Delft University of Technology
Master of Science Thesis in Embedded Systems

Deriving Timing Properties from System Traces using Data-driven Techniques

Şerban Vădineanu
Supervised by Dr. Mitra Nasri

Embedded Networked Systems

TU Delft Delft University of Technology

Deriving Timing Properties from System Traces using Data-driven Techniques

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Şerban Vădineanu

TU Delft Student Number: 4824989
Supervised by Dr. Mitra Nasri

23.07.2020

Author
Şerban Vădineanu ()(

(**TU Delft Student Number:** 4824989)

Title
Deriving Timing Properties from System Traces using Data-driven Techniques

MSc Presentation Date
03.08.2020

Graduation Committee

dr. ir. Fernando Kuipers (chairman)	Delft University of Technology
dr. Mitra Nasri	Delft University of Technology and Eindhoven University of Technology
dr. Jan Rellermeyer	Delft University of Technology

Abstract

With the growth in the complexity of real-time embedded systems, there is an increasing need for tools and techniques to understand and compare the observed runtime behavior of a system with the expected one. Since many real-time applications require periodic interactions with the environment, one of the fundamental problems in guaranteeing/monitoring their temporal correctness is to be able to infer the periodicity of certain events in the system. The practicability of a period inference tool, however, depends on both its accuracy and robustness (resilience) against noise in the output trace of the system, e.g., when the system trace is impacted by events that have a non-deterministic nature such as the presence of aperiodic tasks, release jitters, and runtime execution-time variations of the tasks.

This work **(i)** presents a period inference framework that uses regression-based machine-learning (RBML) methods, **(ii)** thoroughly investigates the accuracy and robustness of different families of RBML methods in the presence of uncertainties in the system parameters, and **(iii)** proposes further accuracy improvements by deriving candidate pruning rules based on the inherent properties of the underlying scheduling policies. We show, on both synthetically generated traces and traces from actual systems, that our solutions can reduce the error of period estimation by two to three orders of magnitudes w.r.t. state of the art. Also, our methods showed to be robust against most sources of disturbance.

Preface

This work concludes my 2-year journey as an Embedded Systems master student at TU Delft. Throughout the time I spent here, and especially during my thesis, I have encountered numerous difficulties but an even greater amount of opportunities.

I would like to thank my supervisor, Dr. Mitra Nasri, for her invaluable guidance and for teaching me how to view every challenge as an opportunity, how to critically think my way out of problems, and, overall, how to become a better researcher.

My sincere gratitude goes towards my parents, who never questioned my decisions and gave me their full support even when I was not entirely confident about my choices. They invested me with their trust which was the most motivating aspect for me to finish everything and make them proud.

I would also like to thank Lorena, who provided me with 24/7 ”psychological consultancy” and whose patience and willingness to help allowed for rays of sunshine in my most cloudy days.

Also, I would like to say an honest “mulțumesc” to all my friends from Romania who not only were there to listen to my complaints about how hard is to do a thesis abroad, but also provided me with computing resources when the experiments where many and the time was tight.

Lastly, I don't want to forget the bonds I built in Delft, which although are few I truly believe that are strong. Thank you, guys, for sharing some of your brief moments of free time with me and for allowing me to know you better!

Şerban Vădineanu

Delft, The Netherlands
23rd July 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid growth of software size and complexity in real-time embedded systems has posed imminent challenges to the ability to debug systems, identify runtime deviations from the correct service [52], and detect (and evade) security attacks at runtime [33]. This raises an urge for tools and techniques to understand (or infer) the runtime behavior of a system from its observable outputs such as the traces of output messages, task executions, actuations, etc. without impacting the system itself or, in some cases, without being able to access the source code or the internal parts of the system.

1.1 Problem Statement

In this work, we focus on developing a tool for inferring the timing properties of a system. Such a tool can be used to **(i)** find time-bugs during the development phase (e.g., to check if activities happen with the expected frequency or period), **(ii)** detect timing anomalies and security attacks that leave a trace on the observable timing profile of the system during the operation phase (e.g., such as those explained in [33, 42] to spot anomalies in the regularity of an activity in the system [20]), and **(iii)** diagnosing the system after applying a patch or an upgrade during the maintenance phase (e.g., to check if a data-consumer application still performs periodically after installing an upgrade on the data-producer application).

Since many real-time applications require periodic interactions with the environment, one of the primary use cases of a timing inference tool is to infer the periodicity of events from a system's output traces. What makes this very first step challenging is that the observable timing traces are typically obtained from the components' interfaces and hence are impacted by the internal structure of the application, operating system, hardware platform, and their interactions. For instance, consider an execution trace that indicates the time intervals during which a certain task has occupied the processor. It is easy to infer the period if the task exclusively runs on top of dedicated hardware. It becomes harder if the task is one of the low-priority tasks in a set of periodic tasks running on top of a real-time operating system (RTOS) with a preemptive fixed-priority scheduling (FP) policy because then the task's execution intervals are affected by the interference from the higher-priority periodic tasks. Finally, it becomes much harder if the latter system also includes high-priority aperiodic or event-driven activities (such as interrupt services), sporadic tasks, release jitters, and deadline misses. A timing inference tool, therefore, must be *robust* against these interferences, dynamic behavior, and uncertainties; otherwise, it might not be able to address true challenges faced by real systems and hence becomes useless in practice. Furthermore, it must be accurate, else it will not be helpful to find time bugs or to detect deviations from the expected periodicity.

We consider the problem of inferring a task's period from a timed-sequence of zeros and ones (called a *binary projection*) that shows when the task was occupying the resource during the interval of time for which the trace has been obtained. We consider a single processing resource, e.g., a CPU core, a network link, a CAN bus, etc. We assume *no prior knowledge* about the number of other tasks in the system and their parameters, execution model, runtime, execution-time variations, and release jitters.

1.2 Research Questions

The aim of this work is to create an accurate and robust period inference tool. We try to achieve this goal by answering the following research questions:

- **RQ1.** *Can one infer the tasks' periods from traces of a real-time system using RBML (regression-based machine learning) techniques? If so, how effective (in terms of accuracy) and efficient (in terms of runtime) would those methods be?*

  This question's aim is to explore how a data-driven method for estimating the period of a task can be compared to the already existing approaches. We propose the first framework to use RBML methods for period inference problem and then compare its accuracy, runtime, and memory consumption with the state of the art.

- **RQ2.** *What impact does the choice of the RBML algorithm have on the effectiveness of period inference?*

  Various families of RBML methods have been introduced and applied on different problems by the literature. We investigate the performance of different families of RBML methods in terms of their accuracy and robustness for the problem of period inference in real-time systems.

- **RQ3.** *Can we derive a set of pruning rules to further restrict the number of possible period values? If so, what will be the impact of adding the set of rules over the RBML performance in terms of accuracy?*

  By answering this question, we want to find out whether incorporating knowledge derived from the real-time systems theory can help in increasing the accuracy of the RBML methods.

- **RQ4.** *How robust is our solution against the interferences caused by the non-determinism present during the operation of real systems?*

  The purpose of this question is to assess the impact of different sources of uncertainty in the underlying system's behavior on the proposed solutions for the period-inference problem.

1.3 Contributions

In order to address the first research question **(RQ1)**, we propose the **first period inference framework that utilizes RBML methods**. We focus on regression-based methods due to the continuous nature of the target variable: the period. Our method relies on training a model from labeled traces. However, apart from labels, our RBML solution requires a set of features to be trained on. As a consequence, we interpret the execution of a task as a binary signal so that we can apply signal-processing techniques (i.e., periodogram and circular-autocorrelation) in order to generate period candidates, which will be further used as the required features. However, since the number of the obtained candidates is large, we include in our process an additional step of selection and combination such that we only choose as features the right amount of candidates in order to reach a trade-off between accuracy and input size. Once the model is trained, we further enhance its accuracy by creating a **period adjustment step**, based on selecting the closest candidate to our model's prediction.

For answering the second and fourth research questions (**RQ2** and **RQ4**), we perform a **thorough investigation** of widely-used families of regression algorithms, which showed promising performance in recent literature studies such as the review of Delgado et al. [12]. We include not only comparisons in terms of accuracy, but also in terms of runtime and memory consumption, and robustness of these methods.

We address the third research question **(RQ3)** by **deriving two theoretical bounds** based on the properties of real-time scheduling policies to narrow down the search space of the period estimate.

1.4 Organization

Chapter 2 introduces the essential notions which we will use throughout the thesis, it also includes the system model with a formalization of our problem. The state of the art is presented in Chapter 3. In Chapter 4 we offer a description of our period mining solution. Chapter 5 is dedicated to the empirical evaluation of our methods, while Chapter 6 provides insight on the strengths and weaknesses of different families of RBML methods for the problem of period inference. Finally, Chapter 7 provides the conclusion and possible research paths for future work.

Chapter 2

Background and System Model

In this chapter, we introduce the fundamental notions over which we define our problem and formulate the solution. Section 2.1 provides an overview of the background information, while in Section 2.2 we present the model of our system and we state the problems we aim to solve.

2.1 Background

A real-time system is a system whose correctness depends not only on its logical correctness, but also on its temporal correctness.

The most relevant real-time systems notions that we will use throughout this work are the following:

- **Task.** A task implements a functionality of the system. It can represent a process or a thread that are executed on the processor, and it can create an unlimited number of instances, called **jobs**. A task is characterized by the following properties:
  - **Release time.** The release time of a job is the time at which the job is ready for execution.
  - **Execution time.** The execution time is the time a job requires to access the resource (e.g., processor) to complete. Different jobs of the same task can have different execution time values. The lower and the upper bounds on the execution time are called the *best-case execution time* (BCET) and the *worst-case execution time* (WCET), respectively.
  - **Deadline.** The *deadline* is the time before which a job must complete its execution in order to guarantee the system's correctness. If this time is specified w.r.t. the release time of the task, then it is denoted as *relative deadline.* If the deadline is specified w.r.t. to a time origin, e.g., time 0, then it is referred as *absolute deadline* [6]. Depending on the consequence of a deadline miss for the system, a deadline can be categorized as *soft* or *hard.* Missing a soft deadline

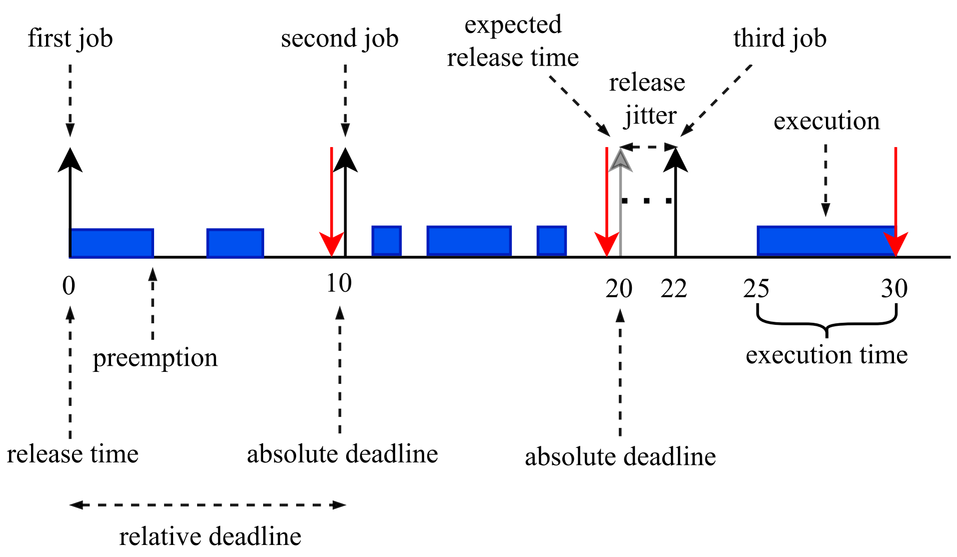

Figure 2.1: **Example of a task with a period of 10, relative deadline equal to the period, and release time jitter.**

will not have a catastrophic consequence for the system while missing a hard deadline will result in the system failure.

The aforementioned properties are exemplified in Figure 2.1.

- **Periodicity.** Depending on the release pattern of its jobs, a task can be:
  a. **Periodic.** The jobs are released with a certain period.
  b. **Aperiodic.** Tasks have non-deterministic release time, which can be triggered, for instance, by events.
  c. **Sporadic.** The inter-arrival time between two consecutive jobs is separated by a minimum inter-arrival time (also called period).
- **Release time jitter.** The maximum deviation of the release time among all invocations of a task is called *(maximum) release time jitter*. The release time jitter can be caused by timer inaccuracy, interrupt latency, or networking delays [34]. An illustration of release time jitter is present in Figure 2.1.

• **Schedule.** The schedule is a particular assignment of tasks to the processor(s) and time intervals. This assignment determines the task execution sequence and is typically generated by a scheduling algorithm (*policy*).

• **Scheduling policy.** It is the policy by which jobs are scheduled. There are three main categories of scheduling policies: task-level fixed-priority scheduling (FP), job-level fixed-priority scheduling (JLFP), and dynamic-priority scheduling (DP).

- **Fixed-priority scheduling.** For this category of algorithms, all jobs of a task will have the same priority which is typically assigned to the task at design time. After being assigned, the priority of the task will not change during the lifetime of the system. A common priority assignment method that is used by fixed-priority scheduling is *rate monotonic* (RM) for periodic tasks. RM assigns a higher priority to a task with a smaller period. Considering that the period of a task is a constant property, the priorities of the tasks will not change throughout the system's execution, making RM a fixed-priority assignment.

  – **Job-level fixed-priority scheduling.** This type of policies allow jobs to have different priorities. JLFP policies include widely implemented scheduling policies in real-time systems such as the *earlier-deadline first* (EDF), FP, and *first-in-first-out* (FIFO) policies. EDF is a policy that assigns a higher priority to a job with an earlier absolute deadline. EDF is not a task-level fixed-priority policy since at different moments in the schedule the same task can have different priorities.

  – **Dynamic-priority scheduling.** In these policies, a job's priority can change over the time. The *least-laxity first* scheduling policy is one of the dynamic-priority policies as it changes the priority of a job over the time as the job progresses towards its completion.

- **Execution model.** The most general categorization of execution models is as: *preemptive* and *non-preemptive.*

  During scheduling, the tasks that are waiting for a time slice on the processor reside into the *ready queue*, while the task in execution is called *running task* [6].

  – **Preemptive execution.** The operation of suspending the running task and inserting it into the ready queue is called *preemption* [6]. Hence, when a higher-priority task enters the ready queue, it will preempt the lower priority running task.

  – **Non-preemptive execution.** The non-preemptive execution model disables the use of preemption completely. Thus, a running task cannot be suspended by other higher-priority tasks.

2.2 System Model and Problem Definition

We assume a system with a single (processing) resource (such as a CPU core, I/O or CAN bus, or a link on the network). The resource can be occupied/used by a set of tasks $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$, scheduled by a *job-level fixed priority* (JLFP) scheduling policy on the resource. Furthermore, we assume no restriction on whether each task executes preemptively or non-preemptively.

A task in τ can be activated periodically, sporadically, or aperiodically. A periodic or sporadic task is identified by $\tau_i = (C_i^{min}, C_i^{max}, T_i, D_i, \sigma_i)$, where C_i^{min} and C_i^{max} are the *best-case* and *worst-case execution times* (BCET and WCET), T_i is the period, D_i is the relative deadline (which is assumed to be equal to the period), and σ_i is the maximum release jitter of the task. If the task is sporadic, its period indicates the minimum-inter arrival time between its activations. An aperiodic task is identified by a 3-tuple $\tau_j = (C_j^{min}, C_j^{max}, D_j)$, where C_i^{min} and C_i^{max} are the BCET and WCET and D_j is the relative deadline of the task, respectively. We further assume that all timing parameters are positive integer values in $\mathbb{N}^+$ with the exception of C_i^{min} and σ_i that can be 0. The total utilization of the system is denoted by U and it is the sum of the utilization of all periodic and sporadic tasks in the system, where a task utilization is the ratio of its WCET to its period. The *hyperperiod* of a task set, denoted by H, is the least common multiple of the periods.

We use $J_{i,k}$ to denote the k-th job of a task τ_i. Each job $J_{i,k}$ has a priority $p_{i,k}$ that is assigned by the scheduling policy at its release time. The priority remains unchanged as long as the job is in the system and priority ties are broken arbitrarily. We assume that at any time instant t, either one of the tasks in τ or the *idle task*, denoted by τ_0, is running on the resource.

A trace $\mathcal{T} = ([t_s, t_e], \langle \epsilon_1, \epsilon_2, \ldots, \epsilon_N \rangle)$ represents a time-ordered sequence of symbols that represents a schedule generated by the JLFP scheduler for the task set $\tau \cup \{\tau_0\}$ from the time t_s to t_e. Each symbol $\epsilon_i \in \mathcal{T}$ is an identifier (index) of a task that was occupying the resource at time i, where $i \in \{t_s, t_{s+1}, \ldots, t_e\}$. Hence, $\epsilon_i \in \tau \cup \{\tau_0\}$. The length of a trace is $|\mathcal{T}| = t_e - t_s$. Figure 2.2(a) and (b) show a schedule of a task set with 4 tasks and the equivalent trace of that schedule.

Before we formally define the problems that are being solved in the thesis, we need to introduce two other notions that are tied to a trace: *binary projection* and *ternary projection*. A binary projection for task τ_i is a sequence of zeros and ones that represents the times at which a job of the task was occupying the resource in the trace. Figure 2.2(c) shows the binary projection of the task τ_3.

Definition 1 *A* binary projection *of a trace $\mathcal{T}$ for task τ_i, denoted by $P_i^B = \langle p_1, p_2, \ldots, p_{|\mathcal{T}|} \rangle$, is a time-ordered sequence of elements p_k, where*

$$p_k = \begin{cases} 1, & \epsilon_k = i \\ 0, & otherwise \end{cases} . \tag{2.1}$$

For convention, we assume that $p_k = 1$ means that task τ_i was running during the interval $[k, k+1)$. For instance, in Figure 2.2(a), $p_6 = 1$ denotes that τ_3 was running during the interval $[6, 7)$.

A ternary projection for a task τ_i is similar to the binary projection except that it also contains the resource-idle intervals. Figure 2.2(d) shows the ternary projection of the task τ_3.

Definition 2 *A* ternary projection *of a trace $\mathcal{T}$ for task τ_i, denoted by $P_i^T = \langle p_1, p_2, \ldots, p_{|\mathcal{T}|} \rangle$, is a time-ordered sequence of elements p_k, where*

$$p_k = \begin{cases} 1, & \epsilon_k = i \\ idle, & \epsilon_k = 0 \\ 0, & otherwise \end{cases} . \tag{2.2}$$

Next, we define two versions of the *period inference* (PI) problem as follows:

Problem 1 *Find the period of τ_i from its projection P_i^B.*

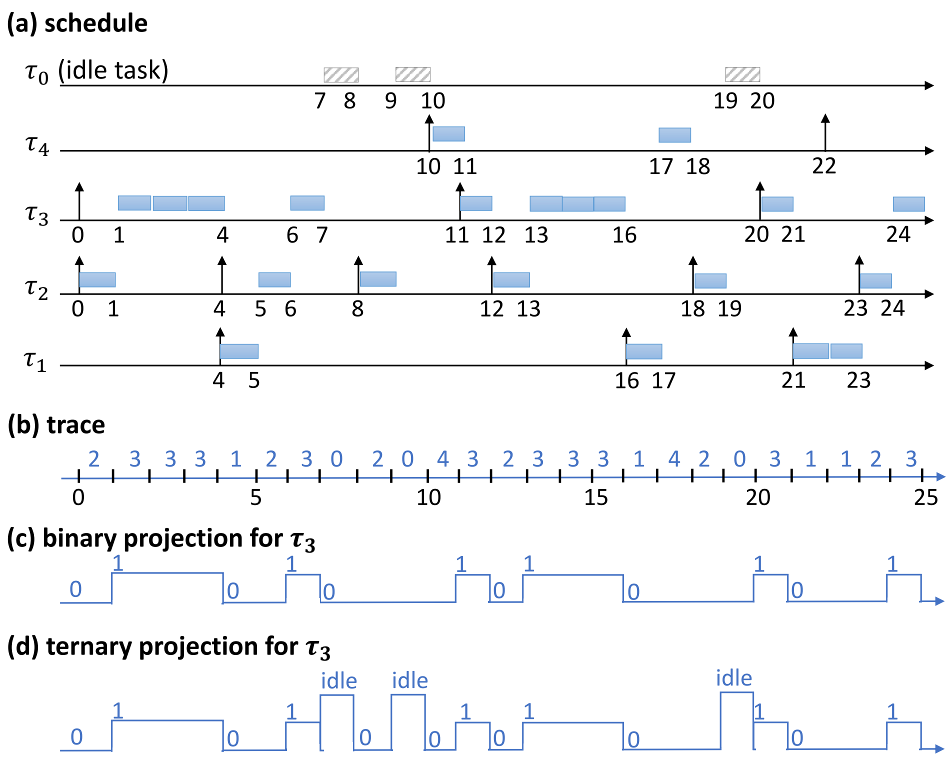

Figure 2.2: **An example of a task set with one aperiodic task (τ_1), two sporadic tasks (τ_2 and τ_4) and one periodic task with release jitter (τ_3) scheduled by a FP policy (assuming $p_i = i$). (a) shows a schedule, (b) shows the trace of the schedule, and (c) show the binaryprojection of the trace for task τ_3.**

Problem 2 *Find the period of τ_i from its projection P_i^T provided that P_i^T does not include a deadline miss from τ_i.*

It is worth noting that the only input to the Problems 1 and 2 is a projection. Since a projection is just a sequence of limited symbols ('0', '1', and 'idle'), it does not contain any information about the scheduling policy, tasks' parameters (such as the execution times, release jitter, periods, deadlines, etc.). Moreover, the projections themselves do not contain information about whether or not there are tardy jobs (i.e., jobs that have completed after their deadline) in the original trace.

Chapter 3

Related Work

In this chapter we review previous works that share similarities with our project. We begin our review with a section dedicated to the works on period estimation (Section 3.1), we then explore the state of the art on deriving other timing properties from real-time systems apart from the periodicity of the tasks (Section 3.2). Also, we explore how other authors employed machine-learning techniques to extract meaningful attributes from real-time systems traces (Section 3.3). Finally, we provide an overview for the works that applied regression-based machine learning (RBML) on features extracted from signals (Section 3.4).

A summary of the most relevant works on inferring properties of real-time systems can be found in Table 3.1.

3.1 Period Estimation

Although periodicity mining has been extensively studied in the past couple of decades, very few works have actually focused on inferring periods from the execution traces of real-time tasks. Patel and Modi [35] have surveyed various types of periodicity mining in the problem of finding similar patterns and frequent items in a set, however, since their goal is not to find a single numerical value as the period of an event, they are not really relevant to this thesis. In the rest of this section, we will only focus on the works that are aiming to find a numeric period in the data as part of their solutions.

Iegorov et al. [21] are among the few pioneers who proposed a solution for the problem of inferring periods from execution traces. They created an algorithm which identifies the time intervals between consecutive task activations and computes the period as the mode of the intervals' distribution. However, their method performs poorly when the tasks have runtime execution-time variation and/or the true period of the task under analysis does not divide all other smaller periods in the task set, i.e., it is not *harmonic* with the rest of the tasks. Young et al. [52] use a *fast Fourier transformation* to infer the periodicity of messages sent on a *controller area network* (CAN) in order to detect security attacks that impact the timing of the messages. Their problem, however, is only a subset of ours since CAN applies a non-preemptive fixed-priority (FP) policy and messages have typically a fixed size with a low runtime variation on the message length.

Paper	**Problem**	**Approach**
Iegorov [21]	Period inference	Determine the distribution of time intervals between the jobs of a task
Young [52]	Intrusion detection	Infer periods of messages using fast Fourier transform
Iegorov [20]	Precedence constraints	Mine a task precedence graph from execution traces
Cutulenco [11]	Timing constraints	Mine timed regular expressions from execution traces
Lamichhane [26]	Program tracing	Identify the running task from power consumption traces
Sucholutsky [47]	Trace restoration	Use recurrent neural networks to reconstruct execution traces

Table 3.1: **Works on reverse-engineering real-time systems.**

Paper	**Periodogram**	**Autocorrelation**
Vlachos [49]	x	x
Berberidis [3]		x
McKilliam [31]	x	
Puech [37]	x	x
Unnikrishnan [48]	x	
Malode [30]		x
Li [29]	x	
Liang [8]	x	

Table 3.2: **Works on period estimation from signals.**

Finding the periodicity of a signal is a well-studied problem in signal-processing research [43, 3, 49, 29, 31, 30, 48, 37, 18]. As it can be observed from Table 3.2, *Periodogram* [43] and *circular autocorrelation* [18] are among the widely used methods to find a plausible set of periods for a signal. However, as we will see in the experimental section, these methods perform poorly when used on signals generated from preempted tasks.

3.2 Timing Properties from Real-time Systems

Apart from the works of Iegorov et al. [21] and Yang et al. [52] who focused on deriving the periods of the tasks, other works aimed to infer a different type of timing properties. Cutulenco et al. [11] proposed a technique for extracting timed regular expressions from execution traces, which provide information about the occurrence pattern of the events in the system. Similarly, Iegorov et al. [20] mine an occurrence pattern between the tasks in the system under the form of a task precedence graph. However, although both works aim to derive a timing property which relies on periodicity, namely the precedence constraints, they do not infer the actual values for the periods. They only determine interpretations of the occurence patterns under the form of regular expressions and precedence graphs, respectively.

3.3 Machine Learning Methods for Reverse-engineering Real-time Systems

Data-driven methods such as k-nearest neighbors and dynamic time-warping algorithms have been used in reverse engineering real-time systems to identify tasks from their runtime power traces [26]. In addition, *long short-term memory* (LSTM) neural networks have been used to reconstruct traces affected by noise [47]. However, to the best of our knowledge, no study so far has utilized *regression-based machine learning* (RBML) methods to infer the timing properties of real-time systems.

3.4 RBML on Signals

Our proposed approach for finding the periods of the tasks relies on data-driven techniques. Since we aim to estimate a continuous value, the family of algorithms that we considered is regression. Thus, we investigated solutions that employed regression algorithms to determine trends from signals.

Although there are no records of regression models being used to directly estimate the period of a signal, these algorithms have been successfully employed together with signal features to solve various tasks. Benkedjouh et al. [2] used Mel-frequency cepstral coefficients extracted from raw sensor signals to train a support vector machine. The end goal was to estimate the remaining useful life of tools used in the manufacturing process. Similarly, Wang et al. [50] created a tool wear prediction model using Gaussian mixture regression on features extracted from cutting force signals. In another work, Xia et al. [51] use random forest to detect acoustic events, while Slapničar et al. [46] also make use of a related regression algorithm to estimate the blood pressure from photoplethysmogram signals. When it comes to electrical signals, Fei et al. [14] train a support vector on voltage waveforms to find the faulty locations in the transmission line.

The wide applicability of regression demonstrated that this family of algorithms is suitable to be used in tandem with signal-processing techniques. However, to the best of our knowledge, no work has used regression models as a substitute for signal-processing techniques or to directly improve their output.

Chapter 4

RBML-based Period Inference Framework

4.1 Solution Overview

This section first introduces the challenges of the period inference (PI) problem and then presents our solution.

Challenges. As mentioned earlier, the PI problem has a long history in signal processing. Methods such as periodogram [43] and circular autocorrelation (or autocorrelation for short) [18] have been applied to infer periodicity of a signal and shown to work well in the presence of small (or standard) noise. However, when applied to the PI problem, the results revealed a couple of issues: **(i)** these methods can generate many candidate periods most of which are irrelevant, **(ii)** although they assign a weight (called power) to each period candidate, there is no direct relation between the weight and the true period, **(iii)** they cannot handle scenarios with preemptions well because they perceive each preemption as a new occurrence of the event under analysis (which adds a significant amount of noise to their inputs), and **(iv)** the true period is not necessarily among the candidates, particularly for the autocorrelation method. Hence, they could not provide an accurate solution for the PI problem.

We then decided to look into the machine-learning methods that could work well on the PI problem. We specially focused on those whose decision logic is *explainable* and traceable by a human. Therefore, we deliberately avoided using deep neural networks for the problem or for the feature extraction. However, this raised the next challenge: how to extract meaningful and helpful features from a projection? A starting point could be to use the whole binary projection as a feature and let the machine-learning method figure out the period. However, that could lead to two major issues: **(i)** dimensionality problems with the feature space, and **(ii)** having inputs with varying-length. High-dimensional feature spaces typically lead to sparse data which in turn reduces the efficiency and increases the runtime of model learning [1]. Moreover, most machine-learning methods require a fixed input size which implies that the input projection must be cut (for all projections of all training and testing task sets). However, since task sets have different hyperperiods, putting a predetermined cut-off threshold could either lead to low accuracy (if the cut-off is too short) or to a huge runtime

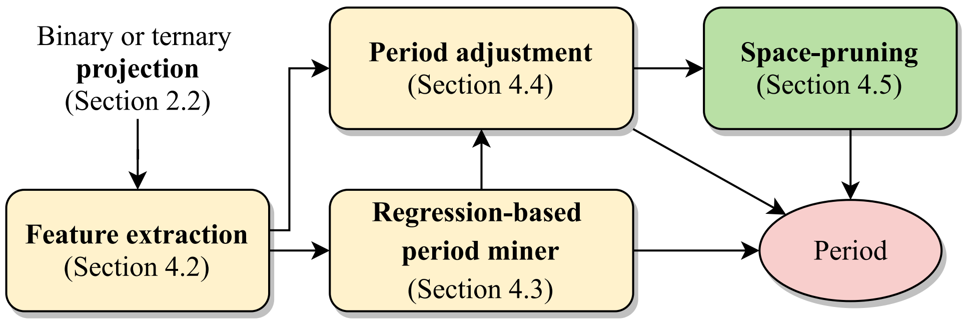
Binary or ternary projection (Section 2.2)
Period adjustment (Section 4.4)
Space-pruning (Section 4.5)
Feature extraction (Section 4.2)
Regression-based period miner (Section 4.3)
Period

Figure 4.1: **Our period inference framework. The edges denote the information flow between our algorithms.**

and low efficiency in learning the model (if the cut-off is too long).

Solution highlights. The framework we propose to solve the period inference problem suggests a four-stage pipeline where Stage 0 extracts features and Stages 1 to 3 are for accuracy improvement of period estimation. Figure 4.1 shows the pipeline and the stages. In Stage 0, we use both periodogram and autocorrelation to compile a fixed set of features from their top k highest-rank candidates (see Section 4.2). In Stage 1, we use supervised learning methods, and in particular, the regression-based machine-learning methods, to determine the relationship between our feature vectors and the target output, i.e., task period (see Section 4.3). RBML methods are commonly used when the goal is to predict a continuous output that takes order into consideration (in our case, the period). We call our RBML solution *regression-based period miner* (RPM). In Stage 2, we further adjust the predictions of RPM by providing it with a set of high-ranked candidates from periodogram and autocorrelation. This aims to use RPM as a referee whose purpose is to highlight the most accurate peak from the two signal processing methods (see Section 4.4) Stage 3, on the other hand, uses the extra information provided in ternary projections to extract pruning rules to further restrict the number of candidates (Section 4.5).

4.2 Feature Extraction

This section briefly explains how periodogram and autocorrelation methods work.

Periodogram [43]. Consider a sequence $x(n)$ and its discrete Fourier transform $X(f)$. The periodogram $\mathcal{P}$ gives an estimation of the spectral density of the discrete signal $x(n)$ and is obtained from the squared magnitude of the Fourier coefficients $X(f)$, as presented in [28]:

$$\mathcal{P}(k) = \frac{1}{N}\|X(k)\|^2, \tag{4.1}$$

where N is the sequence length and $\mathcal{P}(k)$ is the power of frequency k.

Figure 4.2(a) and (b) show two periodograms obtained for two periodic tasks with period 1000 and 5000 from a task set with four tasks scheduled by rate monotonic scheduling policy. As can be seen in Figure 4.2(a), the highest peak in this example (here, peak refers to a jump in the diagram) of the periodogram

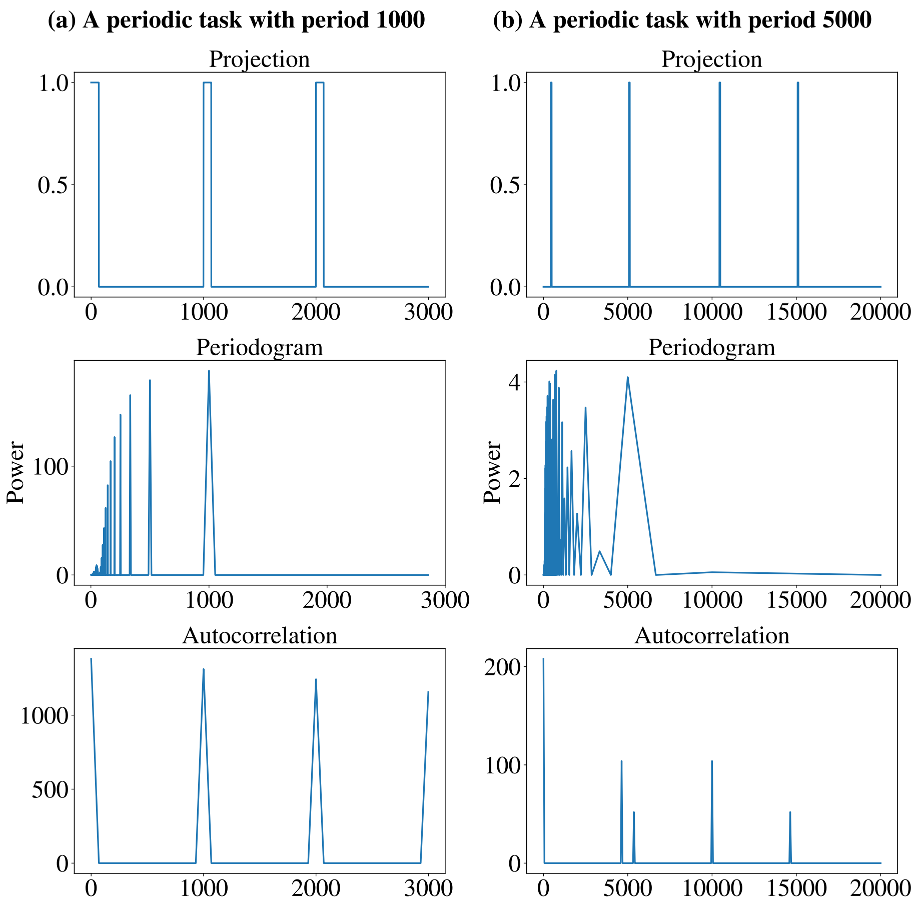

Figure 4.2: **Periodogram and circular autocorrelation methods applied on task projections for a task with (a) period 1000, and (b) period 5000 from a system containing 4 tasks, with a total utilization of 30% and scheduled by the rate monotonic scheduling policy. The other two tasks have a period of 2000 and 10000, respectively.**

indicates the true period of the task, i.e., 1000. However, for the task with period 5000, this observation does not hold; the true period of this task is not the highest-peak but the 5$^{\text{th}}$ highest peak. The lower the priority of a task, the higher is the amount of interference it will have in its schedule. These interferences make the projections less regular and hence result in a more irregular periodogram that has many peaks.

Circular Autocorrelation [18]. It is a metric that describes how similar is a sequence to its past values for different circular phase shifts. We use Vlachos et al. [49] method to compute the circular autocorrelation:

$$\mathcal{ACF}(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(k)x(n+k), \tag{4.2}$$

where N is the sequence length and k is the phase shift. In the case of period inference problem, we would expect that the highest value of the autocorrelation function would be at a lag k equal to the true period.

A practical way to compute the $\mathcal{ACF}$ is to translate the operations into the frequency domain. Since (4.2) is a convolution, one can compute it with the dot product between the Fourier coefficients of the sequence and their complex conjugates [49]:

$$\mathcal{ACF} = \mathcal{DFT}^{-1} X \cdot X^*, \tag{4.3}$$

For our problem, the discrete Fourier transform is applied to the projection in order to extract the Fourier coefficients. Furthermore, we perform the dot product between the coefficients and their complex conjugates and apply the inverse Fourier transform on the result to obtain the autocorrelation.

Figure 4.2 illustrates the usage of autocorrelation when the input is a projected trace. Firstly, we notice that the highest value that this technique exhibits is for a lag (period) of 0. This behavior is normal, since the highest similarity between a signal and itself is present when the two signals perfectly overlap with each other, e.g., at time 0. Hence, the peak at 0 is excluded from the examination. The other observation is that, similar to the periodogram, the autocorrelation method is able to discover the true period only in the case from Figure 4.2(a), while for the second period, its top peak indicates an erroneous value. Moreover, we observe that the autocorrelation is sensitive to low utilization values. In Figure 4.2(b), although the task has not been preempted, the slight variation at the start of its execution causes the projection to not have any overlap with itself when is shifted by the true period of 5000. As a result, the autocorrelation method could not detect the actual periodic behavior. However, it could observe two smaller peaks slightly shorter and slightly larger than 5000 at 4635 and 5365, respectively.

It is worth noting that, both periodogram and autocorrelation methods have an $O(p \log p)$ time complexity, where p is the length of the projection[36].

Extracting fixed-size features. Our fixed-size candidate list is constructed from the top $k = 3$ peaks of the outputs of the two methods, namely, we gather k-highest peaks from periodogram and k-highest peaks from the autocorrelation methods. This allows us to work on a much smaller dimension for the input-data and have fixed input size to use with our regression-based solution. For the cases when there are fewer than k peaks for a method, the number of features is completed by appending the highest peak of that method until we reach the desired k. The choice on the number of features, i.e., $k = 3$, was made after evaluating the impact of k on various scenarios and finding out the suitable value that results in a high accuracy without increasing the dimensions of the feature space (Figure 5.1(b) in Chapter 5 compares different choices.).

4.3 Regression Methods

Regression analysis is a method originating from statistics, whose purpose is to estimate the relationship between a dependent variable (also referred to as "outcome") and one or more independent variables (also called "features"). In machine learning, regression is employed when the aim is to predict a continuous output, which takes order into consideration. A regression model can be formally described by

Algorithm	Nickname	Category
Cubist Regression [38, 39, 40]	*cubist*	Rule-based
Generalized Boosting Regression [15]	*gbm*	Boosting
Averaged Neural Network [41]	*avNNet*	Neural Networks
Extremely Randomized Regression Trees [16]	*extraTrees*	Random Forests
Bayesian Additive Regression Tree [9]	*bartMachine*	Bayesian Models
Support Vector Regression [10]	*svr*	Support Vector Machines

Table 4.1: **Overview of best performing families of regression algorithms and for each family the best model [12].**

$$Y_i = f(X_i, \beta) + e_i, \tag{4.4}$$

where Y_i is the outcome variable, belonging to the vector Y, X_i is a vector comprised of the independent variables, β represent the unknown parameters and e_i is an additive error term (residual) which is associated with the prediction.

Since we try to estimate the period from a projection, in our regression scenario, the dependent variable Y_i is the task's period T_i. The independent variables X_i contain the features we extracted at the previous step, while the function f comes from the choice of a regression algorithm, whose parameters β we need to estimate through training. Thus, our goal is to choose the form of function f and to compute the estimates of the parameters $\hat{\beta}$ such that the function has the best fit on the data. In order to assess how well the model fits the data, the predicted outcome, i.e.,

$$\hat{Y_i} = f(X_i, \hat{\beta}), \tag{4.5}$$

is compared against the true dependent variable. The comparison is present in the shape of a loss function $L(Y, f(X, \hat{\beta}))$, where Y is a vector containing the outcome variables and X includes all vectors of independent variables. For instance, the most commonly used loss function is the mean square error (MSE):

$$MSE = \frac{\sum_{j=1}^{N}(Y_j - \hat{Y_j})^2}{N}, \tag{4.6}$$

where N is the total number of observations.

The choice of regression methods. Table 4.1 lists the overall best performing families of regression algorithms and for each family the best model, as suggested by Delgado et al. [12] in their extensive recent survey on the performance and effectiveness of regression methods. These methods present distinctive characteristics in their implementation, namely, they do not theoretically dominate each other. Hence, in order to answer the question "which regression method performs best for the period-inference problem", we implemented and investigated all of these methods to gather insights about their performance on our particular problem. We, however, anticipate to see that the tree-based solutions (*cubist*, *gbm*, *extraTrees*, *bartMachine*) have a better performance than *svr* and *avNNet* because we expect that the transition from a set of candidate periods (the features) to the true period to be better approximated by a set of rules and/or comparisons rather than a linear or non-linear combination of these features as in *svr* and *avNNet*, respectively.

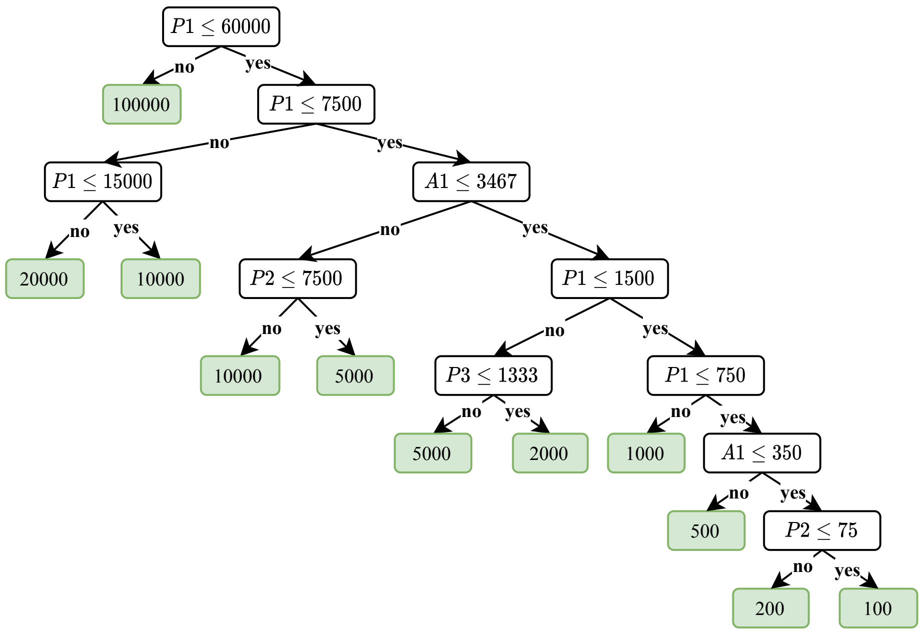

Figure 4.3: **Regression tree fitted on a data set of traces with 4 tasks and total utilization of 30% (see details in Section 5.1 for automotive task sets)**

Regression trees. A majority of the RBML methods in Table 4.1 are in fact variations of *regression trees*. A regression tree [5] recursively partitions the feature space of the data into smaller regions until the final sub-divisions are similar enough to be summarized by a simple model in a leaf. This model can be simply the average of the outcomes from that sub-division.

Figure 4.3 shows the rules generated by a regression tree that was trained on the automotive task sets with four periodic tasks and 30% utilization (see details of the task set generation in Section 5.1). The features used for training are the three highest peaks from the periodogram (denoted by $P1$, $P2$, and $P3$) and the autocorrelation (denoted by $A1, A2$, and $A3$) methods. The non-terminal nodes represents the *rules* that will be used to guide the inference process by narrowing down the period estimate of a new task.

To make it more tangible, we explain how to use the regression tree in Figure 4.3 to estimate the period of the two tasks in Figure 4.2(a) and (b). In the first step, we derive the three highest peaks of the periodogram and autocorrelation methods to build the feature vectors X_1 and X_2 for the first and second tasks, respectively. Here, $X_1 = \langle P1{=}1000, P2{=}500, P3{=}333, A1{=}1000, A2{=}2000, A3{=}3000\rangle$ and $X_2 = \langle P1{=}769, P2{=}666, P3{=}5000, A1{=}4635, A2{=}10000, A3{=}5365\rangle$. Next, we traverse the tree by evaluating the rules starting from the root node. For example, for the first task, $P1 = 1000$ and hence the condition in the root node (i.e., $P1 \leq 60000$) is satisfied. Thus, we go to the right branch in the tree and repeat the process from there until we reach to a leaf. The value in the leaf is our period estimate. In this example, the trained model allows us to accurately estimate both tasks' period.

An interesting observation in Figure 4.3 is the exclusion of $A2$ and $A3$ in the tree's rules which basically means that these two features had no impact on the final period estimate. With a further investigation, we observed that typically in task sets with low utilization, the trained regression trees tend to be smaller and rules contain fewer features because there are less preemptions (noise) in the input. However, with an increase in utilization, the tree is forced to consider more features and even become deeper to keep the accuracy high.

Training a regression tree can be done in $O(m \cdot N \cdot \log N)$, where m is the number of features (in our case 6) and N is the number of samples (projections) used for training. Later in Section 5.8, we provide an evaluation on the runtime and memory consumption of various RBML methods.

Cubist Regression [25, 38, 39, 40]. It is a regression tree whose leaves embed linear regression models instead of simple 'estimates of the output'. The tree can be further reduced by combining or pruning the rules via collapsing the nodes of the trees into rules.

By training a cubist regression model on the same data-set as in Figure 4.3, we obtain the following rules:

1. **If** $(A1 \leq 2000)$ **then** return $P1$,
2. **If** $(P1 \leq 1250 \wedge A1 > 2000)$ **then** return 5000,
3. **If** $(P1 > 1250)$ **then** return $P1$.

In this example, we observe that while the rules and outputs rely on the top candidates of the periodogram, they are not limited to them. For example, rule 2 outputs the period 5000 which is not among the three top features of periodogram. The cubist regression uses these rules to compensates for projections where the periodogram is wrong.

Cubist regressions consume notably less memory than the regression trees (see Section 5.8 Figure 5.8(b)) and hence they might be a better choice when the solution must have low memory consumption and runtime. However, we also noticed a growth in the number of rules when they are trained on task sets with high utilization. This is due to the fact that the underlying regression tree from which the cubist regression rules are obtained, gets larger and deeper when the number of preemptions increases.

Generalized Boosting Regression [17, 15]. This algorithm is a regression tree-based solution which uses a committee of regression trees of fixed size. The initial prediction of the algorithm starts from a leaf, which contains the average value of the outcome variables (i.e., the periods). The next step is to compute the residuals of this initial prediction against the true output (true period). Next, a regression tree is fitted on the data, but having the previously computed residuals as the outcome variables. In order to preserve the generalization capabilities of the model, the results from the tree are multiplied by a constant value. Afterwards, the output from the tree is added to the initial leaf to obtain a new set of predictions, which are again used to compute residuals. The process is repeated until the maximum number of trees is reached.

The following three algorithms are also based on regression trees, their main differences residing in their construction, but not in their way of operation is similar to regression trees.

Extremely Randomized Regression Trees [45, 16]. The algorithm relies on a committee of regression trees for its predictions. However, when building the trees this method, instead of searching for a rule that minimizes the error, it randomly picks a rule for each feature and then chooses the one that provides the lowest the error. In this way, a randomized regression tree is much faster to compute than a regular regression tree.

Bayesian Additive Regression Tree [22, 9]. Similar to *gbm*, this method also relies on a group of trees, where each tree is fit on the residuals of the predictions from a previous tree. The major difference is that *bartMachine* is based on a probability model containing a set of priors for the tree structure and a likelihood for the leaves' values. This probability model is incorporated into a penalization factor which enforces the tree depth and the leaf values to be small. In this way, it ensures that the contribution of each tree is small enough so that the model does not overfit the data.

Averaged Neural Network [24, 41]. The technique involves a committee of five multilayer perceptrons having the same size, but trained using different random seeds. The seeds are used when first creating the untrained networks, in order to initialize the weights, thus, ensuring that each of the five networks will have a different starting point. After training, the resulting weights are used to build a function whose purpose is to map the input vector of features (i.e., the periods from periodogram and autocorrelation) to an output value (the estimated period). The network is set to have linear output neurons, which makes it suitable for regression. Finally, the predictions from the five networks are averaged to provide the final estimate.

Support Vector Regression [32, 10]. The goal of *svr* is to find a line or a hyperplane that is able to fit the most data points within a certain margin from it. Moreover, it can accommodate non-linear trends by fitting the line in a transformed feature space using a kernel function.

Sections 5.5, 5.6, 5.7, 5.8 and Chapter 6 provide further insights about the performance of these RBML methods.

4.4 Candidate Selection

As it can be seen in Figure 4.2, the true period of the sequence is indicated by one of the peaks of the periodogram and autocorrelation, although not always by the highest. Further investigations showed that, on the one hand, in a majority of projections, the true period is indeed among the peaks of periodogram and autocorrelation. However, it is hard to know which of those peaks just by looking at their power or rank. On the other hand, the RBML methods typically predict only an approximation of the real period which is not always equal to the true one (hence having a non-zero error in most cases). We then introduced a further pruning phase to the output of our RPM method and created a method called *RPM with period adjustment* (RPMPA).

RPMPA treats the RPM method as a referee which chooses the right period from a set of candidates. Namely, it first calculates the period estimate using the RPM method and then finds the closest period to this estimate from a fixed set of values gathered from the 20 highest peaks of each of the periodogram and autocorrelation (hence, 40 candidates in total). The number of candidates (i.e., 40) is a hand-tuned value and comes from experimenting on many task sets (see Section 5.4).

4.5 Improving Accuracy using Ternary Projections

Choosing from a pool of candidates can help in considerably increasing the performance. However, in the cases when the regression algorithm significantly deviates from the true period, the chosen candidate may be even further from the correct value. Thus, by ensuring that only the most relevant candidates are kept in the set, we can expect an even greater drop in error. The goal of our *space-pruning method* (SPM) is to derive a lower and an upper bound on the possible set of period values. These bounds allow us to remove the impossible period values from the candidate set generated from the highest 20 peaks of each of the periodogram and autocorrelation methods.

In order to prune impossible candidates and find an upper bound on the period, we use the extra information available in ternary projections, i.e., the idle times. Given that the underlying scheduling policy is work-conserving, if a task has accessed the resource anywhere between two idle times, it must have released a job somewhere between those idle times. For example, from Figure 2.2(d) we can infer that at least one job of τ_3 must have been released in the interval [10, 19] since there is at least a “1” in the task’s projection in this interval.

Our key idea to derive an upper bound on the task’s period is to traverse the ternary projection to find pairs of consecutive intervals separated by idle times in which the task has occupied the resource. We call them *effective intervals.* From each pair of effective intervals, we then obtain one upper bound on the task’s period. After traversing the whole projection, the smallest upper bound found is the value we use to prune the candidates obtained from the peaks of periodogram and autocorrelation.

Extracting effective intervals. Let x be a time instant at which $p_x = 1$ and $\exists z < x$ in the ternary projection such that $p_z = idle$, then the beginning of the effective interval that contains the time instant k is the latest idle time prior to the execution of τ_i, namely,

$$\begin{aligned} I^s_j(x) = \max\{k \mid z < k < x \ \wedge\ p_k = idle \ \wedge \\ \forall p_y,\ k < y < x,\ p_y \neq idle\}. \end{aligned} \tag{4.7}$$

For example, in Figure 2.2(c), the beginning of the effective interval that contains time instants 11, 13, or 16 is 9 while for time instant 24 is 19. By calculating (4.7) for any $p_x = 1$, one can obtain the starting points of all effective intervals in a projection. For the example in Figure 2.2(c), the starting point of the effective intervals are at times 9 and 19.

Calculating an upper bound for T_i. Let I^s_j and I^s_{j+1} be the starting points of two consecutive effective intervals for task τ_i. The latest time instant at which a job of τ_i has occupied the resource during interval I_j is obtained as follows

$$\begin{aligned} \mathit{fin}(I^s_j, I^s_{j+1}) = \max\{k \mid k \in [I^s_j, I^s_{j+1}) \ \wedge\ p_k = 1 \ \wedge \\ \forall p_y,\ k < y < I^s_{j+1},\ p_y \neq 1\}, \end{aligned} \tag{4.8}$$

where p_x is the value of the ternary projection at time x. For example, in Figure 2.2(c), *fin*(9, 19) = 15.

Theorem 1 *Given the start time of two consecutive effective intervals I_j^s and I_{j+1}^s for task τ_i, T_i must follow*

$$T_i \leq \mathit{fin}(I_j^s, I_{j+1}^s) - I_{j-1}^s. \tag{4.9}$$

Proof. By definition, there is at least one time instant in I_{j-1}^s in which task τ_i has occupied the resource. Hence, the earliest release time of τ_i in I_{j-1}^s cannot be smaller than I_{j-1}^s. Moreover, since the last moment at which the task has occupied the resource in the interval I_j is at $\mathit{fin}(I_j^s, I_{j+1}^s)$, the latest possible release time of a job of the task cannot be later than $\mathit{fin}(I_j^s, I_{j+1}^s)$. The period of the task is bounded by the time interval between the latest possible release time of the current job and the earliest possible release time of the previous job (i.e., I_{j-1}^s). Hence, (4.9) provides an upper bound on the inter-arrival time between two jobs of τ_i. □

Calculating a lower bound for T_i. Provided that we know that the projection does not contain any deadline misses, we obtain a lower bound on the period of task τ_i. To do so, we extract the largest interval L in which the task τ_i does not occupy the resource. If there is no deadline miss in the projection, then the length of the largest interval in which no job of task τ_i has occupied the resource is upper bounded by $|L| \leq 2 \cdot T_i$. Namely, the period of the task cannot be smaller than half of that interval. The reason is that, in the worst case, the largest interval that can be observed is between a job which executed right after its release and the next job that executed right before its deadline, resulting in a value slightly lower than two times the period. For example, in Figure 4.2(b), the interval [5200, 11300] is the largest interval in which no job of the task has occupied the resource. Hence, period of τ_i must be at least 3050. This allows us to remove a large number of peaks of the periodogram from the candidate set. As this example shows, periodogram can contain many peaks at small periods.

Since both the lower bound and the upper bound can be calculated at the same time and by passing through the projection only once, they have a a linear time complexity w.r.t. the length of the projection.

Algorithm 1 presents the pseudo-code for deriving the bounds required by SPM. This algorithm receives the ternary projection of a task and traverses this projection only once to calculate both the lower and upper bound on the period estimates. The purpose of the first part of the algorithm (lines 8-15) is to find the length of the current idle interval between two appearances of the task under analysis. When the next execution of the task is seen (line 19) the interval length Δ is compared against the lower bound to decide whether the value of the lower bound should be updated. The second part (lines 22-31) focuses on determining the starting point of the current effective interval, as well as the starting point of the previous effective interval. The latest time instant when a job was executing, stored in variable *fin*, is constantly updated within lines 16-17. Whenever a task's execution is observed between the current and previous idle times, the upper bound on the period estimate is updated in line 27 (according to (4.9)). Note that we are only interested in the smallest observed period to use it as an upper bound, hence line 27 uses a min operator.

Algorithm 1: Bound Deriving Algorithm

```
Input  : Ternary projection $P^T$
Output: Period bounds ($LB$, $UB$)
1  $prevIdle \leftarrow 0$
2  $fin \leftarrow 0$
3  $recentIdle \leftarrow 0$
4  $\Delta \leftarrow 0$
5  $LB \leftarrow 0$
6  $UB \leftarrow \infty$
7  for $k = 1$ to $|P^T|$ do
8      if $p_k = idle$ then
9          if $p_{k-1} \neq idle$ then
10             $\Delta \leftarrow 1$
11         end
12         else
13             $\Delta \leftarrow \Delta + 1$
14         end
15     end
16     if $p_k = 1$ then
17         $fin \leftarrow k$
18         if $p_{k-1} \neq 1$ then
19             $LB \leftarrow \max\{\Delta/2, LB\}$
20         end
21     end
22     if $p_k \neq idle \wedge p_{k-1} = idle$ then
23         if is the first idle time to be found then
24             $recentIdle \leftarrow k$
25         end
26         else if task ran between current idle and previous idle then
27             $UB \leftarrow \min\{fin - prevIdle, UB\}$
28             $prevIdle \leftarrow recentIdle$
29             $recentIdle \leftarrow k$
30         end
31     end
32 end
```

Since the algorithm requires to traverse the ternary projection only once, it has an $O(|P^T|)$ complexity.

Chapter 5

Experiments

We performed a series of experiments to answer our research questions: In order to answer **RQ1**, we want to determine whether our framework improves the accuracy w.r.t. the state of the art. As part of solving **RQ2**, we check how do various families of RBML methods compare against each other, and we also explore what are the tradeoffs between the accuracy, runtime, and the memory requirements for these methods. **RQ3** is evaluated by employing the spatial-pruning method (SPM) on cases when period adjustment step (RPMPA) failed to improve the accuracy. Our last research question (**RQ4**) is answered by examining the robustness of our solution against uncertainties and non-deterministic events.

We divide our task systems into three categories: periodic task systems where every task is periodic but tasks might have release jitter or execution time variation (Section 5.5), non-periodic task systems, where the task under analysis is periodic but the rest of the system might not be periodic (Section 5.5), and case studies from actual systems (Section 5.7).

5.1 Experimental Setup

For the experiments in Section 5.5 and 5.6, we consider two types of task sets: automotive benchmark applications and synthetic task sets. For the automotive benchmark applications, we adopt the model proposed by Kramer et al. [23], where task periods are chosen randomly from {1, 2, 5, 10, 20, 50, 100, 200, 1000}ms with a non-uniform distribution provided in [23]. For simplicity, we refer to the traces of these task sets as *automotive traces.* Our synthetic task sets are comprised of non-harmonic periods. In order to ensure that the chosen periods cover evenly all magnitudes, we use a log-uniform distribution as suggested and described by Emberson et al. [13]. The periods are thereby generated for the range [100, 10000] with a base period of 100ms. For simplicity, we refer to the traces of these task sets as *log-uniform traces.* We use Strafford's Randfixedsum algorithm [13] to generate random utilization values for the tasks and then use the utilization and the period to calculate the WCET of each task.

The traces for both cases are generated using *Simso* [7], an open source and flexible simulation tool to generate schedules under various scheduling policies and setups.

The data-set used for building the regression models is composed of the projections from 2000 traces. The length of a trace is set to be either 6 hyperperiods (traces without random variations) and 10 hyperperiods (execution time variation and release jitter). The errors are calculated by using 5-fold cross-validation. Therefore, the data-set is randomly split into five subsets of equal size. Out of the five subsets, four are used for training and one is used for testing. The process is repeated until all five subsets have been used once for testing.

We compare the results against three baselines:

- PeTaMi, a mining algorithm for periodic tasks [21];
- periodogram [43];
- autocorrelation [18].

PeTaMi represents the state of the art on period inference in the real-time systems community, while the other two represent widely used solutions from the signal-processing literature. These two are chosen to evaluate the improvements made by our RPM and RPMPA over solutions that are (only) based on signal-processing techniques.

We compare the RBML methods mentioned in Table 4.1, denoted by *cubist* [40], *gbm* [15], *avNNet* [41], *extraTrees* [16], *bartMachine* [9], and *svr* [10]. Each of these methods is defined by a set of hyperparameters that require tuning for improving the model's fit on the data. Hence, we perform an additional tuning phase using random search on the parameter's space. This step is integrated in the cross-validation process such that every training set comprised of the four subsets, is further split into a training and validation set. The parameters are varied while being trained on the training set and the model's performance is estimated on the validation set. The purpose of doing one more split is to avoid bias by not involving the test set into the parameter choice.

The metric we use to evaluate the accuracy is the average error, which is the mean of the individual errors a method makes for every period in the test set unless it is explicitly stated that the error has been obtained for only one task in the task set. Moreover, to be able to focus on the accuracy of the RBML methods, we only show the results of RPM method (e.g., in Figures 5.1, 5.2, and 5.8) unless it is explicitly mentioned (e.g., in Figures 5.4(a, b, c) and Table 5.3).

We performed our evaluation on Cartesius, a Dutch supercomputer based in the cloud. We used thin nodes with 2 $\times$ 16-core 2.6 GHz Intel Xeon E5-2697A v4 (Broadwell) and 64 GB of memory.

5.2 Using the Baselines

Although for the two signal processing techniques (i.e., periodogram and autocorrelation) there are available resources that implement them, PeTaMi had no open-source implementation available. Therefore, we reproduced the part of the work of Iegorov et al. [21] which included the period mining algorithm. However, in order to be compliant to the input required by PeTaMi, we also designed a method to extract inter-arrival times from a binary projection.

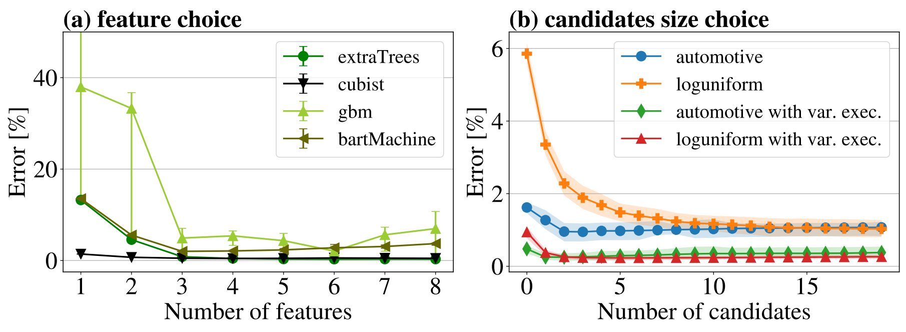

Figure 5.1: **The impact of the number of features (for RPM) and the number of candidates (for RPMPA) on the solution's accuracy. Note that the shade around the curves represents the confidence intervals for 0.95 confidence level.**

5.3 Additions to Simso

Although Simso encompasses a plethora of essential features, it did not include support for certain setups that we wanted to observe. As a consequence, we extended the framework to cover cases for:

- uniformly distributed variable execution time;
- release time jitter;
- sporadic tasks;
- random missed jobs;
- non-preemptive execution.

5.4 Parameter Tuning

Before evaluating our solutions, we need to determine their parameters, i.e., the number of features for RPM and the number of candidates for RPMPA, since they impact the solution's accuracy. The evaluation from Figure 5.1(a) was performed on an aggregated data set, containing automotive traces with four levels of utilization (30%, 50%, 70% and 90%). Similarly, for the second experiment from Figure 5.1 we used datasets incorporating the four utilization values and we also kept 20% execution time variation for the tasks in both data sets. We only picked the tree-based solutions since only these algorithms showed to be able to learn from the data. Thus, they were the only algorithms that would exhibit a trend w.r.t. the number of features. For *svr* and *avNNet* we kept the same number of features as we chose for the tree-based algorithms. The experiments were conducted by generating 20 random splits of the data set into training and testing sets (for every parameter value). The model would then be fit on the training data and the average error measured on the test data.

Figure 5.1(a) shows how the error for the tree-based algorithms decreases when we include more features. However, we notice that after adding more than three features from periodogram and autocorrelation, the gain in accuracy becomes insignificant. Thus, we kept three features from each of the periodogram and autocorrelation (i.e., six in total). Also, it is interesting to note that *cubist* is able to show good performance even for as few as two features (one from each signal processing technique). This behavior is due to the fact that *cubist* is a rule-based solution, thus, when it is trained only on the highest peak from the periodogram and from the autocorrelation, it is able to mimic the signal processing technique which minimizes the error, namely the periodogram.

We further analyzed the contribution of increasing the pool of candidates for RPMPA method on the accuracy. As it can be observed from Figure 5.1(b), a relatively small number of candidates is required in order to achieve a low error until it reaches saturation.

5.5 Assessing Accuracy in Periodic Systems

5.5.1 Impact of system utilization

Figures 5.2(a) and 5.2(b) show the average error as a function of the total utilization for task sets with 8 tasks. We notice that every regression model shows increased error with the increase in the utilization since the total utilization of the system has a direct impact on the number of preemptions. Furthermore, we observe a dramatic upward shift in PeTaMi's accuracy when it is applied on log-uniform traces, from 4.35% error on automotive traces to 640% error on log-uniform traces. This increased error is caused by the fact that the periods of the tasks generated from a log-uniform distribution are not harmonic. However, the regression algorithms are not significantly affected when handling non-harmonic periods. To support the previous statement, we show in Figure 5.3(a) the error relative to each period from the automotive benchmark. We notice that PeTaMi presents an error of 27.3% and of 96.4% for periods of 5 and 50 ms, respectively, while our solutions keep the error below 1% for 5ms and below 5% for 50ms. The aforementioned period values are the only periods that are not divided by all smaller periods, thus, they are the only non-harmonic periods in the automotive traces.

5.5.2 Impact of the number of tasks

Figures 5.2(c) and 5.2(d) illustrate how the error changes when increasing the number of tasks. We observe here a reduction of the error when having more tasks in the system for log-uniform traces in the case of some of the tree-based solutions (from 1.84% for 4 tasks to 1.23% for 16 tasks for *bartMachine*). The pattern is due to the decrease in the individual task utilization. Thus, although the system is as congested, the individual projection of a task contains larger idle intervals and shorter execution times that are likely not preempted much. This enables the periodogram to extract more meaningful features. However, in automotive task sets, the algorithms are rather unaffected by the number of tasks in the trace since they already have a good performance ($< 1\%$ error) even for lower number of tasks.

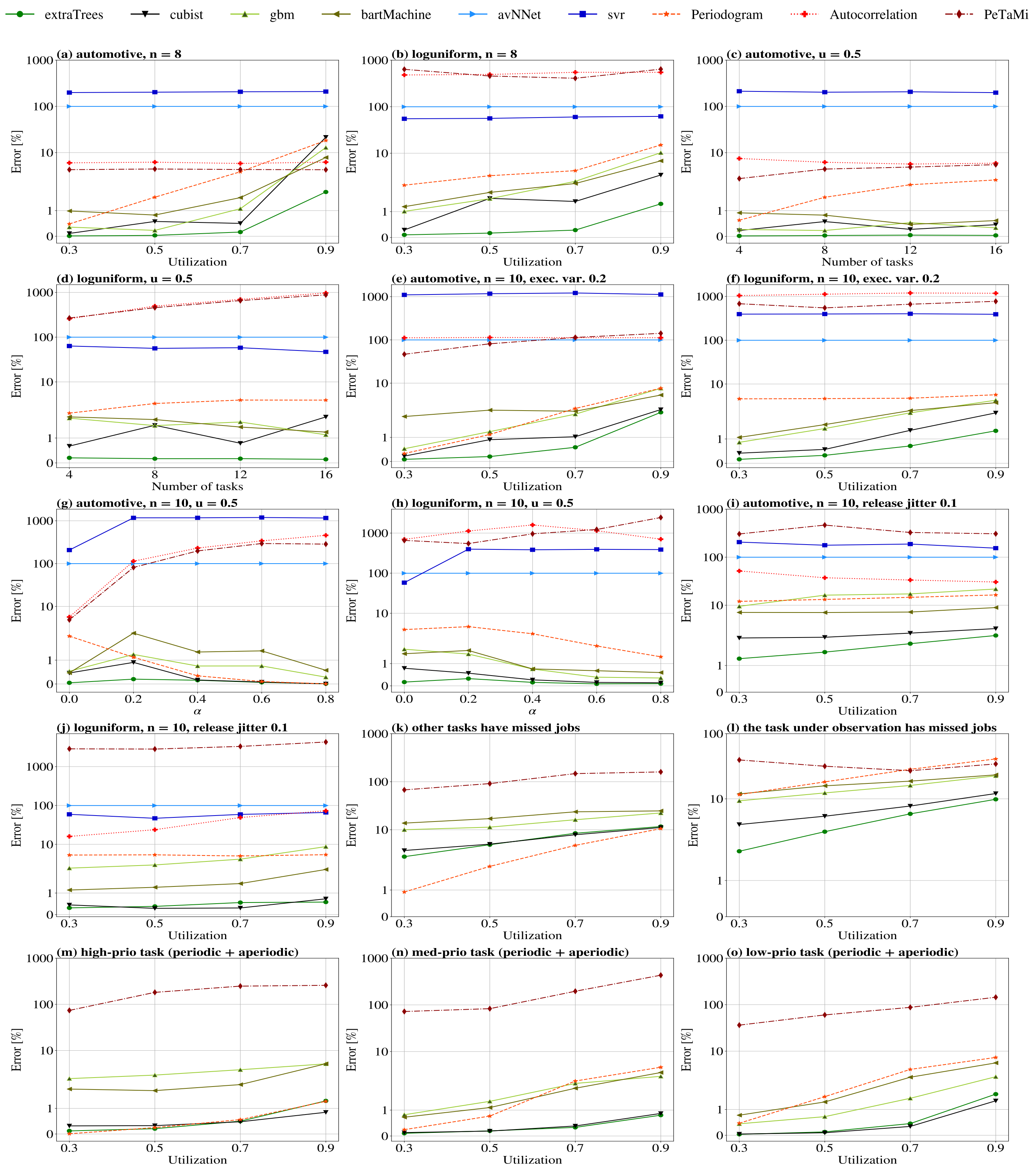

Figure 5.2: **Experimental results**

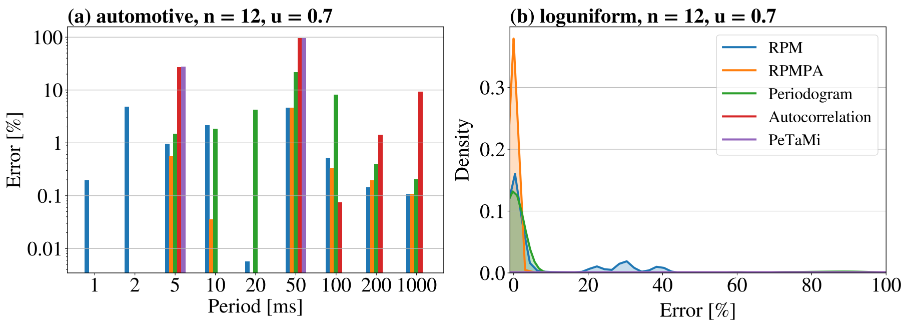

Figure 5.3: **Error distribution for RPM based on *cubist*.**

5.5.3 Impact of execution time variations

From Figures 5.2(e) and 5.2(f), we observe that the regression methods (0.69% average error over all task sets for *extraTrees*) are more robust to runtime execution time variations than the baselines (3.92% periodogram, 625% autocorrelation, 381% PeTaMi), showing a similar trend for both types of traces to the case with constant execution time. Also, when we increase the variation in execution in Figures 5.2(g) and 5.2(h), we notice that most of the RBML methods show robustness w.r.t. to this variation for automotive traces (0.11% average error over all task sets for *extraTrees*, 0.65% periodogram, 137% autocorrelation, and 142% PeTaMi), while for log-uniform traces the error decreases with the increase in variation (from 0.98% for 20% variation to 0.3% for 80% variation). This behavior is due to the reduction in the average execution time for individual tasks. Since the execution time for a job is drawn from a uniform distribution in the range $[(1-\alpha) \times WCET, WCET]$, the wider the interval becomes the lower is the average execution time of each task. Having lower execution time results in having lower utilization for the system and we previously observed that the methods perform better for lower utilization values.

5.5.4 Impact of release jitter

Figures 5.2(i) and 5.2(j) show that release time jitter has a much more negative impact on the error than the execution time variation. One possible explanation is that the periodogram, which provides most of the information to the algorithms, is negatively impacted by jitter, thus, it produces less useful features for training. However, we observe that some regression algorithms such as *extraTrees* (1.09% average error over all task sets) and *cubist* (1.49% average error over all task sets) are still able to keep a low error even for this challenging scenario.

5.5.5 Impact of candidate adjustment method (RPMPA)

Figures 5.4(a) and 5.4(b) show that RPMPA has about 60% less error than RPM for most RBML methods when used for cases with execution time variation and, implicitly, on ideal traces too. Also, from Figure 5.3(b) we notice how this method shifts the error distribution of log-uniform traces even more towards

			Error [%]	
Setup	Model	Data set	RPM	RPMPA

n = 8
U = 0.7
	cubist	automotive	0.5013	0.3297

		log-uniform	1.3927	0.6383
	gbm	automotive	1.0734	0.5879
		log-uniform	2.4894	0.8243
	extraTrees	automotive	0.1639	0.1339
		log-uniform	0.2853	0.2137
	bartMachine	automotive	1.5045	1.092
		log-uniform	2.2353	1.061

n = 10
U = 0.7
α = 0.2
	cubist	automotive	1.0279	0.5765

		log-uniform	1.3534	0.7911
	gbm	automotive	1.9674	0.8059
		log-uniform	2.1963	1.0936
	extraTrees	automotive	0.5882	0.3538
		log-uniform	0.7122	0.7508
	bartMachine	automotive	2.2421	1.2111
		log-uniform	2.4703	1.3406
robustness U = 0.7 med. prio	*cubist*	automotive	0.3871	0.2542
	gbm	automotive	2.0731	0.6527
	extraTrees	automotive	0.3327	0.0951
	bartMachine	automotive	1.8421	0.5076
job miss U = 0.7 current task	*cubist*	automotive	7.7464	5.4036
	gbm	automotive	16.0373	11.0846
	extraTrees	automotive	5.8743	4.4529
	bartMachine	automotive	18.6747	13.9603

Table 5.1: **The effect of period adjustment.**

0%, most of the errors being below 5%. We also observe that unlike RPM, our RPMPA does not result in error values that are as large as 20% to 40%. However, when the signal processing techniques (periodogram and autocorrelation) are disturbed, e.g., by release jitter, RPMPA results in worse error than RPM.

Table 5.1 shows the reduction in error generated by RPMPA for setups including ideal traces, execution time variation, robustness to higher priority aperiodic jobs, and to job misses. Also, we chose the parameters of the setups such that they will not represent extreme cases.

5.5.6 Impact of space-pruning method (SPM)

Figure 5.4(c) shows that the inclusion of an upper and a lower bound for SPM contributes to lowering the error even further, proving that the regression is still prone to mistakes even when choosing candidates. Also, it needs to be mentioned that under specific setups there can be cases where no candidates are left after the pruning phase. This situation occurs most frequently when the release time jitter disturbs so much the signal processing techniques that they only generate infeasible candidates. As a consequence, when analyzing the effect of release time jitter, we defined two criteria to provide a period estimate when no candidates are available. Therefore, we either select the output of regression

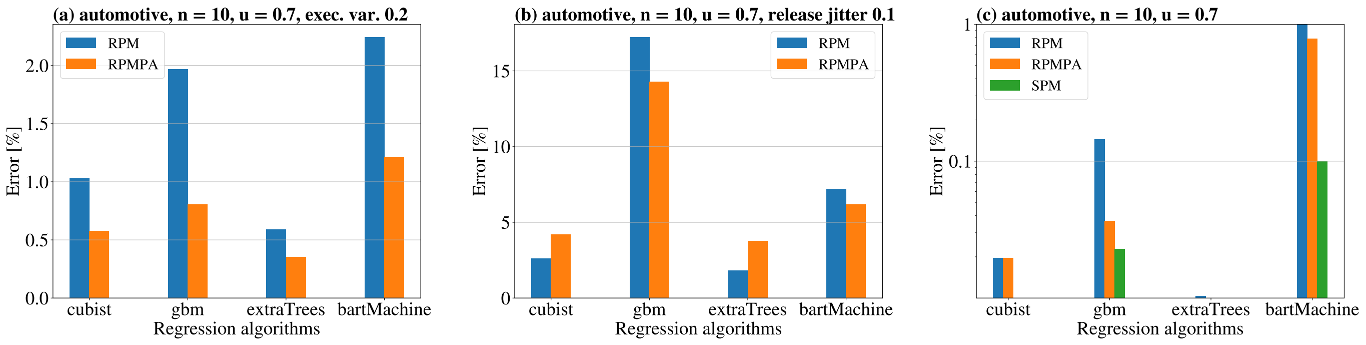

Figure 5.4: **(a)-(b) the impact of period adjustment method, and (c) the impact of space-pruning.**

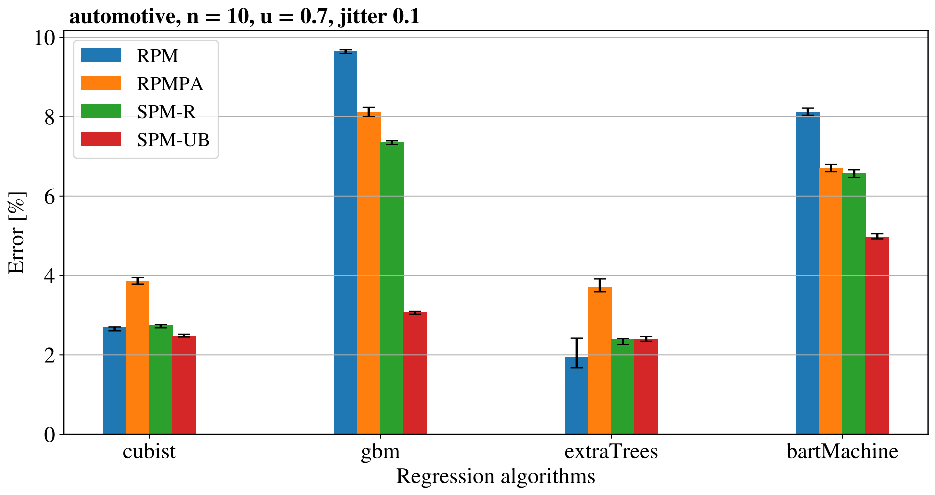

Figure 5.5: **Space-pruning for traces with jitter.**

(SPM-R), or we select the upper bound (SPM-UB).

The results for SPM on traces with jitter are shown in Figure 5.5. We notice that in all cases, both versions of SPM succeed in reducing the error of RPMPA up to 45%. Also, SPM-UB is able to achieve an average error below RPM for *cubist*, *gbm*, and *bartMachine*, while for *extraTrees*, although it has a larger error, it presents a much narrower confidence interval. Thus, we can expect that the estimate of SPM-UB based on *extraTrees* to be more reliable than the corresponding RPM.

Considering the results, we can conclude that the upper bound becomes a better estimate than the regression for the cases when there are no available candidates. This behavior is explainable since the cases without candidates appear in situations when the signal processing techniques are affected the most by jitter. Thus, the features that they provide to the regression algorithms are affected too, negatively influencing the prediction. However, the upper bound alone cannot replace the RPM methods since it can be too pessimistic as it

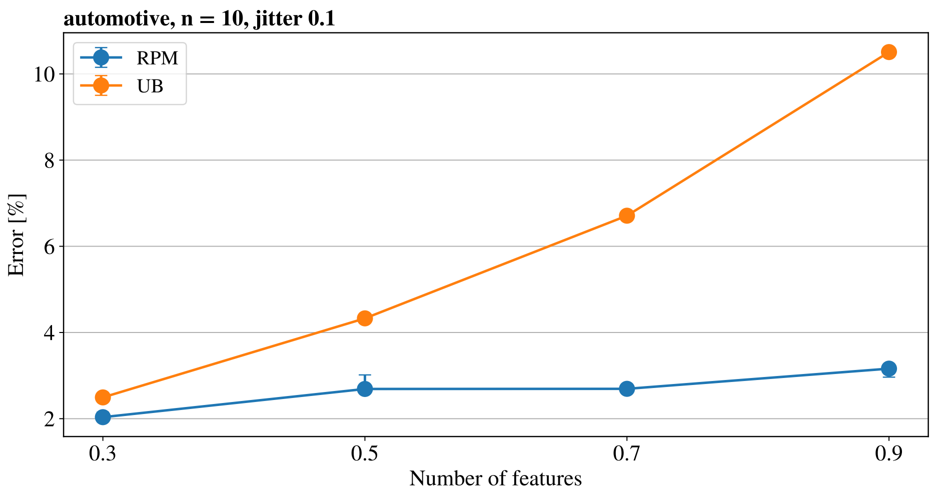

Figure 5.6: ***cubist* against upper bound.**

can be observed from Figure 5.6. The figure illustrates the evolution of the error with the increase in utilization. We notice that only considering the upper bound (UB) as an estimate results in a more drastic increase in the error than the RPM with *cubist*.

5.6 Assessing Robustness

5.6.1 Robustness w.r.t. the presence of higher-priority aperiodic tasks

Another step in our evaluation framework was to test the robustness of our methods for a setup that mimics possible real-world challenges. Thus, we created a configuration with 12 automotive tasks (6 periodic and 6 sporadic) scheduled by Rate Monotonic and under the interference of high-priority aperiodic tasks, arriving according to a Poisson process with a rate λ=0.0005 events/ns or 5 arrivals at every 10us. Furthermore, we focused on inferring the period of one periodic task in scenarios of having high, medium and low, priority, respectively. For each of the three priority scenarios, the task's priority has been chosen randomly in the ranges [1, 3] for high, [4, 7] for medium, and [8, 12] for low-priority tasks.

Figures 5.2(k), 5.2(l), 5.2(m) show the error as a function of utilization for the tree-based algorithms, periodogram and PeTaMi. We notice that the periodogram is affected by the change of priority, increasing its error with every lower level of priority. Although, this behavior signifies the generation of features with more noise, the RPM algorithms only suffer slight increases in their errors for larger utilization values, while *gbm* and *bartMachine* even show a decrease in error for lower utilization values when the priority of the task decreases. This aspect is related to the fact that these two algorithms may not be able to generalize well when the periodogram has small error. Having a small error for

Algorithm	Utilization	RM [%]	EDF [%]
cubist	0.3	0.1047	0.1046
	0.5	0.1122	0.1057
	0.7	0.1552	0.1554
	0.9	0.2165	0.2024
gbm	0.3	0.7145	0.7151
	0.5	0.8730	0.8731
	0.7	0.9558	0.9493
	0.9	1.0915	1.0951
extraTrees	0.3	0.3024	0.3026
	0.5	0.3951	0.3953
	0.7	0.4891	0.4821
	0.9	0.8702	0.9175
bartMachine	0.3	0.5947	0.5929
	0.5	0.6332	0.6316
	0.7	0.9501	0.9509
	0.9	1.3092	1.3155

Table 5.2: **Scheduling policy robustness for log-uniform traces with 10 tasks and 50% execution time variation.**

periodogram means having less significant (shorter) peaks, which in turn do not provide enough information for these algorithms to excel.

When it comes to the period adjustment step, Table 5.1 indicates substantial improvements, generating errors up to 3.5 times lower than RPM for a setup when the task we observed has a medium priority level.

5.6.2 Robustness w.r.t. dropping jobs

Next, we explored the impact of having missed jobs in the input projections on the accuracy of our solutions. The setup included 10 automotive tasks. For the first scenario, we assumed that the tasks under analysis have dropped jobs (with a 15% probability of dropping a job), while for the second one, all the other tasks in the system were dropping jobs, except for the task we observed.

Figures 5.2(n) and 5.2(o) show a larger error exhibited by all algorithms compared to the other experiments when tasks drop jobs (and hence, projections are not perfect). Moreover, we observe that the periodogram is clearly affected when the task under analysis drops jobs. However, while the RBML methods (RPM) show little variations from one case to the other, they are still able to retain meaningful information from their features even when the task under analysis has a low utilization.

The effect of missing jobs on the periodogram is also shown in Table 5.1 by the performance of RPMPA. There, we observe that although the period adjustment stage is still able to reduce the error, the overall error is still considerably higher than for the previously studied cases.

5.6.3 Robustness w.r.t. the scheduling policy

One other aspect we took into account was the robustness of our method when there is a change in the scheduling policy. Hence, we developed a scenario in which we trained the algorithms on traces of task sets coming from both rate

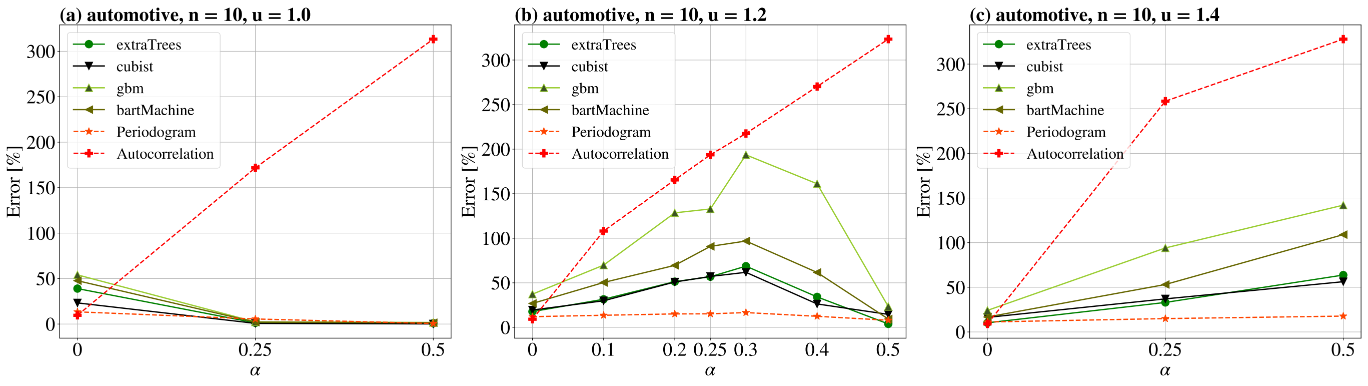

Figure 5.7: **The impact of different levels tardiness.**

monotonic (RM) and earlier-deadline first (EDF) scheduling policies. Afterwards, we tested the regression models on separate test sets, each containing traces only from RM and EDF, respectively. Also, since the periods coming from a log-uniform distribution are not harmonic, we chose log-uniform traces for this experiment. Since for harmonic periods (as in automotive traces), RM generates an almost identical schedule to EDF, it was not useful to consider those task sets in this experiment.

Table 5.2 shows that all considered regression algorithms have a very similar performance, regardless of the utilization. This means that we can safely employ our solutions for systems that are using either of the two studied scheduling policies.

5.6.4 Robustness w.r.t. tardiness

For this case, we experimented with task sets whose total utilization exceeded 100%. A consequence of overloading this bound is that lower-priority tasks will experience *starvation* (i.e., these tasks will not get any time slice on the processor). Since the projections of starved tasks do not include any information about their periodicity, we excluded such tasks from both the training and testing phase.

Figure 5.7 illustrates the effect of tardiness on the performance of the four tree-based regression algorithms and on the two signal processing techniques, for different levels of execution time variation. In Figure 5.7(a) we observe a reduction of the error with respect to the variation factor α when the total utilization of the system is 100%. This trend is due to the decrease in the average utilization with the increase of α. For instance, for $\alpha = 0.5$, the execution time of the tasks will be uniformly drawn from [0.5 × WCET, WCET], which implies an average execution of 0.75 × WCET, hence, the total utilization will be around 75% rather than 100%. However, in Figure 5.7(c) we experience an opposite trend for 140% utilization. In this case, the lower error when there is no execution time variation is due to the elimination of the tasks suffering from starvation, while the increase in error for larger α appears since the random reduction of execution time, which happens with variation, allows previously starved tasks to find chances of execution. However, since these chances appear

Data set	Algorithm	RPM [%]	RPMPA [%]
Car hacking data set [27]	*extraTrees*	28.2568	1.6179
	cubist	**3.2461**	**0.9703**
	gbm	15.8224	1.3919
	bartMachine	20.8588	14.0103
	Periodogram	5.3368	-
CAN intrusion data set [44]	*extraTrees*	13.0256	**1.7039**
	cubist	**5.5005**	2.8963
	gbm	19.8341	9.8881
	bartMachine	14.956	5.0264
	Periodogram	9.5448	-

Table 5.3: **Results on the two CAN data sets.**

randomly, the resulting projections would not be consistent enough to become meaningful training samples for the algorithms to learn. Also, both levels of execution time variation indicate a total utilization larger than 100%, which does not provide chances for all the tasks in the system to run. Figure 5.7(b) reflects the combination of the observations for the previous two cases. Until 30% execution time variation, the total utilization does not fall under 100%, hence Figure 5.7(b) shows a similar trend to Figure 5.7(c). However, when α increases, the curve becomes similar to Figure 5.7(a), since now the system allows more systematic running intervals for the lower-priority tasks.

5.7 Case Study

In this section we validate our period inference methods on two case studies from actual systems. We use two data-sets consisting of traces coming from Controller Area Networks (CANs) [19], denoted by *Car hacking data set* [27] and *CAN intrusion data set* [44] in Table 5.3.

Car hacking data set is a data set consisting of CAN traces coming from vehicles that underwent different types of attacks such as fuzzy attacks, denial-of-service or spoofing. However, we were only interested in the attack-free traces since checking the robustness for systems under attack is beyond the scope of this work. Thus, we collected 988,987 messages with 27 tasks. The unique periods of the the tasks were {10, 20, 50, 100, 1000}ms. Also, we need to mention that the periods of the tasks were not reported in the data set, hence, we manually analyzed the timestamps of the messages in order to deduct their periods.

CAN intrusion data set is another attack-based CAN data set coming from a KIA SOUL automobile. Similarly to the previous data set, we only extracted attack-free traces, which totaled 2,369,868 messages with 45 tasks. In this case, the range of periods included {10, 20, 100, 200, 1000}ms and were also manually identified.

For both data sets, the available information included: the timestamp of the message, the message ID, the size of the message, and the data that was sent.

In order to generate our test data, we needed a set of projections of the tasks in the systems. However, no information about the execution time of the tasks was included into the data set. Thus, when building a projection, we generated

Method	Average Runtime [s]
Candidate generation	5.101
PeTaMi	10.731

Table 5.4: **Runtime comparison.**

a fixed execution time proportional with the size of the message. Furthermore, from all the timestamps corresponding to a task, we subtracted the timestamp of the first occurrence of that task in the system, such that the origin of our projection will start at zero. Finally, from every resulting timestamp, we added a set of ones of length equal to the previously chosen execution time to finalize the projection. We were able to do so since we know that a CAN network follows a non-preemptive execution model [4], thus, once started, a task will run to completion.

Having the projections from the messages, we further split them into smaller projections of 100 jobs. As for our training data, we synthetically generated traces that would provide a good proxy for real data. Thus, we created a data set of 6000 automotive traces, with 20 tasks scheduled by non-preemptive rate monotonic, with 50% utilization and 5% jitter. The results from Table 5.3 show that our methods successfully estimated the periods of the messages on the actual use case, showing errors below 2% for both data sets.

5.8 Evaluating Runtime and Memory Requirements

Figure 5.8(a) shows how the increase in utilization impacts the number of rules generated by *cubist* for log-uniform traces with 12 tasks. We observe that the number of rules grows with the utilization. This behavior is expected since with the increase in utilization, comes an increase in complexity for the shape of the projection. Thus, the 16 rules that accurately described the traces with 30% utilization are not sufficient for larger utilization values since projections with the same period can appear increasingly different due to preemptions.

The memory consumption of each regression algorithm is shown in 5.8(b). The values were measured on trained models by using `object.size()`, a built-in function of R programming language. The results indicate a large gap between *cubist* and the other methods, which is expected since *cubist* only needs to store in its model the rules by which it estimates the periods, while the other tree-based solutions (*extraTrees*, *gbm*, *bartMachine*) need to reserve memory for all the regression trees they have built to make predictions.

The final experiment we performed was a runtime comparison between PeTaMi and the candidate generation from both the periodogram and autocorrelation. For every trace we used in the experiments from Section 5.5, we measured the amount of time required for both signal processing techniques to generate 20 period candidates from a projection. Also, we measured the duration of applying PeTaMi on each binary projection.

Table 5.4 shows that PeTaMi requires, on average, twice the amount of time needed by the two signal processing methods to obtain 20 candidates (denoted as *Candidate generation* in Table 5.4).

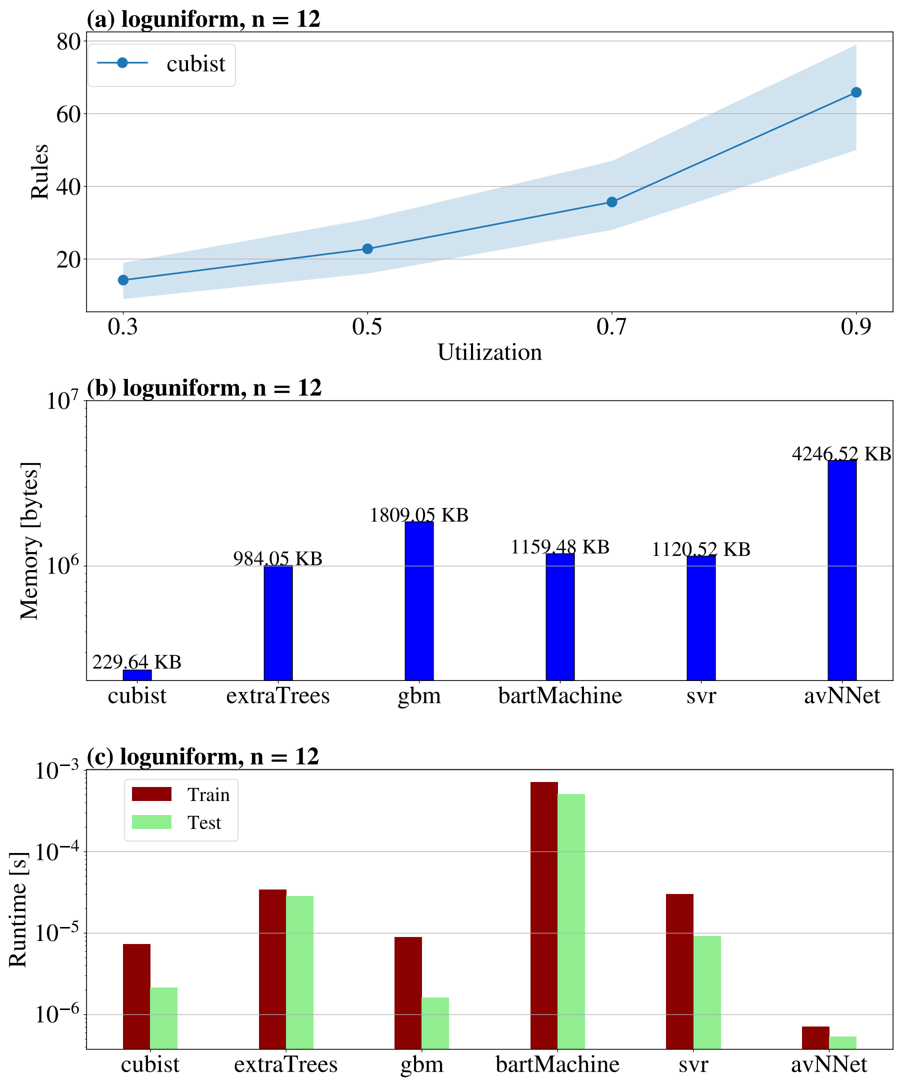

Figure 5.8: **(a) the number of rules in a trained *cubist* method and its confidence intervals for a confidence level of 0.95, (b) memory consumption, and (c) runtime of various regression algorithms.**

Chapter 6

Discussions

This chapter highlights the benefits and the downsides of the considered solutions. We elaborate on the implications of choosing a specific regression algorithm in terms of accuracy, memory consumption, and runtime.

6.1 Tree-based Solutions

Among our experiments, we noticed that *extraTrees* algorithm kept the smallest error in almost all the setups. However, when tested on a real dataset (Section 5.7), its generalization capability dropped to an error of 28.5% on the *Car hacking dataset*, the largest among all non-tree based algorithms. This makes *extraTrees* a good choice when we have access to training data from the same system we want to test on. However, when the target system is unknown, or we do not have access to the traces, *cubist* regression is a better choice, being the only algorithm that reached an error below the periodogram for both CAN datasets. This remarkable performance of *cubist* is due to its particularity of having linear regressions as output instead of means, as the other tree-based algorithms have.

6.2 Non-tree-based Solutions

The experiments have confirmed our statement from Section 4.3, where we expected the tree-based solutions to outperform the other types of algorithms, when applied to our problem. We noticed that *avNNet* has almost a constant yet very large error of 100% for all the experiments. This expresses the inability of the algorithm to learn a non-linear function that can map the input features to the target period, deciding instead to approximate the output as a constant value, namely the average of the periods from the training set. Since the test set is generated the same way as the training set, it will have a similar average value for its periods. Hence, the average error of *avNNet* will present a similar value regardless of the utilization of the tasks within the data set, since the estimate of *avNNet* is a value close to the mean of the periods of each test set.

Furthermore, *svr* is also not a good choice for the PI problem since our feature space is rather sparse, namely the features from periodogram and autocorrelation do not have values that place the data points close to each other in this space. Hence, *svr* is not able to find a suitable hyperplane to fit the data.

6.3 Memory Consumption and Runtime

Figure 5.8(b) addresses both the memory requirements, while Figure 5.8(c) shows the runtime of the six considered RBML methods. We notice that *cubist* has a considerably low memory consumption compared to the rest of the algorithms. Also, when it comes to runtime, *cubist* has the smallest runtime for training among the well-performing algorithms, and is only surpassed by a small margin by *gbm* in testing runtime. On the other end of the spectrum, *bartMachine* has the largest runtimes for both training and testing, while *avNNet* has the largest demand for memory among the considered solutions.

Also, we observed from Table 5.4 that PeTaMi is slower than the candidate generation step. Since the features that we use for the regression models reside among the generated candidates, and also that the runtime required for RBML to generate a prediction is less than 1ms in the worst case, we can conclude that regression-based period mining (RPM) also shows better efficiency than the state of the art. Moreover, the period adjustment of RPMPA requires just an additional selection step from the list of 40 candidates, which is almost like a constant complexity, i.e., O(1), in comparison to the candidate generation runtime.

When it comes to our space-pruning method (SPM), the generation of the valid period bounds happens in $O(p)$ time, which is faster than the $O(p \log p)$ time of the signal processing techniques, where p is the length of the projection. Since the generation of the candidates was twice as fast as PeTaMi, we can expect that the addition of SPM would still keep the runtime of our solution below the one of the state of the art.

6.4 Period Adjustment

It is important to note the benefit of adding the candidate adjustment step, which can drastically reduce the error of RPM (up to 3 times reduction for *gbm* in Table 5.1). However, special attention must be paid to cases with large jitter, where this additional step is negatively impacted. Also, if the target application indulges a linear-time complexity overhead, then the limitation of RPMPA for traces with jitter can be attenuated by employing SPM.

Chapter 7

Conclusions

7.1 Summary

In this work, we introduced the first regression-based machine learning (RBML) solution for the problem of inferring a task's period from its binary projections. From the projections we generated features using the periodogram [43] and autocorrelation [18] signal processing techniques. Hence, we developed a regression-based period miner (RPM) by using the features to train regression models. We investigated six most-successful and widely used families of RBML methods for this problem and provided comprehensive evaluations and discussions about their accuracy and robustness under various scenarios.

We proposed further steps for improving the accuracy by creating period-adjustment (RPMPA) and space-pruning methods (SPM) that use the properties of a work-considering scheduler to prune the space of valid periods of a task. Our solutions proved to be robust and highly accurate. The average observed error of our (best) solution was under 1% in most scenarios including those with a mixture of periodic, aperiodic, and sporadic tasks, execution time variation, and release jitter while the existing work has two to three orders-of-magnitude higher errors. On the case studies from actual systems, the error of our best solution was below 1.7%.

7.2 Answers to Research Questions

Throughout our work we have tried to answer the following research questions:

RQ1. *Can one infer the tasks' periods from traces of a real-time system using RBML (regression-based machine learning) techniques? If so, how effective (in terms of accuracy) and efficient (in terms of runtime) would those methods be?*

We have introduced the first RBML solution for the period inference problem. In comparison to the state of the art, our solutions are two to three orders of magnitude more accurate and have half the runtime of PeTaMi, the state-of-the-art algorithm, in the best case (for RPM) or their runtime is comparable to ours in the worst case (for SPM).

RQ2. *What impact does the choice of the RBML algorithm have on the effect-*

iveness of our solution?

This research question was addressed by employing six regression algorithms (*cubist*, *gbm*, *extraTrees*, *bartMachine*, *svr*, and *avNNet*, introduced in Table 4.1) that showed the most promising performance in a very recent and thorough survey of RBML methods by Delgado et al. [12]. We noticed that the tree-based solutions, namely *cubist*, *gbm*, *extraTrees*, and *bartMachine*, were the only methods that were able to successfully learn to solve the period inference problem. Also, among these algorithms we notice that cubist has remarkable generalization capabilities (3.25% and 5.5% error for real traces in Table 5.3), low memory requirements (229KB for the trained model in Figure 5.8(b), while the second lowest value was more than 4 times larger, 984.05 KB for *extraTrees*), and fast runtime in both training and testing stages (below 10μs for both stages in Figure 5.8(c)).

RQ3. *Can we derive a set of pruning rules to further restrict the number of possible period values? If so, what will be the impact of adding the set of rules over the RBML performance in terms of effectiveness?*

To answer this question, we derived a lower bound and an upper bound on the range of valid periods of the target task by incorporating additional information about the time intervals in which the system is idle. We further used these bounds to filter the candidates provided by the signal processing techniques and we proposed two methods to obtain an estimate for the cases when none of the candidates satisfied the bounds. The evaluation indicated that the space-pruning method (SPM) effectively addressed the shortcomings of the period adjustment step (RPMPA), reducing its error by up to 45% (Figure 5.5). Thus, SPM proved to be a more reliable period miner, especially for the cases with jitter.

RQ4. *How robust is our solution against the interferences caused by the non-determinism present during the operation of real systems?*

For the final research question, we evaluated four scenarios for the robustness of our solution, namely we tested the robustness to **the presence of higher-priority aperiodic tasks** in the system, to **dropping jobs**, to the **change in scheduling policy** and to **tardiness**. We noticed that the RPM methods, especially the ones using *cubist* and *extraTrees*, were resilient to most sources of interference (errors below 5% for execution time variation in Figures 5.2(e)-(h), below 7% for jitter in Figures 5.2(i)-(j), under 10% for missed jobs in Figures 5.2(k)-(l), and below 2% in the presence of aperiodic tasks in Figures 5.2(m)-(o)).

7.3 Future Work

Given the benefit of incorporating knowledge about the idle-time (space-pruning method), it would be interesting to investigate the advantages of including other sources of information on the period inference problem. One such source can be to know the intervals when a higher-priority task is running in the system. This knowledge may then provide a way to derive a tighter upper bound for the period estimation, which could further enhance the accuracy of RPM, particularly for the cases with jitter where the best performing approach was SPM based on the upper bound.

Furthermore, we would like to explore RBML methods to infer the timing properties of parallel applications running on multiprocessor platforms. Such cases present the challenge of having the jobs of the same task run on different processors, depending on the schedule. Thus, creating simple binary projections of the tasks running on one processor will not suffice anymore to generate meaningful features. A possible solution could include aggregating projections coming from different processors, however, timing inaccuracies can occur since the jobs may not be executed periodically on each processor.

Currently, our solutions require execution traces coming from real-time systems. A future research direction could involve exploring methods to determine the tasks' periods without needing to assume any information about the properties of the tasks in the system or their sequence of execution. A first step towards this goal was taken by Lamichhane et al. [26] with their work on determining the running task based on the power trace of the microcontroller. However, in our case, one would require to find periodic patterns with high confidence within the power trace and, then, map each individual pattern to a projection, out of which the period can be mined.

Bibliography

[1] Robert Bellman. Curse of Dimensionality. *Adaptive Control Processes: A Guided Tour*, 3:2, 1961.

[2] Tarak Benkedjouh, Noureddine Zerhouni, and Said Rechak. Tool Condition Monitoring Based on Mel-frequency Cepstral Coefficients and Support Vector Regression. In *International Conference on Electrical Engineering-Boumerdes (ICEE-B)*, pages 1–5, 2017.

[3] Christos Berberidis, Walid G Aref, Mikhail Atallah, Ioannis Vlahavas, and Ahmed K Elmagarmid. Multiple and Partial Periodicity Mining in Time Series Databases. In *European Conference on Artificial Intelligence (ECAI)*, pages 370–374, 2002.

[4] Robert Bosch et al. CAN Specification Version 2.0. *Rober Bousch GmbH, Postfach*, 300240:72, 1991.

[5] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and Regression Trees*. 1984.

[6] Giorgio C Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications*, volume 24. 2011.

[7] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. Simso: A Simulation Tool to Evaluate Real-time Multiprocessor Scheduling Algorithms. 2014.

[8] Kuo ching Liang, Xiaodong Wang, and Ta-Hsin Li. Robust Discovery of Periodically Expressed Genes Using the Laplace Periodogram. *BMC bioinformatics*, 10(1):15, 2009.

[9] Hugh A Chipman, Edward I George, Robert E McCulloch, et al. BART: Bayesian Additive Regression Trees. *The Annals of Applied Statistics*, 4(1):266–298, 2010.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[11] Greta Cutulenco, Yogi Joshi, Apurva Narayan, and Sebastian Fischmeister. Mining Timed Regular Expressions From System Traces. In *International Workshop on Software Mining (SoftwareMining)*, pages 3–10, 2016.

[12] Manuel Fernández Delgado, M. S. Sirsat, Eva Cernadas, Sadi Alawadi, Senén Barro, and Manuel Febrero-Bande. An Extensive Experimental Survey of Regression Methods. *Neural Networks*, 111:11–34, 2019.

[13] Paul Emberson, Roger Stafford, and Robert I Davis. Techniques for the Synthesis of Multiprocessor Task Sets. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–11, 2010.

[14] Chunguo Fei, Guoyuan Qi, and Chunxin Li. Fault Location on High Voltage Transmission Line by Applying Support Vector Regression with Fault Signal Amplitudes. *Electric Power Systems Research*, 160(1):173–179.

[15] Jerome H Friedman. Stochastic Gradient Boosting. *Computational Statistics & Data Analysis (CSDA)*, 38(4):367–378, 2002.

[16] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely Randomized Trees. *Machine Learning*, 63(1):3–42, 2006.

[17] Brandon Greenwell, Bradley Boehmke, Jay Cunningham, and GBM Developers. *gbm: Generalized Boosted Regression Models*, 2019. R package version 2.1.5.

[18] John A Gubner. *Probability and Random Processes for Electrical and Computer Engineers.* 2006.

[19] Hacking and Countermeasure Research Lab. `http://ocslab.hksecurity.net/Datasets/`, 2010. [Online; accessed 01-July-2020].

[20] Oleg Iegorov and Sebastian Fischmeister. Mining Task Precedence Graphs from Real Time Embedded System Traces. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 251–260, 2018.

[21] Oleg Iegorov, Reinier Torres, and Sebastian Fischmeister. Periodic Task Mining in Embedded System Traces. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 331–340, 2017.

[22] Adam Kapelner and Justin Bleich. bartMachine: Machine Learning with Bayesian Additive Regression Trees. *Journal of Statistical Software*, 70(4):1–40, 2016.

[23] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real World Automotive Benchmarks for Free. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[24] Max Kuhn. *caret: Classification and Regression Training*, 2020. R package version 6.0-86.

[25] Max Kuhn and Ross Quinlan. *Cubist: Rule- And Instance-Based Regression Modeling*, 2020. R package version 0.2.3.

[26] Kamal Lamichhane, Carlos Moreno, and Sebastian Fischmeister. Non-intrusive Program Tracing of Non preemptive Multitasking Systems Using Power Consumption. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1147–1150, 2018.

[27] H. Lee, S. H. Jeong, and H. K. Kim. OTIDS: A Novel Intrusion Detection System for In-vehicle Network by Using Remote Frame. In *Annual Conference on Privacy, Security and Trust (PST)*, volume 00, pages 57–5709, 2017.

[28] Cornelius T Leondes. *Computer Techniques and Algorithms in Digital Signal Processing: Advances in Theory and Applications.* 1996.

[29] Ta-Hsin Li. Detection and Estimation of Hidden Periodicity in Asymmetric Noise by Using Quantile Periodogram. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3969–3972, 2012.

[30] YB Malode, DB Khadse, and DV Jamthe. Efficient Periodicity Mining Using Circular Autocorrelation in Time Series Data. *International Research Journal of Engineering and Technology (IRJET)*, 2(3):430–436, 2015.

[31] Robby G McKilliam, I Vaughan L Clarkson, and Barry G Quinn. Fast Sparse Period Estimation. *IEEE Signal Processing Letters*, 22(1):62–66, 2014.

[32] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2019. R package version 1.7-3.

[33] M. Nasri, T. Chantem, G. Bloom, and R. M. Gerdes. On the Pitfalls and Vulnerabilities of Schedule Randomization Against Schedule-Based Attacks. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 103–116, 2019.

[34] Mitra Nasri, Geoffrey Nelissen, and Björn B Brandenburg. A Response-time Analysis for Non-preemptive Job Sets under Global Scheduling. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 9–16, 2018.

[35] Mukesh Patel and Nilesh Modi. A Comprehensive Study on Periodicity Mining Algorithms. In *International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pages 567–575, 2016.

[36] Maurice Bertram Priestley. *Spectral Analysis and Time Series Probability and Mathematical Statistics.* Number 4. 1981.

[37] Tom Puech, Matthieu Boussard, Anthony D'Amato, and Gaëtan Millerand. A Fully Automated Periodicity Detection in Time Series. In *International Workshop on Advanced Analysis and Learning on Temporal Data (AALTD)*, pages 43–54, 2019.

[38] J Quinlan. Learning with Continuous Classes. In *Australian Joint Conference on Artificial Intelligence (AI)*, volume 92, pages 343–348, 1992.

[39] J Quinlan. Combining Instance-based and Model-based Learning. In *International Conference on Machine Learning (ICML)*, pages 236–243, 1993.

[40] J Quinlan. *C4. 5: Programs for Machine Learning.* 2014.

[41] Brian D Ripley. *Pattern recognition and neural networks.* 2007.

[42] Mahmoud Salem, Mark Crowley, and Sebastian Fischmeister. Anomaly Detection using Inter-arrival Curves for Real-time Systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 97–106, 2016.

[43] Arthur Schuster. On the Investigation of Hidden Periodicities with Application to a Supposed 26 Day Period of Meteorological Phenomena. *Terrestrial Magnetism*, 3(1):13–41, 1898.

[44] E. Seo, H. M. Song, and H. K. Kim. GIDS: GAN based Intrusion Detection System for In-Vehicle Network. In *Annual Conference on Privacy, Security and Trust (PST)*, pages 1–6, 2018.

[45] Jaak Simm, Ildefons Magrans de Abril, and Masashi Sugiyama. *Tree-Based Ensemble Multi-Task Learning Method for Classification and Regression*, 2014. R package version 6.0-86.

[46] Gasper Slapničar, Mitja Luštrek, and Matej Marinko. Continuous Blood Pressure Estimation from PPG Signal. *Informatica*, 42(1):33–42.

[47] Ilia Sucholutsky, Apurva Narayan, Matthias Schonlau, and Sebastian Fischmeister. Deep Learning for System Trace Restoration. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[48] Poornima Unnikrishnan and V Jothiprakash. Daily Rainfall Forecasting for One Year in a Single Run Using Singular Spectrum Analysis. *International Journal of Hydrology (IJH)*, 561(1):609–621, 2018.

[49] Michail Vlachos, Philip Yu, and Vittorio Castelli. On Periodicity Detection and Structural Periodic Similarity. In *SIAM International Conference on Data Mining (SDM)*, pages 449–460, 2005.

[50] Guofeng Wang, Lei Qian, and Zhiwei Guo. Continuous Tool Wear Prediction based on Gaussian Mixture Regression Model. *The International Journal of Advanced Manufacturing Technology (IJAMT)*, 66(9-12):1921–1929, 2013.

[51] Xianjun Xia, Roberto Togneri, Ferdous Sohel, and David Huang. Random Forest Regression-based Acoustic Event Detection with Bottleneck Features. In *International Conference on Multimedia and Expo (ICME)*, pages 157–162, 2017.

[52] Clinton Young, Habeeb Olufowobi, Gedare Bloom, and Joseph Zambreno. Automotive Intrusion Detection Based on Constant CAN Message Frequencies Across Vehicle Driving Modes. In *ACM Workshop on Automotive Cybersecurity*, page 9–14, 2019.