = 0 then
        break
/* Return the best completed solution. */
return (Φ^min, g^min)
```

8 Conclusions

In this paper, we recognised that a key problem that needs to be addressed by probabilistic analysis is the calculation of the timing failure rate, the rate at which we can expect deadlines to be missed during the operation of a system. This failure rate can then be compared with

reliability requirements to determine if the system is acceptable.

As a first step towards a solution to this problem, we proposed three sub-problems. These sub-problems relate to finding an optimal priority assignment for fixed priority pre-emptively scheduled systems with probabilistic execution times. The problems involve optimising three different metrics based on the probability of tasks missing their deadlines over some interval of time.

The three sub-problems were as follow:

1. Basic Priority Assignment Problem (**BPAP**). This problem involves finding a priority assignment such that the deadline miss ratio of every task does not exceed the threshold specified.

2. Minimization of the Maximum Priority Assignment Problem (**MPAP**). This problem involves finding a priority assignment that minimizes the maximum deadline miss ratio of any task.

3. Average Priority Assignment Problem (**APAP**). This problem involves finding a priority assignment that minimizes the sum of the deadline miss ratios for all tasks.

For each sub-problem we proposed an optimal algorithm. The first two algorithms were inspired by Audsley's algorithm which is a greedy, lowest priority first, approach that is optimal for the equivalent deterministic case. As shown in Section 7 such a greedy approach is not suitable for the third sub-problem (APAP). Therefore we proposed a tree search algorithm in this case.

As future work we intend to provide conditions that need to be met in order to obtain an optimal priority ordering for APAP using a greedy algorithm. Such general conditions exist for the deterministic case [7] and our first step will be to consider their extension to the probabilistic case.

9 Acknowledgements

This work was partially funded by the UK EPSRC funded Tempo project (EP/G055548/1) and the EU funded ArtistDesign Network of Excellence.

References

[1] L. Abeni and Buttazzo. QoS guarantee using probabilistic deadlines. In *IEEE Euromicro Conference on Real-Time Systems (ECRTS99)*, pages 242–249, 1999.

[2] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, University of York, 1991.

[3] N. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[4] L. Cucu and E. Tovar. A framework for response time analysis of fixed-priority tasks with stochastic inter-arrival times. *ACM SIGBED Review*, 3(1), 2006.

[5] R. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *the 28th IEEE Real-Time Systems Symposium (RTSS07)*, pages 3–14, 2007.

[6] R. Davis and A. Burns. Robust priority assignment for messages on controller area network (CAN). *Real-Time Systems*, 41(2):152–180, 2009.

[7] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems Journal*, 47(1):1–40, 2011.

[8] J. Díaz, D. Garcia, K. Kim, C. Lee, L. Bello, J. López, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *the 23rd IEEE Real-Time Systems Symposium (RTSS02)*, pages 289–300, 2002.

[9] M. Gardner and J. Lui. Analyzing stochastic fixed-priority real-time systems. In *5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 1999.

[10] G. Kaczynski, L. Lo Bello, and T. Nolte. Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems. 2007.

[11] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *the 11th IEEE Real-Time Systems Symposium (RTSS'90)*, pages 201–209, 1990.

[12] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case bahavior. 1989.

[13] J.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[14] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[15] J. Lopez, J. L. Díaz, J. E., and D. García. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-time Systems*, 40(2):180–207, 2008.

[16] D. Maxim and L. Cucu-Grosjean. Towards optimal priority assignment for probabilistic real-time systems with variable execution times. In *the 3rd Junior Researcher Workshop on Real-Time Computing (JRWRTC 2009)*, 2009.

[17] D. Maxim and L. Cucu-Grosjean. Towards optimal priority assignment for probabilistic CAN-based systems. In *WIP session of the 8th IEEE International Workshop on Factory Communication Systems (WFCS'2010)*, 2010.

[18] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: prentice Hall, 1995.

[19] T. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. Wu, and J. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *the 1st IEEE Real-Time and Embedded Technology and Applications Symposium*, 1995.

[20] D. Wolpert and W. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[21] H. Zeng, M. D. Natale, P. Giusto, and A. L. Sangiovanni-Vincentelli. Stochastic analysis of distributed real-time automotive systems. *IEEE Trans. Industrial Informatics*, 5(4):388–401, 2009.